CS471 Project 2

Generated by Doxygen 1.8.13

Contents

1	Nam	espace	Index														1
	1.1	Names	space List					 	 	 		 	 				1
2	Hier	archica	l Index														3
	2.1	Class	Hierarchy					 	 	 		 	 				3
3	Clas	s Index															5
	3.1	Class	List					 	 	 		 	 				5
4	File	Index															7
	4.1	File Lis	st					 	 	 		 	 				7
5	Nam	nespace	Docume	ntatio	n												9
	5.1		Namespa			е.		 	 	 	 	 	 				9
		5.1.1	Enumera														9
			5.1.1.1		orithm .												9
	5.2	mdata	Namespa	_													10
	5.3	mfunc	Namespa	ice Re	ference) ·		 	 	 		 	 		 		10
		5.3.1	Detailed														11
		5.3.2	Typedef	Docur	nentati	on .		 	 	 		 	 				11
			5.3.2.1		ncPtr .												11
		5.3.3	Variable	Docui	mentati	on .		 	 	 		 	 				11
			5.3.3.1	NUN	M_FUN	СТІО	NS	 	 	 		 	 				11
	5.4	util Na	mespace f														11
		5.4.1	Function	n Docu	mentat	ion		 	 	 		 	 		 		12
			5.4.1.1	alloc	cArray())		 	 	 		 	 				12
			5.4.1.2	allo	cMatrix(() -		 	 	 		 	 				13
			5.4.1.3		yArray()												13
			5.4.1.4	initA	rray() .			 	 	 		 	 				14
			5.4.1.5		/latrix()												15
			5.4.1.6	relea	aseArra	ay()		 	 	 		 	 				15
			5417	rele:	aseMat	rix()											16

ii CONTENTS

6	Clas	s Docu	mentation	17
	6.1	enums	s::AlgorithmNames Struct Reference	17
		6.1.1	Detailed Description	17
		6.1.2	Member Function Documentation	17
			6.1.2.1 get()	18
		6.1.3	Member Data Documentation	18
			6.1.3.1 BLIND_SEARCH	18
			6.1.3.2 LOCAL_SEARCH	18
	6.2	mdata	::BlindSearch< T > Class Template Reference	19
		6.2.1	Detailed Description	19
		6.2.2	Member Function Documentation	20
			6.2.2.1 run()	20
	6.3	mdata	::DataTable < T > Class Template Reference	21
		6.3.1	Detailed Description	21
		6.3.2	Constructor & Destructor Documentation	22
			6.3.2.1 DataTable()	22
			6.3.2.2 ~DataTable()	22
		6.3.3	Member Function Documentation	22
			6.3.3.1 exportCSV()	23
			6.3.3.2 getColLabel()	24
			6.3.3.3 getEntry()	25
			6.3.3.4 setColLabel()	25
			6.3.3.5 setEntry()	26
	6.4	mfunc:	::Experiment< T > Class Template Reference	26
		6.4.1	Detailed Description	27
		6.4.2	Constructor & Destructor Documentation	27
			6.4.2.1 Experiment()	27
			6.4.2.2 ~Experiment()	28
		6.4.3	Member Function Documentation	28
			6.4.3.1 init()	28

CONTENTS

		6.4.3.2	testAllFunc()	30
		6.4.3.3	testFuncThreaded()	32
6.5	mfunc:	::Function[Desc Struct Reference	33
	6.5.1	Detailed	Description	33
	6.5.2	Member	Function Documentation	34
		6.5.2.1	get()	34
6.6	mfunc:	::Functions	s< T > Struct Template Reference	35
	6.6.1	Detailed	Description	36
	6.6.2	Member	Function Documentation	36
		6.6.2.1	ackleysOne()	36
		6.6.2.2	ackleysTwo()	37
		6.6.2.3	alpine()	37
		6.6.2.4	dejong()	38
		6.6.2.5	eggHolder()	39
		6.6.2.6	exec()	39
		6.6.2.7	get()	40
		6.6.2.8	griewangk()	41
		6.6.2.9	levy()	42
		6.6.2.10	mastersCosineWave()	42
		6.6.2.11	michalewicz()	43
		6.6.2.12	nthroot()	44
		6.6.2.13	pathological()	45
		6.6.2.14	quartic()	46
		6.6.2.15	rana()	46
		6.6.2.16	rastrigin()	47
		6.6.2.17	rosenbrok()	47
		6.6.2.18	schwefel()	48
		6.6.2.19	sineEnvelopeSineWave()	49
		6.6.2.20	step()	49
		6.6.2.21	stretchedVSineWave()	50

iv CONTENTS

		6.6.2.22 w()			 	 50
6.7	util::lni	Reader Class Reference .			 	 51
	6.7.1	Detailed Description			 	 51
	6.7.2	Constructor & Destructor E	Documentation		 	 52
		6.7.2.1 IniReader()			 	 52
		6.7.2.2 \sim IniReader() .			 	 52
	6.7.3	Member Function Docume	entation		 	 52
		6.7.3.1 entryExists() .			 	 52
		6.7.3.2 getEntry()			 	 53
		6.7.3.3 getEntryAs() .			 	 54
		6.7.3.4 openFile()			 	 54
		6.7.3.5 sectionExists()			 	 54
6.8	mdata	:LocalSearch< T > Class T	Template Reference	э	 	 55
	6.8.1	Detailed Description			 	 56
	6.8.2	Member Function Docume	entation		 	 56
		6.8.2.1 run()			 	 56
6.9	mdata	:Population< T > Class Ter	mplate Reference		 	 58
	6.9.1	Detailed Description			 	 59
	6.9.2	Constructor & Destructor E	Documentation		 	 59
		6.9.2.1 Population() .			 	 59
		6.9.2.2 ∼Population()			 	 60
	6.9.3	Member Function Docume	entation		 	 60
		6.9.3.1 calcFitness() .			 	 60
		6.9.3.2 generate()			 	 61
		6.9.3.3 getAllFitness()			 	 62
		6.9.3.4 getBestFitnessIn	ndex()		 	 62
		6.9.3.5 getBestFitnessF	Ptr()		 	 63
		6.9.3.6 getDimensionsS	Size()		 	 64
		6.9.3.7 getFitness() .			 	 65
		6.9.3.8 getFitnessPtr()			 	 66

CONTENTS

		6.9.3.9	getPopulationPtr()	66
		6.9.3.10	getPopulationSize()	67
		6.9.3.11	isReady()	67
		6.9.3.12	outputFitness()	68
		6.9.3.13	outputPopulation()	68
		6.9.3.14	setFitness()	69
6.10	mfunc:	:RandomB	Sounds < T > Struct Template Reference	70
	6.10.1	Detailed I	Description	70
	6.10.2	Member I	Data Documentation	70
		6.10.2.1	max	70
		6.10.2.2	min	71
6.11	mdata:	:SearchAlo	gorithm< T > Class Template Reference	71
	6.11.1	Detailed I	Description	72
	6.11.2	Construc	tor & Destructor Documentation	72
		6.11.2.1	SearchAlgorithm()	72
		6.11.2.2	~SearchAlgorithm()	72
	6.11.3	Member I	Function Documentation	72
		6.11.3.1	run()	73
		6.11.3.2	startTimer()	73
		6.11.3.3	stopTimer()	73
	6.11.4	Member I	Data Documentation	73
		6.11.4.1	timeDiff	74
		6.11.4.2	timer	74
6.12	mdata:	:TestParan	neters < T > Struct Template Reference	74
	6.12.1	Detailed I	Description	74
	6.12.2	Construc	tor & Destructor Documentation	75
		6.12.2.1	TestParameters()	75
	6.12.3	Member I	Data Documentation	75
		6.12.3.1	alg	75
		6.12.3.2	alpha	76

vi

		6.12.3.3 execTimesCol	 76
		6.12.3.4 execTimesRow	 76
		6.12.3.5 execTimesTable	 76
		6.12.3.6 funcld	 77
		6.12.3.7 resultsCol	 77
		6.12.3.8 resultsRow	 77
		6.12.3.9 resultsTable	 77
6.13	mdata:	::TestResult< T > Struct Template Reference	 78
	6.13.1	Detailed Description	 78
	6.13.2	Constructor & Destructor Documentation	 78
		6.13.2.1 TestResult()	
	6.13.3	Member Data Documentation	 78
		6.13.3.1 err	
		6.13.3.2 execTime	 79
		6.13.3.3 fitness	79
6.14		dPool Class Reference	79
		Detailed Description	80
	6.14.2	Constructor & Destructor Documentation	80
		6.14.2.1 ThreadPool()	80
		6.14.2.2 ~ThreadPool()	81
	6.14.3	Member Function Documentation	81
		6.14.3.1 enqueue()	81
		6.14.3.2 stopAndJoinAll()	 81

CONTENTS vii

7	File I	Docum	entation							83
	7.1	include	/blindsearch.h	File Reference		 	 	 	 	83
		7.1.1	Detailed Desc	ription		 	 	 	 	84
	7.2	blindse	arch.h			 	 	 	 	85
	7.3	include	/datatable.h Fil	e Reference		 	 	 	 	85
		7.3.1	Detailed Desc	ription		 	 	 	 	87
	7.4	datatak	le.h			 	 	 	 	87
	7.5	include	/experiment.h I	File Reference		 	 	 	 	88
		7.5.1	Detailed Desc	ription		 	 	 	 	90
	7.6	experir	nent.h			 	 	 	 	90
	7.7	include	/inireader.h File	Reference .		 	 	 	 	91
		7.7.1	Detailed Desc	ription		 	 	 	 	92
	7.8	iniread	er.h			 	 	 	 	93
	7.9	include	/localsearch.h	File Reference		 	 	 	 	93
		7.9.1	Detailed Desc	ription		 	 	 	 	95
		7.9.2	Macro Definiti	on Documenta	ıtion	 	 	 	 	95
			7.9.2.1 DEG	C_PRECISION	1	 	 	 	 	95
	7.10	localse	arch.h			 	 	 	 	95
	7.11	include	/mem.h File Re	ference		 	 	 	 	97
		7.11.1	Detailed Desc	ription		 	 	 	 	99
	7.12	mem.h				 	 	 	 	99
	7.13	include	/mfuncptr.h File	Reference .		 	 	 	 	100
		7.13.1	Detailed Desc	ription		 	 	 	 	102
	7.14	mfuncp	tr.h			 	 	 	 	102
	7.15	include	/mfunctions.h F	ile Reference		 	 	 	 	102
		7.15.1	Detailed Desc	ription		 	 	 	 	104
		7.15.2	Macro Definiti	on Documenta	ıtion	 	 	 	 	105
			7.15.2.1 _ac	kleysOneDesc		 	 	 	 	105
			7.15.2.2 _ac	kleysTwoDesc		 	 	 	 	105
			7.15.2.3 _alp	oineDesc		 	 	 	 	105

viii CONTENTS

	7.15.2.4 _dejongDesc	105
	7.15.2.5 _eggHolderDesc	105
	7.15.2.6 _griewangkDesc	106
	7.15.2.7 _levyDesc	106
	7.15.2.8 _mastersCosineWaveDesc	106
	7.15.2.9 _michalewiczDesc	106
	7.15.2.10 _pathologicalDesc	106
	7.15.2.11 _quarticDesc	107
	7.15.2.12 _ranaDesc	107
	7.15.2.13 _rastriginDesc	107
	7.15.2.14 _rosenbrokDesc	107
	7.15.2.15 _schwefelDesc	107
	7.15.2.16 _sineEnvelopeSineWaveDesc	108
	7.15.2.17 _stepDesc	108
	7.15.2.18 _stretchedVSineWaveDesc	108
	7.15.2.19 _USE_MATH_DEFINES	108
7.16	6 mfunctions.h	109
7.17	' include/population.h File Reference	114
	7.17.1 Detailed Description	116
7.18	population.h	116
7.19	include/searchalg.h File Reference	117
	7.19.1 Detailed Description	118
7.20	searchalg.h	119
7.21	include/stringutils.h File Reference	120
	7.21.1 Detailed Description	121
7.22	? stringutils.h	121
7.23	include/testparam.h File Reference	122
	7.23.1 Detailed Description	123
7.24	testparam.h	123
7.25	include/testresult.h File Reference	124

CONTENTS ix

	7.25.1 Detailed Description	124
7.26	testresult.h	125
7.27	include/threadpool.h File Reference	125
7.28	threadpool.h	126
7.29	src/experiment.cpp File Reference	127
	7.29.1 Macro Definition Documentation	128
	7.29.1.1 INI_FUNC_RANGE_SECTION	128
	7.29.1.2 INI_TEST_ALGORITHM	128
	7.29.1.3 INI_TEST_ALPHA	129
	7.29.1.4 INI_TEST_DIMENSIONS	129
	7.29.1.5 INI_TEST_EXECTIMESFILE	129
	7.29.1.6 INI_TEST_ITERATIONS	129
	7.29.1.7 INI_TEST_NUMTHREADS	129
	7.29.1.8 INI_TEST_POPULATION	130
	7.29.1.9 INI_TEST_RESULTSFILE	130
	7.29.1.10 INI_TEST_SECTION	130
7.30	experiment.cpp	130
7.31	src/inireader.cpp File Reference	136
	7.31.1 Detailed Description	137
7.32	inireader.cpp	137
7.33	src/main.cpp File Reference	139
	7.33.1 Detailed Description	139
	7.33.2 Function Documentation	140
	7.33.2.1 main()	140
	7.33.2.2 runExp()	140
7.34	main.cpp	141
7.35	src/population.cpp File Reference	142
	7.35.1 Detailed Description	142
7.36	population.cpp	143
Index		147

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

enums										 	 													S
mdata											 													10
mfunc										 	 							 						10
util																								- 11

2 Namespace Index

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

enums::AlgorithmNames	17
$ndata::DataTable < T > \dots $	21
$nfunc :: Experiment < T > \ \dots \$	26
nfunc::FunctionDesc	33
$nfunc :: Functions < T > \dots $	35
ıtil::IniReader	51
ndata::Population <t></t>	58
$\label{eq:func::RandomBounds} \textit{I} > \dots $	
ndata::SearchAlgorithm < T >	71
$mdata::BlindSearch < T > \dots \dots$. 19
$mdata:: Local Search < T > \dots \dots$. 55
ndata::TestParameters $<$ T $>$	74
ndata::TestResult< T >	78
ThreadPool	79

4 Hierarchical Index

Chapter 3

Class Index

3.1 Class List

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

enums::AlgorithmNames	
Struct that contains constant string names for the different search algorithms	17
mdata::BlindSearch< T >	
The BlindSearch class implements the Blind Search algorithm, which is ran using the overridden	
SearchAlgorithm::run() function	19
mdata::DataTable< T >	
The DataTable class is a simple table of values with labeled columns	21
mfunc::Experiment < T >	
Contains classes for running the CS471 project experiment	26
mfunc::FunctionDesc	
Get() returns a function's description Returns a C-string description for the given function id if the	
id is valid. Otherwise returns null	33
mfunc::Functions< T >	
Struct containing all static math functions. A function can be called directly by name, or indirectly	
using Functions::get or Functions::exec	35
util::IniReader	
Simple ∗.ini file reader and parser	51
mdata::LocalSearch< T >	
The LocalSearch class implements the Local Search algorithm, which is ran using the overridden	
SearchAlgorithm::run() function	55
mdata::Population< T >	
Data class for storing a multi-dimensional population of data with the associated fitness	58
mfunc::RandomBounds< T >	
Simple struct for storing the minimum and maximum input vector bounds for a function	70
mdata::SearchAlgorithm< T >	
The SearchAlgorithm class is used as a base class for other implemented search algorithms.	
Provides a common interface to run each algorithm	71
mdata::TestParameters< T >	
Packs together various test experiment parameters	74
$mdata:: TestResult < T > \dots \dots$	78
ThreadPool	79

6 Class Index

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

include/blindsearch.h	
Implements the BlindSearch class, which inherits SearchAlgorithm. BlindSearch::run executes	
the blind search algorithm on a given population	83
include/datatable.h	
Header file for the DataTable class, which represents a spreadsheet/table of values that can	
easily be exported to a *.csv file	85
include/experiment.h	
Header file for the Experiment class. Contains the basic logic and functions to run the cs471	
project experiment	88
include/inireader.h	
Header file for the IniReader class, which can open and parse simple $*.$ ini files $\ldots\ldots\ldots$	91
$include/local search.h \\ \ldots \\ $	93
include/mem.h	
Header file for various memory utility functions	97
include/mfuncptr.h	
Contains the type definition for mfuncPtr, a templated function pointer to one of the math func-	
tions in mfunctions.h	100
include/mfunctions.h	
Contains various math function definitions	102
include/population.h	
Header file for the Population class. Stores a population and resulting fitness values	114
include/searchalg.h	
Defines the SearchAlgorithm class, Algorithm enum, and AlgorithmNames struct. The Search ←	
Algorithm class serves as a base class for implemented search algorithms	117
include/stringutils.h	
Contains various string manipulation helper functions	120
include/testparam.h	
Contains the definition of the TestParameters struct, which is a data type used to transfer test	
parameters between functions	122
include/testresult.h	
Simple structure that packs together various return values for the search algorithms. functions	124
include/threadpool.h	125
src/experiment.cpp	127
src/inireader.cpp	
Implementation file for the IniReader class, which can open and parse simple * ini files	136

8 File Index

src/main.cpp	
Program entry point. Creates and runs CS471 project 2 experiment	139
src/population.cpp	
Implementation file for the Population class. Stores a population and fitness values	142

Chapter 5

Namespace Documentation

5.1 enums Namespace Reference

Classes

• struct AlgorithmNames

Struct that contains constant string names for the different search algorithms.

Enumerations

• enum Algorithm { BlindSearch = 0, LocalSearch = 1, Count = 2 }

Enum of different available search algorithms.

5.1.1 Enumeration Type Documentation

5.1.1.1 Algorithm

enum enums::Algorithm

Enum of different available search algorithms.

Enumerator

BlindSearch	
LocalSearch	
Count	

Definition at line 28 of file searchalg.h.

```
00030 BlindSearch = 0,
00031 LocalSearch = 1,
00032 Count = 2
```

5.2 mdata Namespace Reference

Classes

class BlindSearch

The BlindSearch class implements the Blind Search algorithm, which is ran using the overridden SearchAlgorithm

∴run() function.

class DataTable

The DataTable class is a simple table of values with labeled columns.

class LocalSearch

The LocalSearch class implements the Local Search algorithm, which is ran using the overridden SearchAlgorithm ::run() function.

· class Population

Data class for storing a multi-dimensional population of data with the associated fitness.

· class SearchAlgorithm

The SearchAlgorithm class is used as a base class for other implemented search algorithms. Provides a common interface to run each algorithm.

struct TestParameters

Packs together various test experiment parameters.

struct TestResult

5.3 mfunc Namespace Reference

Classes

class Experiment

Contains classes for running the CS471 project experiment.

struct FunctionDesc

get() returns a function's description Returns a C-string description for the given function id if the id is valid. Otherwise returns null

struct Functions

Struct containing all static math functions. A function can be called directly by name, or indirectly using Functions::get or Functions::exec.

· struct RandomBounds

Simple struct for storing the minimum and maximum input vector bounds for a function.

Typedefs

```
template < class T >
using mfuncPtr = T(*)(T *, size_t)
```

Function pointer that takes two arguments T* and size_t, and returns a T value.

Variables

• constexpr const unsigned int NUM_FUNCTIONS = 18

5.3.1 Detailed Description

Scope for all math functions

5.3.2 Typedef Documentation

5.3.2.1 mfuncPtr

```
template<class T >
using mfunc::mfuncPtr = typedef T (*)(T*, size_t)
```

Function pointer that takes two arguments T* and size_t, and returns a T value.

Template Parameters

```
T Data type for vector and return value
```

Definition at line 28 of file mfuncptr.h.

5.3.3 Variable Documentation

5.3.3.1 NUM_FUNCTIONS

```
constexpr const unsigned int mfunc::NUM_FUNCTIONS = 18
```

Constant value for the total number of math functions contained in this namespace

Definition at line 47 of file mfunctions.h.

 $Referenced \ by \ mfunc:: Experiment < T > :: testAllFunc(), \ and \ mfunc:: Experiment < T > :: testFuncThreaded().$

5.4 util Namespace Reference

Classes

· class IniReader

The IniReader class is a simple *.ini file reader and parser.

Functions

```
• template<class T = double>
  void initArray (T *a, size_t size, T val)
      Initializes an array with some set value.
• template < class T = double >
  void initMatrix (T **m, size_t rows, size_t cols, T val)
      Initializes a matrix with a set value for each entry.
• template < class T = double >
  void releaseArray (T *&a)
      Releases an allocated array's memory and sets the pointer to nullptr.
```

```
• template < class T = double >
  void releaseMatrix (T **&m, size_t rows)
```

Releases an allocated matrix's memory and sets the pointer to nullptr.

• template<class T = double>

```
T * allocArray (size_t size)
```

Allocates a new array of the given data type.

• template < class T = double >

```
T ** allocMatrix (size t rows, size t cols)
```

Allocates a new matrix of the given data type.

• template < class T = double > void copyArray (T *src, T *dest, size_t size)

Copies the elements from one equal-sized array to another.

5.4.1 Function Documentation

5.4.1.1 allocArray()

```
template<class T = double>
T* util::allocArray (
            size_t size ) [inline]
```

Allocates a new array of the given data type.

Template Parameters

Data	type of the array

Parameters

Returns

Returns a pointer to the new array, or nullptr allocation fails

Definition at line 108 of file mem.h.

5.4.1.2 allocMatrix()

Allocates a new matrix of the given data type.

Template Parameters

Data	type of the matrix entries
------	----------------------------

Parameters

rows	The number of rows
cols	The number of columns

Returns

Returns a pointer to the new matrix, or nullptr if allocation fails

Definition at line 122 of file mem.h.

```
T** m = (T**)allocArray<T*>(rows);
00125
              if (m == nullptr) return nullptr;
00126
00127
00128
              for (size_t i = 0; i < rows; i++)</pre>
00129
                 m[i] = allocArray<T>(cols);
00130
                  if (m[i] == nullptr)
00131
00132
00133
                      releaseMatrix<T>(m, rows);
                      return nullptr;
         }
00134
                  }
00135
00136
00137
00138 }
            return m;
```

5.4.1.3 copyArray()

Copies the elements from one equal-sized array to another.

Template Parameters

Parameters

5	src	Source array from where the elements will be copied from	
(dest	Destination array from where the elements will be copied to	
5	size	Number of elements in the array	

Definition at line 149 of file mem.h.

5.4.1.4 initArray()

Initializes an array with some set value.

Template Parameters

Data	type of array

Parameters

а	Pointer to array
size	Size of the array
val	Value to initialize the array to

Definition at line 29 of file mem.h.

Referenced by initMatrix().

5.4.1.5 initMatrix()

Initializes a matrix with a set value for each entry.

Template Parameters

Data	type of matrix entries
------	------------------------

Parameters

m	Pointer to a matrix	
rows	Number of rows in matrix	
cols	Number of columns in matrix	
val	Value to initialize the matrix to	

Definition at line 49 of file mem.h.

References initArray().

5.4.1.6 releaseArray()

Releases an allocated array's memory and sets the pointer to nullptr.

Template Parameters

Data	type of array
	, ,,

Parameters

```
a Pointer to array
```

Definition at line 66 of file mem.h.

5.4.1.7 releaseMatrix()

Releases an allocated matrix's memory and sets the pointer to nullptr.

Template Parameters

Data type	of the matrix
-----------	---------------

Parameters

m	Pointer th the matrix
rows	The number of rows in the matrix

Definition at line 82 of file mem.h.

Referenced by mdata::DataTable < T >::~DataTable().

```
00083
00084
                 if (m == nullptr) return;
00085
00086
                 for (size_t i = 0; i < rows; i++)</pre>
00087
00088
                      if (m[i] != nullptr)
00089
                     {
                           // Release each row
releaseArray<T>(m[i]);
00090
00091
00092
00093
00094
00095
                 }
                 // Release columns
delete[] m;
00096
00097
                 m = nullptr;
00098
```

Chapter 6

Class Documentation

6.1 enums::AlgorithmNames Struct Reference

Struct that contains constant string names for the different search algorithms.

```
#include <searchalg.h>
```

Static Public Member Functions

• static const char * get (Algorithm alg)

Static Public Attributes

- static constexpr const char * BLIND_SEARCH = "Blind Search"
- static constexpr const char * LOCAL_SEARCH = "Local Search"

6.1.1 Detailed Description

Struct that contains constant string names for the different search algorithms.

Definition at line 39 of file searchalg.h.

6.1.2 Member Function Documentation

18 Class Documentation

6.1.2.1 get()

Definition at line 44 of file searchalg.h.

References enums::BlindSearch, and enums::LocalSearch.

Referenced by mfunc::Experiment< T >::init().

```
00045
00046
                    switch (alg)
00047
                        case Algorithm::BlindSearch:
    return BLIND_SEARCH;
00048
00049
                       case Algorithm::LocalSearch:
00050
00051
                            return LOCAL_SEARCH;
00052
                        default:
                        return "";
break;
00053
00054
00055
                    }
00056
              }
```

6.1.3 Member Data Documentation

6.1.3.1 BLIND_SEARCH

```
constexpr const char* enums::AlgorithmNames::BLIND_SEARCH = "Blind Search" [static]
```

Definition at line 41 of file searchalg.h.

6.1.3.2 LOCAL_SEARCH

```
constexpr const char* enums::AlgorithmNames::LOCAL_SEARCH = "Local Search" [static]
```

Definition at line 42 of file searchalg.h.

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

• include/searchalg.h

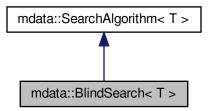
6.2 mdata::BlindSearch < T > Class Template Reference

The BlindSearch class implements the Blind Search algorithm, which is ran using the overridden SearchAlgorithm

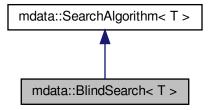
∷run() function.

#include <blindsearch.h>

Inheritance diagram for mdata::BlindSearch< T >:



Collaboration diagram for mdata::BlindSearch< T >:



Public Member Functions

virtual TestResult< T > run (mfunc::mfuncPtr< T > funcPtr, const T fMin, const T fMax, Population< T > *const pop, const T alpha)

Executes Blind Search with the given population and parameters.

Additional Inherited Members

6.2.1 Detailed Description

template < class T> class mdata::BlindSearch < T>

The BlindSearch class implements the Blind Search algorithm, which is ran using the overridden SearchAlgorithm

∴:run() function.

20 Class Documentation

Template Parameters

T	Data type used

Definition at line 27 of file blindsearch.h.

6.2.2 Member Function Documentation

6.2.2.1 run()

Executes Blind Search with the given population and parameters.

Parameters

funcPtr	Function pointer to the math function being used to generate the population
fMin	Minimum bound for the population matrix vector components
fMax	Maximum bound for the population matrix vector components
рор	Pointer to a population object that will be used in the blind search
alpha	Unused in this algorithm

Returns

TestResult <T > Returns a TestResult struct containing the error code, fitness, and execution time

Implements mdata::SearchAlgorithm< T >.

Definition at line 44 of file blindsearch.h.

References mdata::Population< T >::calcFitness(), mdata::Population< T >::generate(), mdata::Population< T >::getBestFitnessPtr(), mdata::Population< T >::getDimensionsSize(), mdata::Population< T >::getPopulation \leftarrow Size(), mdata::SearchAlgorithm< T >::startTimer(), and mdata::SearchAlgorithm< T >::stopTimer().

```
00045
00046
                     \ensuremath{//} Get population size and dimensions
                     size_t popSize = pop->getPopulationSize();
size_t dimSize = pop->getDimensionsSize();
00047
00048
00049
00050
                     // Make sure funcPtr is valid;
                     if (funcPtr == nullptr) return TestResult<T>(1, 0, 0.0); // Invalid function id, return with
00051
        error code 1
00052
00053
                     // Start recording execution time
00054
                     startTimer();
```

```
// Generate values for population vector matrix
00056
00057
                  pop->generate(fMin, fMax);
00058
00059
                  // For each population vector, calculate the fitness using the funcPtr
00060
                  for (size_t sol = 0; sol < popSize; sol++)</pre>
00061
00062
                      // Populate fitness values using given math function pointer
00063
                      if (!pop->calcFitness(sol, funcPtr))
                          return TestResult<T>(2, 0, 0.0); // Invalid fitness index, return with error code 2
00064
00065
00066
00067
                  // Return best fitness value in population
00068
                  return TestResult<T>(0, *pop->getBestFitnessPtr(), stopTimer());
00069
```

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

· include/blindsearch.h

6.3 mdata::DataTable < T > Class Template Reference

The DataTable class is a simple table of values with labeled columns.

```
#include <datatable.h>
```

Public Member Functions

DataTable (size_t _rows, size_t _cols)

Construct a new Data Table object Throws std::length_error and std::bad_alloc.

∼DataTable ()

Destroy the Data Table object.

std::string getColLabel (size_t colIndex)

Gets the string label for the column with the given index.

void setColLabel (size_t colIndex, std::string newLabel)

Sets the string label for the column with the given index.

• T getEntry (size_t row, size_t col)

Returns the value in the table at the given row and column.

void setEntry (size_t row, size_t col, T val)

Set the value for the table entry at the given row and column.

bool exportCSV (const char *filePath)

Exports the contents of this DataTable to a .csv file.

6.3.1 Detailed Description

```
template < class T > class mdata::DataTable < T >
```

The DataTable class is a simple table of values with labeled columns.

- Initialize a DataTable object with a specified number of rows and columns: DataTable table(rows, columns);

Set a column's label:

```
table.setColLabel(0, "Column 1");
```

Set an entry in the table:

table.setEntry(n, m, value);

Where 'n' is the row, 'm' is the column, and 'value' is the value of the entry

Export the table to a *.csv file:

bool success = table.exportCSV("my_file.csv");

Definition at line 50 of file datatable.h.

22 Class Documentation

6.3.2 Constructor & Destructor Documentation

6.3.2.1 DataTable()

Construct a new Data Table object Throws std::length_error and std::bad_alloc.

Parameters

_rows	Number of rows in table
_cols	Number of columns in table

Definition at line 60 of file datatable.h.

```
00060
00061
                                                       : rows(_rows), cols(_cols), dataMatrix(nullptr)
00062
                   if (rows == 0)
                   throw std::length_error("Table rows must be greater than 0.");
else if (cols == 0)
00063
00064
00065
                       throw std::length_error("Table columns must be greater than 0.");
00066
00067
                   dataMatrix = util::allocMatrix<T>(rows, cols);
                   if (dataMatrix == nullptr)
00068
00069
                       throw std::bad_alloc();
00070
00071
                   colLabels.resize(_cols, std::string());
00072
```

6.3.2.2 \sim DataTable()

```
template<class T>
mdata::DataTable< T >::~DataTable ( ) [inline]
```

Destroy the Data Table object.

Definition at line 77 of file datatable.h.

References util::releaseMatrix().

6.3.3 Member Function Documentation

6.3.3.1 exportCSV()

Exports the contents of this DataTable to a .csv file.

24 Class Documentation

Parameters

filePath Path to the file that will be filled with this table's values

Returns

true If the file was successfully written to false If there was an error opening the file

Definition at line 155 of file datatable.h.

Referenced by mfunc::Experiment< T >::testAllFunc().

```
00156
00157
                  if (dataMatrix == nullptr) return false;
00158
00159
                  using namespace std:
00160
                  ofstream outFile;
00161
                  outFile.open(filePath, ofstream::out | ofstream::trunc);
00162
                  if (!outFile.good()) return false;
00163
                  // Print column labels
00164
                  for (unsigned int c = 0; c < cols; c++)
00165
00166
                  {
00167
                       outFile << colLabels[c];</pre>
00168
                       if (c < cols - 1) outFile << ",";</pre>
00169
                  }
00170
00171
                  outFile << endl;
00172
00173
                   // Print data rows
00174
                   for (unsigned int r = 0; r < rows; r++)
00175
00176
                       for (unsigned int c = 0; c < cols; c++)
00177
00178
                           outFile << std::setprecision(8) << dataMatrix[r][c];</pre>
00179
                           if (c < cols - 1) outFile << ",";</pre>
00180
00181
                       outFile << endl;
00182
                  }
00183
00184
                  outFile.close();
00185
                  return true;
00186
```

6.3.3.2 getColLabel()

Gets the string label for the column with the given index.

Parameters

colIndex	Index of the column

Returns

std::string String value of the column label

Definition at line 88 of file datatable.h.

6.3.3.3 getEntry()

Returns the value in the table at the given row and column.

Parameters

row	Row index of the table
col	Column index of the table

Returns

T Value of the entry at the given row and column

Definition at line 117 of file datatable.h.

```
00118
00119
                  if (dataMatrix == nullptr)
                       throw std::runtime_error("Data matrix not allocated");
00120
                  if (row >= rows)
00121
                  throw std::out_of_range("Table row out of range");
else if (col >= cols)
00122
00123
                      throw std::out_of_range("Table column out of range");
00125
00126
                  return dataMatrix[row][col];
             }
00127
```

6.3.3.4 setColLabel()

Sets the string label for the column with the given index.

Parameters

collndex	Index of the column
newLabel	New string label for the column

Definition at line 102 of file datatable.h.

Referenced by mfunc::Experiment< T >::testAllFunc().

6.3.3.5 setEntry()

Set the value for the table entry at the given row and column.

Parameters

row	Row index of the table
col	Column index of the table
val	New value for the entry

Definition at line 136 of file datatable.h.

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

• include/datatable.h

6.4 mfunc::Experiment < T > Class Template Reference

Contains classes for running the CS471 project experiment.

```
#include <experiment.h>
```

Public Member Functions

• Experiment ()

Construct a new Experiment object.

∼Experiment ()

Destroys the Experiment object.

• bool init (const char *paramFile)

Initializes the CS471 project 2 experiment. Opens the given parameter file and extracts test parameters. Allocates memory for function vectors and function bounds. Extracts all function bounds.

• int testAllFunc ()

Executes all functions as specified in the CS471 project 2 document, records results, and outputs the data as a *.csv file

int testFuncThreaded (mdata::TestParameters< T > tParams)

Executes a single iteration of a test with the given parameters.

6.4.1 Detailed Description

```
\label{template} \begin{array}{l} \text{template}{<}\text{class T}{>} \\ \text{class mfunc::Experiment}{<}\text{ T}{>} \end{array}
```

Contains classes for running the CS471 project experiment.

The Experiment class opens a given parameter .ini file and executes the CS471 project 2 experiment with the specified parameters. runAllFunc() runs all 18 functions defined in mfunctions.h a given number of times with vectors of random values that have a given number of dimensions and collects all results/data. This data is then entered into a DataTable and exported as a *.csv file.

Definition at line 52 of file experiment.h.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 Experiment()

```
template<class T >
Experiment::Experiment ( )
```

Construct a new Experiment object.

Definition at line 43 of file experiment.cpp.

```
00044 : vBounds(nullptr), tPool(nullptr), resultsFile(""), execTimesFile(""), iterations(0) 00045 { 00046 }
```

6.4.2.2 \sim Experiment()

```
template<class T > Experiment::~Experiment ( )
```

Destroys the Experiment object.

Definition at line 53 of file experiment.cpp.

6.4.3 Member Function Documentation

6.4.3.1 init()

Initializes the CS471 project 2 experiment. Opens the given parameter file and extracts test parameters. Allocates memory for function vectors and function bounds. Extracts all function bounds.

Parameters

paramFile	File path to the parameter ini file

Returns

Returns true if initialization was successful. Otherwise false.

Definition at line 69 of file experiment.cpp.

References enums::Count, enums::AlgorithmNames::get(), util::IniReader::getEntry(), util::IniReader::getEntryAs(), INI_TEST_ALGORITHM, INI_TEST_ALPHA, INI_TEST_DIMENSIONS, INI_TEST_EXECTIMESFILE, INI_TESC_TITERATIONS, INI_TEST_NUMTHREADS, INI_TEST_POPULATION, INI_TEST_RESULTSFILE, INI_TEST_SECTION, and util::IniReader::openFile().

Referenced by runExp().

```
08000
               // Extract test parameters from ini file
               long numberSol = iniParams.getEntryAs<long>(INI_TEST_SECTION,
00081
      INI_TEST_POPULATION);
00082
              long numberDim = iniParams.getEntryAs<long>(INI_TEST_SECTION,
      INI_TEST_DIMENSIONS);
00083
               long numberIter = iniParams.getEntryAs<long>(INI TEST SECTION.
      INI_TEST_ITERATIONS);
00084
               long numberThreads = iniParams.getEntryAs<long>(
      INI_TEST_SECTION, INI_TEST_NUMTHREADS);
00085
              alpha = iniParams.getEntryAs<T>(INI_TEST_SECTION,
      INI_TEST_ALPHA);
00086
              unsigned int selectedAlg = iniParams.getEntryAs<unsigned int>(
      INI_TEST_SECTION, INI_TEST_ALGORITHM);
    resultsFile = iniParams.getEntry(INI_TEST_SECTION,
00087
      INI_TEST_RESULTSFILE);
              execTimesFile = iniParams.getEntry(INI_TEST_SECTION,
00088
      INI_TEST_EXECTIMESFILE);
00089
00090
              // Verify test parameters
00091
              if (numberSol <= 0)</pre>
00092
              {
00093
                   cerr << "Experiment init failed: Param file [test]->"
<< INI_TEST_POPULATION << " entry missing or out of bounds: " <<</pre>
                  return false;
00096
00097
              else if (numberDim <= 0)</pre>
00098
                   cerr << "Experiment init failed: Param file [test]->"
00099
00100
                      << INI_TEST_DIMENSIONS << " entry missing or out of bounds: " <<
return false;
00102
00103
               else if (numberIter <= 0)</pre>
00104
                   cerr << "Experiment init failed: Param file [test]->"
00105
                      << INI_TEST_ITERATIONS << " entry missing or out of bounds: " <<
00106
return false;
00108
00109
              else if (numberThreads <= 0)</pre>
00110
              {
                   cerr << "Experiment init failed: Param file [test]->"
00111
                       << INI_TEST_NUMTHREADS << " entry missing or out of bounds: " <</pre>
00112
     paramFile << endl;</pre>
00113
                   return false;
00114
00115
              else if (alpha == 0)
00116
              {
00117
                   cerr << "Experiment init failed: Param file [test]->"
                       << INI_TEST_ALPHA << " is missing or is equal to zero: " << paramFile << endl
00118
00119
                   return false;
00120
              else if (selectedAlg >= static_cast<unsigned int>(
00121
      enums::Algorithm::Count))
00122
             {
00123
                   cerr << "Experiment init failed: Param file [test]->"
                       << INI_TEST_ALGORITHM << " entry missing or out of bounds: " << paramFile
00124
       << endl;
00125
                   return false:
00126
00127
00128
              // Cast iterations and test algorithm to correct types
00129
               iterations = (size_t)numberIter;
00130
              testAlg = static_cast<enums::Algorithm>(selectedAlg);
00131
              // Print test parameters to console
cout << "Population size: " << numberSol << endl;</pre>
00132
00133
              cout << "Dimensions: " << numberSor << endl;
cout << "Iterations: " << iterations << endl;
cout << "Alpha value: " << alpha << endl;
cout << "Algorithm: " << enums::AlgorithmNames::get(testAlg) << endl;</pre>
00134
00135
00136
00137
00138
00139
               // Allocate memory for all population objects. We need one for each thread to prevent conflicts.
               if (!allocatePopulationPool((size_t)numberThreads, (size_t)numberSol, (size_t)numberDim))
00140
00141
              {
00142
                   cerr << "Experiment init failed: Unable to allocate populations." << endl;</pre>
00143
                   return false;
00144
              }
00145
00146
               // Allocate memory for function vector bounds
00147
               if (!allocateVBounds())
00148
               {
00149
                   cerr << "Experiment init failed: Unable to allocate vector bounds array." << endl;
00150
                   return false;
00151
               }
```

```
// Fill function bounds array with data parsed from iniParams
00154
               if (!parseFuncBounds())
00155
              {
00156
                   cerr << "Experiment init failed: Unable to parse vector bounds array." << endl;</pre>
00157
                   return false:
00158
00159
00160
               // Allocate thread pool
00161
               if (!allocateThreadPool((size_t)numberThreads))
00162
              {
                   cerr << "Experiment init failed: Unable to allocate thread pool." << endl:
00163
00164
                   return false;
00165
00166
00167
              cout << "Started " << numberThreads << " worker threads ..." << endl;</pre>
00168
00169
              // Ready to run an experiment
00170
              return true;
00171
00172
          catch (const std::exception& ex)
00173
              cerr << "Exception occurred while initializing experiment: " << ex.what() << endl;</pre>
00174
00175
              return false;
00176
00177
          catch (...)
00178
00179
              cerr << "Unknown Exception occurred while initializing experiment." << endl;</pre>
00180
00181
          }
00182 }
```

6.4.3.2 testAllFunc()

```
template<class T >
int Experiment::testAllFunc ( )
```

Executes all functions as specified in the CS471 project 2 document, records results, and outputs the data as a *.csv file.

Returns

Returns 0 on success. Returns a non-zero error code on failure.

Definition at line 191 of file experiment.cpp.

References mdata::TestParameters< T >::alg, mdata::TestParameters< T >::alpha, ThreadPool::enqueue(), mdata::TestParameters< T >::execTimesCol, mdata::TestParameters< T >::execTimesRow, mdata::TestParameters< T >::execTimesRow, mdata::TestParameters< T >::execTimesTable, mdata::DataTable< T >::exportCSV(), mdata::TestParameters< T >::funcId, mfunc::FunctionDesc::get(), mfunc::NUM_FUNCTIONS, mdata::TestParameters< T >::resultsCol, mdata::TestParameters< T >::resultsTable, mdata::DataTable< T >::setCol Label(), and ThreadPool::stopAndJoinAll().

Referenced by runExp().

```
00192 {
00193
          if (populationsPool.size() == 0) return 1;
00194
          // Construct results and execution times tables
00195
00196
          mdata::DataTable<T> resultsTable(iterations, (size_t)
      NUM_FUNCTIONS);
00197
          mdata::DataTable<T> execTimesTable(iterations, (size_t)
     NUM_FUNCTIONS);
00198
00199
          // Prepare thread futures vector, used to ensure all async tasks complete
00200
          // succesfully.
00201
          std::vector<std::future<int>> testFutures;
```

```
00202
00203
           // Start recording total execution time
00204
           high_resolution_clock::time_point t_start = high_resolution_clock::now();
00205
           // For each of the NUM_FUNCTIONS functions, prepare a TestParameters // struct and queue an asynchronous test that will be picked up and
00206
00207
           // executed by one of the threads in the thread pool.
00209
           for (unsigned int i = 0; i < NUM_FUNCTIONS; i++)</pre>
00210
00211
               // Update column labels for results and exec times tables \,
               resultsTable.setColLabel((size_t)i, FunctionDesc::get(i + 1));
00212
00213
               execTimesTable.setColLabel((size_t)i, FunctionDesc::get(i + 1));
00214
00215
               // Queue up a new function test for each iteration
00216
               for (size_t iter = 0; iter < iterations; iter++)</pre>
00217
00218
                   mdata::TestParameters<T> curParam;
00219
                   curParam.funcId = i + 1;
                   curParam.alpha = alpha;
00220
00221
                   curParam.alg = testAlg;
00222
                   curParam.resultsTable = &resultsTable;
00223
                   curParam.execTimesTable = &execTimesTable;
00224
                   curParam.resultsCol = i;
00225
                   curParam.execTimesCol = i:
00226
                   curParam.resultsRow = iter;
                   curParam.execTimesRow = iter;
00228
00229
                   // Add function test to async queue
00230
                   testFutures.emplace_back(
00231
                        tPool->enqueue(&Experiment<T>::testFuncThreaded, this
      , curParam)
00232
                   );
00233
00234
00235
          // Get the total number of async tasks queued
00236
00237
          const double totalFutures = static_cast<double>(testFutures.size());
           int tensPercentile = -1;
00239
          std::chrono::microseconds waitTime(100);
00240
00241
           \ensuremath{//} Loop until all async tasks are completed and the thread futures
00242
           // array is empty
00243
          while (testFutures.size() > 0)
00244
00245
               // Sleep a little bit since the async thread tasks are higher priority
00246
               std::this_thread::sleep_for(waitTime);
00247
00248
               \ensuremath{//} Get iterator to first thread future
00249
               auto it = testFutures.begin();
00250
00251
               // Loop through all thread futures
00252
               while (it != testFutures.end())
00253
00254
                    if (!it->valid())
00255
                   {
00256
                        // An error occured with one of the threads
                        cerr << "Error: Thread future invalid.";
00258
                        tPool->stopAndJoinAll();
00259
                        return 1;
00260
                   }
00261
                   // Get the status of the current thread future (async task)
std::future_status status = it->wait_for(waitTime);
00262
00263
00264
                    if (status == std::future_status::ready)
00265
00266
                        // Task has completed, get return value
00267
                        int errCode = it->get();
00268
                        if (errCode)
00269
00270
                               An error occurred while running the task.
00271
                            // Bail out of function
00272
                            tPool->stopAndJoinAll();
00273
                            return errCode;
00274
00275
00276
                        // Remove processed task future from vector
00277
                        it = testFutures.erase(it);
00278
00279
                        // Calculate the percent completed of all tasks, rounded to the nearest 10\%
                        int curPercentile = static cast<int>(((totalFutures - testFutures.size()) / totalFutures) *
00280
       10);
00281
                        if (curPercentile > tensPercentile)
00282
00283
                            // Print latest percent value to the console
                            tensPercentile = curPercentile;
cout << "~" << (tensPercentile * 10) << "% " << flush;</pre>
00284
00285
00286
                        }
```

```
}
00288
00289
00290
                       // Async task has not yet completed, advance to the next one
00291
                       it++;
00292
                  }
00293
00294
00295
00296
          // Record total execution time and print it to the console
          high_resolution_clock::time_point t_end = high_resolution_clock::now();
00297
          long double totalExecTime = static_cast<long double>(duration_cast<nanoseconds>(t_end - t_start).count(
00298
     )) / 1000000000.0L;
00299
00300
          cout << endl << "Test finished. Total time: " << std::setprecision(7) << totalExecTime << " seconds." <</pre>
      < endl;
00301
00302
          if (!resultsFile.empty())
00303
00304
               // Export results table to a *.csv file
00305
              cout << "Exporting results to: " << resultsFile << endl;</pre>
00306
              resultsTable.exportCSV(resultsFile.c_str());
00307
          }
00308
00309
          if (!execTimesFile.empty())
00310
00311
              // Export exec times table to a \star.csv file
00312
              cout << "Exporting execution times to: " << execTimesFile << endl;</pre>
00313
              execTimesTable.exportCSV(execTimesFile.c_str());
00314
          }
00315
00316
          cout << flush;
00317
00318
          return 0;
00319 }
```

6.4.3.3 testFuncThreaded()

Executes a single iteration of a test with the given parameters.

Template Parameters

The data type used by the test

Parameters

tParams The parameters used to set up the test

Returns

int An error code if any

Definition at line 329 of file experiment.cpp.

References mdata::TestParameters < T >::alg, mdata::TestParameters < T >::alpha, enums::BlindSearch, mdata \leftarrow ::TestParameters < T >::execTimesRow, mdata::TestParameters < T >::execTimesRow, mdata::TestParameters < T >::execTimesRow, mdata::TestParameters < T >::execTimesRow, mdata::TestParameters < T >::funcId, util::IniReader::getEntry(), enums::LocalSearch, mfunc::RandomBounds < T >::min, mfunc::NUM_FUNCTIONS, mdata:: \leftarrow TestParameters < T >::resultsRow, mdata::TestParameters < T > \leftarrow ::resultsTable, and mdata::SearchAlgorithm < T >::run().

```
00330 {
00331
          mdata::SearchAlgorithm<T>* alg;
00332
00333
          \ensuremath{//} Construct a search algorithm object for the selected alg
00334
          switch (tParams.alg)
00335
00336
               case enums::Algorithm::BlindSearch:
00337
                  alg = new mdata::BlindSearch<T>();
00338
00339
               case enums::Algorithm::LocalSearch:
00340
                  alg = new mdata::LocalSearch<T>();
00341
                   break;
00342
              default:
00343
                  cerr << "Invalid algorithm selected." << endl;</pre>
00344
                   return 1;
00345
00346
00347
          // Retrieve the function bounds
00348
          const RandomBounds<T>& funcBounds = vBounds[tParams.funcId - 1];
00349
00350
           // Retrieve the next available population object from the population pool
00351
          mdata::Population<T>* pop = popPoolRemove();
00352
          // Run the search algorithm one and record the results
auto tResult = alg->run(Functions<T>::get(tParams.funcId), funcBounds.
00353
00354
     min, funcBounds.max, pop, tParams.alpha);
00355
00356
           // Place the population object back into the pool to be reused by anther thread
00357
          popPoolAdd(pop);
00358
00359
          if (tResult.err)
00360
          {
00361
               cerr << "Error while testing function " << tParams.funcId << endl;</pre>
00362
               return tResult.err;
00363
00364
00365
          // Update results table and execution times table with algorithm results
00366
          tParams.resultsTable->setEntry(tParams.resultsRow, tParams.
     resultsCol, tResult.fitness);
00367
          tParams.execTimesTable->setEntry(tParams.execTimesRow, tParams.
     execTimesCol, tResult.execTime);
00368
00369
          delete alg;
00370
          return 0;
00371 }
```

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

- · include/experiment.h
- src/experiment.cpp

6.5 mfunc::FunctionDesc Struct Reference

get() returns a function's description Returns a C-string description for the given function id if the id is valid. Otherwise returns null

```
#include <mfunctions.h>
```

Static Public Member Functions

• static const char * get (unsigned int f)

6.5.1 Detailed Description

get() returns a function's description Returns a C-string description for the given function id if the id is valid. Otherwise returns null

Parameters

f | Function id to retrieve the description for

Returns

A C-string containing the function description if id is valid, otherwise null.

Definition at line 56 of file mfunctions.h.

6.5.2 Member Function Documentation

```
6.5.2.1 get()
```

Definition at line 58 of file mfunctions.h.

References _ackleysOneDesc, _ackleysTwoDesc, _alpineDesc, _dejongDesc, _eggHolderDesc, _griewangk ← Desc, _levyDesc, _mastersCosineWaveDesc, _michalewiczDesc, _pathologicalDesc, _quarticDesc, _ranaDesc, _rastriginDesc, _rosenbrokDesc, _schwefelDesc, _sineEnvelopeSineWaveDesc, _stepDesc, and _stretchedV ← SineWaveDesc.

Referenced by mfunc::Experiment< T >::testAllFunc().

```
00059
00060
                  switch (f)
00061
                      case 1:
00062
00063
                          return _schwefelDesc;
00064
                      case 2:
00065
                          return _dejongDesc;
00066
                      case 3:
00067
                          return _rosenbrokDesc;
00068
                      case 4:
00069
                         return _rastriginDesc;
00070
                      case 5:
00071
                          return _griewangkDesc;
00072
                      case 6:
00073
                          return _sineEnvelopeSineWaveDesc;
00074
                      case 7:
00075
                          return _stretchedVSineWaveDesc;
00076
                      case 8:
00077
                          return _ackleysOneDesc;
00078
                      case 9:
00079
                          return _ackleysTwoDesc;
08000
                      case 10:
00081
                          return eggHolderDesc:
00082
                      case 11:
00083
00084
                      case 12:
00085
                          return _pathologicalDesc;
00086
                      case 13:
00087
                          return michalewiczDesc;
00088
                      case 14:
00089
                          return _mastersCosineWaveDesc;
00090
                      case 15:
00091
                          return _quarticDesc;
                      case 16:
00092
00093
                          return _levyDesc;
00094
                      case 17:
00095
                          return _stepDesc;
00096
                      case 18:
00097
                          return _alpineDesc;
00098
                      default:
00099
                          return NULL;
00100
                  }
00101
              }
```

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

include/mfunctions.h

6.6 mfunc::Functions < T > Struct Template Reference

Struct containing all static math functions. A function can be called directly by name, or indirectly using Functions.::get or Functions::exec.

```
#include <mfunctions.h>
```

Static Public Member Functions

```
• static T schwefel (T *v, size t n)
```

Function 1. Implementation of Schwefel's mathematical function.

static T dejong (T *v, size_t n)

Function 2. Implementation of 1st De Jong's mathematical function.

• static T rosenbrok (T *v, size_t n)

Function 3. Implementation of the Rosenbrock mathematical function.

• static T rastrigin (T *v, size t n)

Function 4. Implementation of the Rastrigin mathematical function.

static T griewangk (T *v, size_t n)

Function 5. Implementation of the Griewangk mathematical function.

static T sineEnvelopeSineWave (T *v, size_t n)

Function 6. Implementation of the Sine Envelope Sine Wave mathematical function.

• static T stretchedVSineWave (T *v, size_t n)

Function 7. Implementation of the Stretched V Sine Wave mathematical function.

static T ackleysOne (T *v, size_t n)

Function 8. Implementation of Ackley's One mathematical function.

• static T ackleysTwo (T *v, size_t n)

Function 9. Implementation of Ackley's Two mathematical function.

static T eggHolder (T *v, size_t n)

Function 10. Implementation of the Egg Holder mathematical function.

static T rana (T *v, size_t n)

Function 11. Implementation of the Rana mathematical function.

static T pathological (T *v, size_t n)

Function 12. Implementation of the Pathological mathematical function.

• static T mastersCosineWave (T *v, size_t n)

Function 14. Implementation of the Masters Cosine Wave mathematical function.

static T michalewicz (T *v, size_t n)

Function 13. Implementation of the Michalewicz mathematical function.

• static T quartic (T *v, size t n)

Function 15. Implementation of the Quartic mathematical function.

static T levy (T *v, size_t n)

Function 16. Implementation of the Levy mathematical function.

• static T step (T *v, size t n)

Function 17. Implementation of the Step mathematical function.

static T alpine (T *v, size_t n)

Function 18. Implementation of the Alpine mathematical function.

static mfuncPtr< T > get (unsigned int f)

Returns a function pointer to the math function with the given id.

static bool exec (unsigned int f, T *v, size_t n, T &outResult)

Executes a specific function Executes the function with the given id and returns true on success. Otherwise returns false if id is invalid.

- static T nthroot (T x, T n)
- static T w (T x)

6.6.1 Detailed Description

```
template < class T> struct mfunc::Functions < T>
```

Struct containing all static math functions. A function can be called directly by name, or indirectly using Functions::get or Functions::exec.

Template Parameters

```
T Data type for function calculations
```

Definition at line 112 of file mfunctions.h.

6.6.2 Member Function Documentation

6.6.2.1 ackleysOne()

Function 8. Implementation of Ackley's One mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 331 of file mfunctions.h.

```
00332 {
00333
           T f = 0.0;
00334
           for (size_t i = 0; i < n - 1; i++)</pre>
00335
00336
                T \ a = (static\_cast < T > (1.0) \ / \ pow(static\_cast < T > (M_E), \ static\_cast < T > (0.2))) \ * \ sqrt(v[i] * v[i] + v[i+1] 
00337
      1]*v[i+1]);
00338
                 \texttt{T b = static\_cast<T>(3.0) * (cos(static\_cast<T>(2.0) * v[i]) + sin(static\_cast<T>(2.0) * v[i+1])); } 
00339
00340
           }
00341
00342
          return f:
00343 }
```

6.6.2.2 ackleysTwo()

```
template<class T >
T mfunc::Functions< T >::ackleysTwo (
          T * v,
          size_t n ) [static]
```

Function 9. Implementation of Ackley's Two mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 355 of file mfunctions.h.

```
00356 {
                                                  T f = 0.0:
00357
00358
                                                   for (size_t i = 0; i < n - 1; i++)</pre>
00359
                                                                    T = static\_cast < T > (20.0) / pow(static\_cast < T > (M_E), static\_cast < T > (0.2) * sqrt((v[i] * v[i] + v[i+1]) + v[i+1]) + v[i+1] + v
                             ]*v[i+1]) / static_cast<T>(2.0)));
                                      T b = pow(static_cast<T>(M_E), static_cast<T>(0.5) *
00362
                           (cos(static_cast<T>(2.0) * static_cast<T>(M_PI) * v[i]) + cos(static_cast<T>(2.0) *
static_cast<T>(M_PI) * v[i+1])));
00363
 00364
                                                                       f += static_cast<T>(20.0) + static_cast<T>(M_E) - a - b;
 00365
00366
00367
                                                return f;
00368 }
```

6.6.2.3 alpine()

```
template<class T >
T mfunc::Functions< T >::alpine (
          T * v,
          size_t n ) [static]
```

Function 18. Implementation of the Alpine mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 585 of file mfunctions.h.

6.6.2.4 dejong()

```
template<class T >
T mfunc::Functions< T >::dejong (
          T * v,
          size_t n ) [static]
```

Function 2. Implementation of 1st De Jong's mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 183 of file mfunctions.h.

6.6.2.5 eggHolder()

```
template<class T >
T mfunc::Functions< T >::eggHolder (
          T * v,
          size_t n ) [static]
```

Function 10. Implementation of the Egg Holder mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 380 of file mfunctions.h.

```
00381 {
00382
      T f = 0.0;
00383
00384
      for (size_t i = 0; i < n - 1; i++)</pre>
00385
        00386
00387
   static_cast<T>(2.0))));
00388
        f += a - b;
00389
00390
      return f;
00391
00392 }
```

6.6.2.6 exec()

```
template<class T >
bool mfunc::Functions< T >::exec (
          unsigned int f,
          T * v,
          size_t n,
          T & outResult ) [static]
```

Executes a specific function Executes the function with the given id and returns true on success. Otherwise returns false if id is invalid.

Parameters

f	Function id to execute
V	Vector as a T value array
n	Size of the vector 'v'
outResult	Output reference variable for the result of the mathematical function

Returns

true if 'f' is a valid id and the function was ran. Otherwise false.

Definition at line 667 of file mfunctions.h.

6.6.2.7 get()

```
template<class T >
mfunc::mfuncPtr< T > mfunc::Functions< T >::get (
          unsigned int f) [static]
```

Returns a function pointer to the math function with the given id.

Template Parameters

T Data type to be used in the function's calculations

Parameters

```
f Id of the function (1-18)
```

Returns

mfunc::mfuncPtr<T> Function pointer to the associated function, or nullptr if the id is invalid.

Definition at line 609 of file mfunctions.h.

```
00610 {
00611
          switch (f)
00613
00614
                 return Functions<T>::schwefel;
00615
             case 2:
                 return Functions<T>::dejong;
00616
00617
             case 3:
00618
                return Functions<T>::rosenbrok;
00619
              case 4:
00620
                 return Functions<T>::rastrigin;
00621
              case 5:
00622
                 return Functions<T>::griewangk;
00623
              case 6:
00624
                 return Functions<T>::sineEnvelopeSineWave;
00625
              case 7:
00626
                 return Functions<T>::stretchedVSineWave;
00627
              case 8:
00628
                 return Functions<T>::ackleysOne;
00629
              case 9:
00630
                 return Functions<T>::ackleysTwo;
00631
              case 10:
```

```
00632
                 return Functions<T>::eggHolder;
00633
00634
                 return Functions<T>::rana;
             case 12:
00635
00636
                return Functions<T>::pathological;
00637
             case 13:
00638
               return Functions<T>::michalewicz;
00639
             case 14:
00640
                 return Functions<T>::mastersCosineWave;
00641
             case 15:
                return Functions<T>::quartic;
00642
             case 16:
00643
00644
                return Functions<T>::levy;
00645
             case 17:
00646
                 return Functions<T>::step;
00647
             case 18:
00648
                 return Functions<T>::alpine;
00649
             default:
00650
                return nullptr;
00651
         }
00652 }
```

6.6.2.8 griewangk()

Function 5. Implementation of the Griewangk mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 252 of file mfunctions.h.

```
00253 {
00254
          T sum = 0.0;
          T product = 0.0;
00255
00257
          for (size_t i = 0; i < n; i++)</pre>
00258
00259
              sum += (v[i] * v[i]) / static_cast<T>(4000.0);
00260
00261
00262
         for (size_t i = 0; i < n; i++)</pre>
00263
         {
00264
              product *= cos(v[i] / sqrt(static_cast<T>(i + 1.0)));
00265
00266
00267
          return static_cast<T>(1.0) + sum - product;
00268 }
```

6.6.2.9 levy()

Function 16. Implementation of the Levy mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 531 of file mfunctions.h.

```
00532 {
          T f = 0.0;
00534
          for (size_t i = 0; i < n - 1; i++)</pre>
00535
00536
              T = w(v[i]) - static_cast<T>(1.0);
00537
00538
              a *= a;
00539
              T b = sin(static\_cast<T>(M_PI) * w(v[i]) + static\_cast<T>(1.0));
00540
              b *= b;
00541
              T c = w(v[n - 1]) - static\_cast < T > (1.0);
00542
              c *= c;
T d = sin(static_cast<T>(2.0) * static_cast<T>(M_PI) * w(v[n - 1]));
00543
              d *= d;
00544
00545
              f += a * (static_cast<T>(1.0) + static_cast<T>(10.0) * b) + c * (static_cast<T>(1.0) + d);
00546
00547
00548
          T = sin(static\_cast<T>(M_PI) * w(v[0]));
00549
          return e*e + f;
00550 }
```

6.6.2.10 mastersCosineWave()

```
template<class T >
T mfunc::Functions< T >::mastersCosineWave (
          T * v,
          size_t n ) [static]
```

Function 14. Implementation of the Masters Cosine Wave mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 476 of file mfunctions.h.

```
00477 {
  00478
                                                                T f = 0.0;
  00479
   00480
                                                                  for (size_t i = 0; i < n - 1; i++)</pre>
   00481
                                                                                          \texttt{T a = pow(M\_E, static\_cast<T>(-1.0/8.0)*(v[i]*v[i] + v[i+1]*v[i+1] + static\_cast<T>(0.5)*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]
 00482
]));
00483
                                                                                          \texttt{T b = cos(static\_cast<T>(4) * sqrt(v[i]*v[i] + v[i+1]*v[i+1] + static\_cast<T>(0.5)*v[i]*v[i+1])); } 
  00484
                                                                                        f += a * b;
   00486
  00487
                                                                return static_cast<T>(-1.0) * f;
 00488 }
```

6.6.2.11 michalewicz()

```
template<class T >
T mfunc::Functions< T >::michalewicz (
          T * v,
          size_t n ) [static]
```

Function 13. Implementation of the Michalewicz mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 454 of file mfunctions.h.

6.6.2.12 nthroot()

Simple helper function that returns the nth-root

Parameters

Χ	Value to be taken to the nth power
n	root degree

Returns

The value of the nth-root of x

Definition at line 146 of file mfunctions.h.

```
00147 {
00148          return pow(x, static_cast<T>(1.0) / n);
00149 }
```

6.6.2.13 pathological()

Function 12. Implementation of the Pathological mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 428 of file mfunctions.h.

```
00429 {
00430
                T f = 0.0;
00431
                for (size_t i = 0; i < n - 1; i++)</pre>
00432
00433
                       T a = sin(sqrt(static\_cast<T>(100.0)*v[i]*v[i] + v[i+1]*v[i+1]));
00434
                      a = Sin(sqr((static_cast<1>(100.0)*v[i]*v[i] + v[i+i]*v[i+i]));
a = (a*a) - static_cast<T>(0.5);
T b = (v[i]*v[i] - static_cast<T>(2)*v[i]*v[i+1] + v[i+1]*v[i+1]);
b = static_cast<T>(1.0) + static_cast<T>(0.001) * b*b;
f += static_cast<T>(0.5) + (a/b);
00435
00436
00437
00438
00439
                }
00440
00441
                return f;
00442 }
```

6.6.2.14 quartic()

```
template<class T >
T mfunc::Functions< T >::quartic (
          T * v,
          size_t n ) [static]
```

Function 15. Implementation of the Quartic mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 500 of file mfunctions.h.

6.6.2.15 rana()

Function 11. Implementation of the Rana mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 404 of file mfunctions.h.

```
00405 {
  00406
                                                                         T f = 0.0;
 00407
                                                                          for (size_t i = 0; i < n - 1; i++)</pre>
 00408
 00409
                                                                                                     T \ a = v[i] \ * \ sin(sqrt(std::abs(v[i+1] - v[i] + static\_cast<T>(1.0)))) \ * \ cos(sqrt(std::abs(v[i+1] + v[i] + v[i] + v[i])))
00410
                                          v[i] + static_cast<T>(1.0)));
  00411
                                                                                                          \texttt{T} \ b = (\texttt{v[i+1]} \ + \ \texttt{static\_cast} < \texttt{T} > (1.0)) \ \star \ \cos(\texttt{sqrt}(\texttt{std::abs}(\texttt{v[i+1]} \ - \ \texttt{v[i]} \ + \ \texttt{static\_cast} < \texttt{T} > (1.0)))) \ \star \ \star \ (\texttt{v[i+1]} \ - \ \texttt{v[i]} \ + \ \texttt{static\_cast} < \texttt{v[i+1]} \ - \ \texttt{v[i]} \ + \ \texttt{v[i+1]} \ +
                                        sin(sqrt(std::abs(v[i+1] + v[i] + static_cast<T>(1.0))));
  00412
                                                                                                 f += a + b;
  00413
 00414
 00415
                                                                          return f;
 00416 }
```

6.6.2.16 rastrigin()

Function 4. Implementation of the Rastrigin mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 230 of file mfunctions.h.

6.6.2.17 rosenbrok()

Function 3. Implementation of the Rosenbrock mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 205 of file mfunctions.h.

```
00206 {
00207
           T f = 0.0;
00208
00209
           for (size_t i = 0; i < n - 1; i++)</pre>
00210
00211
                T a = ((v[i] * v[i]) - v[i+1]);
               T b = (static_cast<T>(1.0) - v[i]);
f += static_cast<T>(100.0) * a * a;
00212
00213
                f += b * b;
00214
00215
           }
00216
00217
           return f;
00218 }
```

6.6.2.18 schwefel()

Function 1. Implementation of Schwefel's mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 161 of file mfunctions.h.

6.6.2.19 sineEnvelopeSineWave()

```
template<class T >
T mfunc::Functions< T >::sineEnvelopeSineWave (
          T * v,
          size_t n ) [static]
```

Function 6. Implementation of the Sine Envelope Sine Wave mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 280 of file mfunctions.h.

```
00281 {
00282
          T f = 0.0;
00283
          for (size_t i = 0; i < n - 1; i++)</pre>
00284
00285
00286
              T = \sin(v[i]*v[i] + v[i+1]*v[i+1] - static_cast<T>(0.5));
00287
              T b = (static_cast<T>(1.0) + static_cast<T>(0.001)*(v[i]*v[i] + v[i+1]*v[i+1]));
00288
00289
00290
              f += static\_cast<T>(0.5) + (a / b);
00291
00292
00293
         return static_cast<T>(-1.0) * f;
00294 }
```

6.6.2.20 step()

Function 17. Implementation of the Step mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 562 of file mfunctions.h.

6.6.2.21 stretchedVSineWave()

Function 7. Implementation of the Stretched V Sine Wave mathematical function.

Parameters

V	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 306 of file mfunctions.h.

```
00307 {
          T f = 0.0;
00308
00309
          for (size_t i = 0; i < n - 1; i++)</pre>
00310
00311
00312
              T a = nthroot(v[i]*v[i] + v[i+1]*v[i+1], static_cast<T>(4.0));
00313
              T b = \sin(\text{static\_cast} < T > (50.0) * nthroot(v[i] * v[i] + v[i+1] * v[i+1], static\_cast < T > (10.0)));
00314
              b *= b;
              f += a * b + static_cast<T>(1.0);
00315
00316
          }
00317
00318
          return f;
00319 }
```

6.6.2.22 w()

Helper math function used in levy()

Definition at line 518 of file mfunctions.h.

```
00519 {
00520         return static_cast<T>(1.0) + (x - static_cast<T>(1.0)) / static_cast<T>(4.0);
00521 }
```

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

• include/mfunctions.h

6.7 util::IniReader Class Reference

The IniReader class is a simple *.ini file reader and parser.

```
#include <inireader.h>
```

Public Member Functions

• IniReader ()

Construct a new IniReader object.

∼IniReader ()

Destroys the IniReader object.

bool openFile (std::string filePath)

Opens the given ini file and parses all sections/entries. The all file data is stored in memory and the file is closed.

bool sectionExists (std::string section)

Returns true if the given section exists in the current ini file.

• bool entryExists (std::string section, std::string entry)

Returns true if the given section and entry key exists in the current ini file.

• std::string getEntry (std::string section, std::string entry)

Returns the value for the entry that has the given entry key within the given section.

template < class T >

T getEntryAs (std::string section, std::string entry)

6.7.1 Detailed Description

The IniReader class is a simple *.ini file reader and parser.

- Initialize an IniReader object:

IniReader ini;

Open and parse an *.ini file:

ini.openFile("my_ini_file.ini");

Note that the file is immediately closed after parsing, and the file data is retained in memory.

Retrieve an entry from the ini file:

std::string value = ini.getEntry("My Section", "entryKey");

Definition at line 46 of file inireader.h.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 IniReader()

```
IniReader::IniReader ( )
```

Construct a new IniReader object.

Definition at line 21 of file inireader.cpp.

```
00021
00022 {
00023 }
: file(""), iniMap()
```

6.7.2.2 ∼IniReader()

```
IniReader::~IniReader ( )
```

Destroys the IniReader object.

Definition at line 28 of file inireader.cpp.

6.7.3 Member Function Documentation

6.7.3.1 entryExists()

Returns true if the given section and entry key exists in the current ini file.

Parameters

section	std::string containing the section name
entry	std::string containing the entry key name

Returns

Returns true if the section and entry key exist in the ini file, otherwise false.

Definition at line 67 of file inireader.cpp.

Referenced by getEntry().

6.7.3.2 getEntry()

Returns the value for the entry that has the given entry key within the given section.

Parameters

section	std::string containing the section name
entry	std::string containing the entry key name

Returns

The value of the entry with the given entry key and section. Returns an empty string if the entry does not exist.

Definition at line 84 of file inireader.cpp.

References entryExists().

Referenced by getEntryAs(), mfunc::Experiment< T >::init(), and mfunc::Experiment< T >::testFuncThreaded().

6.7.3.3 getEntryAs()

Definition at line 57 of file inireader.h.

References getEntry().

Referenced by mfunc::Experiment< T >::init().

6.7.3.4 openFile()

Opens the given ini file and parses all sections/entries. The all file data is stored in memory and the file is closed.

Parameters

```
filePath Path to the ini file you wish to open
```

Returns

Returns true if the file was succesfully opened and parsed. Otherwise false.

Definition at line 40 of file inireader.cpp.

Referenced by mfunc::Experiment< T >::init().

6.7.3.5 sectionExists()

Returns true if the given section exists in the current ini file.

Parameters

section	std::string containing the section name
---------	---

Returns

Returns true if the section exists in the ini file, otherwise false.

Definition at line 55 of file inireader.cpp.

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

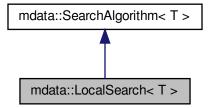
- · include/inireader.h
- src/inireader.cpp

6.8 mdata::LocalSearch< T > Class Template Reference

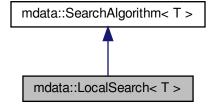
The LocalSearch class implements the Local Search algorithm, which is ran using the overridden Search ← Algorithm::run() function.

```
#include <localsearch.h>
```

Inheritance diagram for mdata::LocalSearch< T >:



Collaboration diagram for mdata::LocalSearch< T >:



Public Member Functions

virtual TestResult< T > run (mfunc::mfuncPtr< T > funcPtr, const T fMin, const T fMax, Population< T > *const pop, const T alpha)

Executes Local Search with the given population and parameters.

Additional Inherited Members

6.8.1 Detailed Description

```
template < class T> class mdata::LocalSearch < T>
```

The LocalSearch class implements the Local Search algorithm, which is ran using the overridden Search Algorithm::run() function.

Template Parameters

```
T Data type used
```

Definition at line 32 of file localsearch.h.

6.8.2 Member Function Documentation

```
6.8.2.1 run()
```

Executes Local Search with the given population and parameters.

Parameters

funcPtr	Function pointer to the math function being used to generate the population
fMin	Minimum bound for the population matrix vector components
fMax	Maximum bound for the population matrix vector components
pop Pointer to a population object that will be used in the local search	
alpha	Alpha value for local search neighbor generation

Returns

TestResult<T> Returns a TestResult struct containing the error code, fitness, and execution time

Implements mdata::SearchAlgorithm< T >.

Definition at line 49 of file localsearch.h.

References mdata::Population< T >:::calcFitness(), mdata::Population< T >::generate(), mdata::Population< T >::getBestFitnessIndex(), mdata::Population< T >::getDimensionsSize(), mdata::Population< T >::getFitness(), mdata::Population< T >::getPopulationPtr(), mdata::Population< T >::getPopulationSize(), mdata::Search Algorithm< T >::startTimer(), and mdata::SearchAlgorithm< T >::stopTimer().

```
00050
               {
                   \ensuremath{//} Get population size and dimensions
00051
                   const size_t popSize = pop->getPopulationSize();
const size_t dimSize = pop->getDimensionsSize();
00052
00054
00055
                   // Make sure funcPtr is valid;
                   if (funcPtr == nullptr) return TestResult<T>(1, 0, 0.0); // Invalid function id, return with
00056
       error code 1
00057
00058
                   // Algorithm related variables
00059
                   bool stop = false;
00060
                   size_t pIndex = 0;
00061
00062
                   startTimer():
00063
00064
                   // Start recording execution time
00065
                   pop->generate(fMin, fMax);
00066
00067
                   for (size_t sol = 0; sol < popSize; sol++)</pre>
00068
00069
                       // Populate fitness values using given math function pointer
00070
                       if (!pop->calcFitness(sol, funcPtr))
00071
                           return TestResult<T>(2, 0, 0.0); // Invalid fitness index, return with error code 2
00072
00073
                   \ensuremath{//} Get the index for the best fitness in the population
00074
00075
                   pIndex = pop->getBestFitnessIndex();
00076
                   // Get population vector and fitness of best solution
00078
                   T* x = pop->getPopulationPtr(pIndex);
00079
                   T xFit = pop->getFitness(pIndex);
08000
00081
                   // Create empty Y vector
00082
                   T* y = util::allocArray<T>(dimSize);
00083
                   T yFit = 0;
00084
00085
                   // Create empty Z vector
00086
                   T* z = util::allocArray<T>(dimSize);
00087
                   T zFit = 0;
00088
00089
                   if (x == nullptr || y == nullptr || z == nullptr)
00090
00091
                       std::cerr << "Error in Local Search: Memory allocation failed" << std::endl;</pre>
00092
                       return TestResult<T>(3, 0, 0.0);
00093
00094
00095
                   // Keep looping until search fails to improve
00096
                   while (!stop)
00097
00098
                       stop = true;
00099
                       // Copy values from X vector into Y vector
00100
00101
                       util::copyArray<T>(x, y, dimSize);
00102
00103
                        // Loop through each dimension in vector
00104
                       for (size_t a = 0; a < dimSize; a++)</pre>
00105
00106
                           // Add alpha value to y[a]
00107
                           y[a] = x[a] + alpha;
00108
                            // Make sure y[a] is within the function bounds
00109
00110
                           lockBounds(y[a], fMin, fMax);
00111
                            // Calculate fitness for v vector
00112
00113
                           yFit = funcPtr(y, dimSize);
00114
00115
                            // Update Z[a] vector value based on the difference between Y and X
```

```
z[a] = x[a] - (alpha * (yFit - xFit));
00118
                           // Make sure z[a] is within the function bounds
00119
                           lockBounds(z[a], fMin, fMax);
00120
                           y[a] = x[a]; // Reset y[a] to prepare for next loop
00121
00122
00123
00124
                       zFit = funcPtr(z, dimSize);
00125
                       // The following 'if' statement may cause extreme execution
00126
                       // times for some functions due to floating point precision:
00127
00128
                       // if (zFit < xFit)</pre>
00129
00130
                       // The replacement 'if' statement below places a limit
00131
                       \ensuremath{//} on the minimum acknowledged improvement fitness,
00132
                       // hopefully preventing extreme run-times:
                       if (xFit - zFit > MIN_IMPROVEMENT)
00133
00134
00135
                           // Z is an improvement on X, so keep searching
00136
                           stop = false;
00137
                           // Swap {\tt Z} and {\tt X} for next loop
00138
                           T* tmp = x;
00139
00140
                           x = z;
                           xFit = zFit;
00141
00142
                           z = tmp;
00143
00144
                  }
00145
00146
                  // Return best result
00147
                  return TestResult<T>(0, xFit, stopTimer());
00148
```

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

· include/localsearch.h

6.9 mdata::Population < T > Class Template Reference

Data class for storing a multi-dimensional population of data with the associated fitness.

```
#include <population.h>
```

Public Member Functions

• Population (size_t popSize, size_t dimensions)

Construct a new Population object.

∼Population ()

Destroy Population object.

• bool isReady ()

Returns true if the population instance is allocated and ready to be used.

• size_t getPopulationSize ()

Returns the size of the population.

• size t getDimensionsSize ()

Returns the dimensions of the population.

T * getPopulationPtr (size_t popIndex)

Returns an array for the population with the given index.

bool generate (T minBound, T maxBound)

Generates new random values for this population that are within the given bounds. Resets all fitness values to zero.

• bool setFitness (size_t popIndex, T value)

Sets the fitness value for a specific population vector index.

bool calcFitness (size_t popIndex, mfunc::mfuncPtr< T > funcPtr)

Uses the given function pointer to update the fitness value for the population vector at the given index.

• T getFitness (size_t popIndex)

Returns the fitness value for a specific population vector index.

• T * getFitnessPtr (size_t popIndex)

Returns the fitness value for a specific population vector index.

• std::vector< T > getAllFitness ()

Returns a std::vector of all current fitness values.

T * getBestFitnessPtr ()

Returns a pointer to the current best fitness value.

size_t getBestFitnessIndex ()

Returns the index of the current best fitness value.

void outputPopulation (std::ostream &outStream, const char *delim, const char *lineBreak)

Outputs all population data to the given output stream.

void outputFitness (std::ostream &outStream, const char *delim, const char *lineBreak)

Outputs all fitness data to the given output stream.

6.9.1 Detailed Description

```
template < class T> class mdata::Population < T>
```

Data class for storing a multi-dimensional population of data with the associated fitness.

Template Parameters

```
T Data type of the population.
```

Definition at line 30 of file population.h.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Population()

Construct a new Population object.

Template Parameters

```
T Data type of the population.
```

Parameters

pSize	Size of the population.
dimensions	Dimensions of the population.

Definition at line 27 of file population.cpp.

6.9.2.2 \sim Population()

```
template<class T >
Population::~Population ( )
```

Destroy Population object.

Template Parameters

```
T \mid Data type of the population.
```

Definition at line 39 of file population.cpp.

6.9.3 Member Function Documentation

6.9.3.1 calcFitness()

Uses the given function pointer to update the fitness value for the population vector at the given index.

Template Parameters

 $T \mid$ Data type of the population.

Parameters

popIndex	Index of the population vector you wish to set the fitness for.
funcPtr	Function pointer to the math function that will be used to calculate the fitness value.

Returns

Returns true on success, otherwise false.

Definition at line 163 of file population.cpp.

Referenced by mdata::BlindSearch< T >::run(), and mdata::LocalSearch< T >::run().

6.9.3.2 generate()

Generates new random values for this population that are within the given bounds. Resets all fitness values to zero.

Template Parameters

T Data	type of the population.
--------	-------------------------

Parameters

minBound	The minimum bound for a population value.
maxBound	The maximum bound for a population value.

Returns

Returns true of the population was succesfully generated, otherwise false.

Definition at line 108 of file population.cpp.

Referenced by mdata::BlindSearch< T >::run(), and mdata::LocalSearch< T >::run().

```
00112
          \ensuremath{//} Generate a new seed for the mersenne twister engine
00113
          rgen = std::mt19937(rdev());
00114
00115
          ^{\prime\prime} Set up a normal (bell-shaped) distribution for the random number generator with the correct function
       bounds
00116
          std::uniform_real_distribution<double> dist((double)minBound, (double)maxBound);
00117
00118
          // Generate values for all vectors in popMatrix
00119
          for (size_t s = 0; s < popSize; s++)</pre>
00120
00121
               for (size_t d = 0; d < popDim; d++)</pre>
00122
              {
00123
                   T rand = (T)dist(rgen);
00124
                   popMatrix[s][d] = rand;
00125
00126
          }
00127
00128
          // Reset popFitness values to 0
00129
          initArray<T>(popFitness, popSize, (T) 0.0);
00130
00131
          return true;
00132 }
```

6.9.3.3 getAllFitness()

```
template<class T >
std::vector< T > Population::getAllFitness ( )
```

Returns a std::vector of all current fitness values.

Template Parameters

 $T \mid$ Data type of the population.

Returns

std::vector<T> std::vector of fitness values

Definition at line 209 of file population.cpp.

```
00210 {
00211         return std::vector<T>(popFitness[0], popFitness[popSize]);
00212 }
```

6.9.3.4 getBestFitnessIndex()

```
template<class T >
size_t Population::getBestFitnessIndex ( )
```

Returns the index of the current best fitness value.

Template Parameters

```
T \mid Data type of the population.
```

Returns

size_t Index of the best fitness value

Definition at line 233 of file population.cpp.

Referenced by mdata::Population < T >::getBestFitnessPtr(), and mdata::LocalSearch < T >::run().

6.9.3.5 getBestFitnessPtr()

```
template<class T >
T * Population::getBestFitnessPtr ( )
```

Returns a pointer to the current best fitness value.

Template Parameters

```
T Data type of the population.
```

Returns

T* Pointer to the best fitness value

Definition at line 221 of file population.cpp.

References mdata::Population< T >::getBestFitnessIndex().

Referenced by mdata::BlindSearch< T >::run().

```
00222 {
00223          return &popFitness[getBestFitnessIndex()];
00224 }
```

6.9.3.6 getDimensionsSize()

```
template<class T >
size_t Population::getDimensionsSize ( )
```

Returns the dimensions of the population.

Template Parameters

```
T \mid Data type of the population.
```

Returns

The number of dimensions in the population.

Definition at line 77 of file population.cpp.

Referenced by mdata::BlindSearch< T >::run(), and mdata::LocalSearch< T >::run().

```
00078 {
00079 return popDim;
00080 }
```

6.9.3.7 getFitness()

Returns the fitness value for a specific population vector index.

Template Parameters

```
T Data type of the population.
```

Parameters

popIndex Index of the population vector you wish to retrieve the fitness from.

Returns

Returns the fitness value if popIndex is valid. Otherwise zero.

Definition at line 180 of file population.cpp.

Referenced by mdata::LocalSearch< T >::run().

```
00181 {
00182          if (popFitness == nullptr || popIndex >= popSize) return 0;
00183
00184          return popFitness[popIndex];
00185 }
```

6.9.3.8 getFitnessPtr()

Returns the fitness value for a specific population vector index.

Template Parameters

```
T Data type of the population.
```

Parameters

popIndex	Index of the population vector you wish to retrieve the fitness from.
----------	---

Returns

Returns the fitness value if popIndex is valid. Otherwise zero.

Definition at line 195 of file population.cpp.

6.9.3.9 getPopulationPtr()

Returns an array for the population with the given index.

Template Parameters

```
T Data type of the population.
```

Parameters

popIndex	Index of the population vector you wish to retrieve.
popiliaon	mask of the population rooter you man to rother or

Returns

Pointer to population vector array at the given index.

Definition at line 90 of file population.cpp.

Referenced by mdata::LocalSearch< T >::run().

```
00091 {
00092          if (popFitness == nullptr || popIndex >= popSize) return nullptr;
00093
00094          return popMatrix[popIndex];
00095 }
```

6.9.3.10 getPopulationSize()

```
template<class T >
size_t Population::getPopulationSize ( )
```

Returns the size of the population.

Template Parameters

```
T \mid Data type of the population.
```

Returns

The size of the population.

Definition at line 65 of file population.cpp.

6.9.3.11 isReady()

```
template<class T >
bool Population::isReady ( )
```

Returns true if the population instance is allocated and ready to be used.

Template Parameters

 $T \mid$ Data type of the population.

Returns

Returns true if the population instance is in a valid state.

Definition at line 53 of file population.cpp.

```
00054 {
00055          return popMatrix != nullptr && popFitness != nullptr;
00056 }
```

6.9.3.12 outputFitness()

Outputs all fitness data to the given output stream.

Template Parameters

```
T Data type of the population.
```

Parameters

outStream	Output stream to write the data to.
delim	Delimiter characters to separate columns.
lineBreak	Delimiter characters to separate rows.

Definition at line 281 of file population.cpp.

```
00282 {
00283
00284
           if (popFitness == nullptr) return;
00285
           for (size_t j = 0; j < popSize; j++)</pre>
00286
               outStream << popFitness[j];
                if (j < popSize - 1)
00288
00289
                        outStream << delim;
00290
           }
00291
           if (lineBreak != nullptr)
   outStream << lineBreak;</pre>
00292
00293
00294 }
```

6.9.3.13 outputPopulation()

```
template<class T >
void Population::outputPopulation (
```

```
std::ostream & outStream,
const char * delim,
const char * lineBreak )
```

Outputs all population data to the given output stream.

Template Parameters

T	Data type of the population.
---	------------------------------

Parameters

outStream	Output stream to write the data to.
delim	Delimiter characters to separate columns.
lineBreak	Delimiter characters to separate rows.

Definition at line 255 of file population.cpp.

```
00256 {
           if (popMatrix == nullptr) return;
00257
00258
           for (size_t j = 0; j < popSize; j++)</pre>
00259
00260
00261
                for (size_t k = 0; k < popDim; k++)</pre>
00262
               {
00263
                    outStream << popMatrix[j][k];</pre>
                    if (k < popDim - 1)
   outStream << delim;</pre>
00264
00265
00266
00267
00268
               outStream << lineBreak;
00269
00270 }
```

6.9.3.14 setFitness()

Sets the fitness value for a specific population vector index.

Template Parameters

```
T Data type of the population.
```

Parameters

popIndex	Index of the population vector you wish to set the fitness for.
value	The value of the fitness.

Returns

Returns true if the fitness was succesfully set, otherwise false.

Definition at line 143 of file population.cpp.

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

- · include/population.h
- · src/population.cpp

6.10 mfunc::RandomBounds< T > Struct Template Reference

Simple struct for storing the minimum and maximum input vector bounds for a function.

```
#include <experiment.h>
```

Public Attributes

- $T \min = 0.0$
- $T \max = 0.0$

6.10.1 Detailed Description

```
template < class T > struct mfunc::RandomBounds < T >
```

Simple struct for storing the minimum and maximum input vector bounds for a function.

Definition at line 35 of file experiment.h.

6.10.2 Member Data Documentation

```
6.10.2.1 max
```

```
template<class T>
T mfunc::RandomBounds< T >::max = 0.0
```

Definition at line 38 of file experiment.h.

Referenced by mfunc::Experiment< T >::testFuncThreaded().

6.10.2.2 min

```
template<class T>
T mfunc::RandomBounds< T >::min = 0.0
```

Definition at line 37 of file experiment.h.

Referenced by mfunc::Experiment< T >::testFuncThreaded().

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

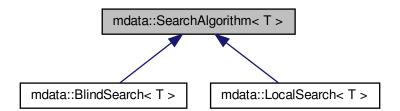
· include/experiment.h

6.11 mdata::SearchAlgorithm < T > Class Template Reference

The SearchAlgorithm class is used as a base class for other implemented search algorithms. Provides a common interface to run each algorithm.

```
#include <searchalg.h>
```

Inheritance diagram for mdata::SearchAlgorithm< T >:



Public Member Functions

- SearchAlgorithm ()
- virtual \sim SearchAlgorithm ()=0
- virtual TestResult< T > run (mfunc::mfuncPtr< T > funcPtr, const T fMin, const T fMax, Population< T > *const pop, const T alpha)=0

Protected Member Functions

• void startTimer ()

Starts the execution time timer.

• double stopTimer ()

Returns the amount of time that has passed since startTimer() was called in miliseconds.

Protected Attributes

- double timeDiff
- high_resolution_clock::time_point timer

6.11.1 Detailed Description

```
\label{template} \begin{split} \text{template} &< \text{class T}> \\ \text{class mdata::SearchAlgorithm} &< \text{T}> \end{split}
```

The SearchAlgorithm class is used as a base class for other implemented search algorithms. Provides a common interface to run each algorithm.

Template Parameters

```
The data type used by the algorithm
```

Definition at line 70 of file searchalg.h.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 SearchAlgorithm()

```
template<class T>
mdata::SearchAlgorithm< T >::SearchAlgorithm ( ) [inline]
```

Definition at line 73 of file searchalg.h.

```
00073 : timeDiff(0.0) {}
```

6.11.2.2 ~SearchAlgorithm()

```
template<class T >
mdata::SearchAlgorithm< T >::~SearchAlgorithm ( ) [pure virtual]
```

Definition at line 102 of file searchalg.h.

```
00102 { }
```

6.11.3 Member Function Documentation

6.11.3.1 run()

Implemented in mdata::LocalSearch< T >, and mdata::BlindSearch< T >.

Referenced by mfunc::Experiment< T >::testFuncThreaded().

6.11.3.2 startTimer()

```
template<class T>
void mdata::SearchAlgorithm< T >::startTimer ( ) [inline], [protected]
```

Starts the execution time timer.

Definition at line 83 of file searchalg.h.

Referenced by mdata::BlindSearch< T >::run(), and mdata::LocalSearch< T >::run().

6.11.3.3 stopTimer()

```
template<class T>
double mdata::SearchAlgorithm< T >::stopTimer ( ) [inline], [protected]
```

Returns the amount of time that has passed since startTimer() was called in miliseconds.

Definition at line 91 of file searchalg.h.

Referenced by mdata::BlindSearch< T >::run(), and mdata::LocalSearch< T >::run().

6.11.4 Member Data Documentation

6.11.4.1 timeDiff

```
template<class T>
double mdata::SearchAlgorithm< T >::timeDiff [protected]
```

Definition at line 77 of file searchalg.h.

6.11.4.2 timer

```
template<class T>
high_resolution_clock::time_point mdata::SearchAlgorithm< T >::timer [protected]
```

Definition at line 78 of file searchalg.h.

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

· include/searchalg.h

6.12 mdata::TestParameters < T > Struct Template Reference

Packs together various test experiment parameters.

```
#include <testparam.h>
```

Public Member Functions

• TestParameters ()

Public Attributes

- · unsigned int funcld
- Talpha
- unsigned int resultsCol
- unsigned int execTimesCol
- size_t resultsRow
- size_t execTimesRow
- DataTable < T > * resultsTable
- DataTable < T > * execTimesTable
- enums::Algorithm alg

6.12.1 Detailed Description

```
\label{template} \begin{split} \text{template} &< \text{class T}> \\ \text{struct mdata::TestParameters} &< \text{T}> \end{split}
```

Packs together various test experiment parameters.

Template Parameters

T | The data type that the test algorithm and functions are using

Definition at line 29 of file testparam.h.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 TestParameters()

```
template<class T>
mdata::TestParameters< T >::TestParameters ( ) [inline]
```

Selected search algorithm for this test

Definition at line 41 of file testparam.h.

References enums::BlindSearch.

```
00042
00043
funcId = 1;
00044
alpha = 0;
00045
alg = enums::Algorithm::BlindSearch;
00046
resultsTable = nullptr;
00047
execTimesTable = nullptr;
00049
execTimesCol = 0;
00050
resultsRow = 0;
00051
execTimesRow = 0;
00052
}
```

6.12.3 Member Data Documentation

```
6.12.3.1 alg
```

```
template<class T>
enums::Algorithm mdata::TestParameters< T >::alg
```

Pointer to the DataTable used to store the execution times

Definition at line 39 of file testparam.h.

Referenced by mfunc::Experiment< T >::testAllFunc(), and mfunc::Experiment< T >::testFuncThreaded().

6.12.3.2 alpha

```
template<class T>
T mdata::TestParameters< T >::alpha
```

Id for the tested math function

Definition at line 32 of file testparam.h.

Referenced by mfunc::Experiment < T >::testAllFunc(), and mfunc::Experiment < T >::testFuncThreaded().

6.12.3.3 execTimesCol

```
template<class T>
unsigned int mdata::TestParameters< T >::execTimesCol
```

DataTable column index to store the fitness result

Definition at line 34 of file testparam.h.

Referenced by mfunc::Experiment < T >::testAllFunc(), and mfunc::Experiment < T >::testFuncThreaded().

6.12.3.4 execTimesRow

```
template<class T>
size_t mdata::TestParameters< T >::execTimesRow
```

DataTable row index to store the fitness result

Definition at line 36 of file testparam.h.

Referenced by mfunc::Experiment< T >::testAllFunc(), and mfunc::Experiment< T >::testFuncThreaded().

6.12.3.5 execTimesTable

```
template<class T>
DataTable<T>* mdata::TestParameters< T >::execTimesTable
```

Pointer to the DataTable used to store the fitness results

Definition at line 38 of file testparam.h.

Referenced by mfunc::Experiment < T >::testAllFunc(), and mfunc::Experiment < T >::testFuncThreaded().

6.12.3.6 funcld

```
template<class T>
unsigned int mdata::TestParameters< T >::funcId
```

Definition at line 31 of file testparam.h.

Referenced by mfunc::Experiment < T >::testAllFunc(), and mfunc::Experiment < T >::testFuncThreaded().

6.12.3.7 resultsCol

```
template<class T>
unsigned int mdata::TestParameters< T >::resultsCol
```

Alpha value if needed by the search alg

Definition at line 33 of file testparam.h.

Referenced by mfunc::Experiment < T >::testAllFunc(), and mfunc::Experiment < T >::testFuncThreaded().

6.12.3.8 resultsRow

```
template<class T>
size_t mdata::TestParameters< T >::resultsRow
```

DataTable column index to store the execution time

Definition at line 35 of file testparam.h.

Referenced by mfunc::Experiment < T > ::testAllFunc(), and mfunc::Experiment < T > ::testFuncThreaded().

6.12.3.9 resultsTable

```
template<class T>
DataTable<T>* mdata::TestParameters< T >::resultsTable
```

DataTable row index to store the execution time

Definition at line 37 of file testparam.h.

 $Referenced \ by \ mfunc:: Experiment < T > :: testAllFunc(), \ and \ mfunc:: Experiment < T > :: testFuncThreaded().$

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

include/testparam.h

6.13 mdata::TestResult < T > Struct Template Reference

```
#include <testresult.h>
```

Public Member Functions

• TestResult (int _err, T _fitness, double _execTime)

Public Attributes

- · const int err
- const T fitness
- const double execTime

6.13.1 Detailed Description

```
\label{template} \begin{split} \text{template} &< \text{class T}> \\ \text{struct mdata::} &\text{TestResult} < \text{T}> \end{split}
```

Definition at line 20 of file testresult.h.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 TestResult()

Definition at line 26 of file testresult.h.

6.13.3 Member Data Documentation

6.13.3.1 err

```
template<class T>
const int mdata::TestResult< T >::err
```

Definition at line 22 of file testresult.h.

6.13.3.2 execTime

```
template<class T>
const double mdata::TestResult< T >::execTime
```

Definition at line 24 of file testresult.h.

6.13.3.3 fitness

```
template<class T>
const T mdata::TestResult< T >::fitness
```

Definition at line 23 of file testresult.h.

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

• include/testresult.h

6.14 ThreadPool Class Reference

```
#include <threadpool.h>
```

Public Member Functions

- ThreadPool (size_t)
- template < class F , class... Args >
 auto enqueue (F &&f, Args &&... args) -> std::future < typename std::result_of < F(Args...) > ::type >
- ∼ThreadPool ()
- void stopAndJoinAll ()

6.14.1 Detailed Description

Copyright (c) 2012 Jakob Progsch, Václav Zeman https://github.com/progschj/ThreadPool

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

- 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
- 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
- 3. This notice may not be removed or altered from any source distribution.

This source file has been modified slightly by Andrew Dunn

Definition at line 42 of file threadpool.h.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 ThreadPool()

Definition at line 64 of file threadpool.h.

```
00065
             stop(false)
00066 {
          for(size_t i = 0;i<threads;++i)</pre>
00068
              workers.emplace_back(
00069
                   [this]
00070
00071
                       for(::)
00072
00073
                           std::function<void()> task;
00074
00075
00076
                                std::unique_lock<std::mutex> lock(this->queue_mutex);
00077
                                this->condition.wait(lock,
00078
                                    [this]{ return this->stop || !this->tasks.empty(); });
00079
                                if(this->stop && this->tasks.empty())
00080
00081
                                task = std::move(this->tasks.front());
00082
                                this->tasks.pop();
00083
00084
00085
                           task();
00086
00087
                  }
00088
              );
00089 }
```

6.14.2.2 \sim ThreadPool()

```
ThreadPool::~ThreadPool () [inline]
```

Definition at line 117 of file threadpool.h.

References stopAndJoinAll().

6.14.3 Member Function Documentation

6.14.3.1 enqueue()

```
template<class F , class... Args> auto ThreadPool::enqueue (  F \&\& f, \\  Args \&\&... \ args ) \rightarrow std::future<typename std::result_of<F(Args...)>::type>
```

Definition at line 93 of file threadpool.h.

Referenced by mfunc::Experiment< T >::testAllFunc().

```
00095 {
00096
          using return_type = typename std::result_of<F(Args...)>::type;
00097
00098
          auto task = std::make_shared< std::packaged_task<return_type()> >(
00099
                  std::bind(std::forward<F>(f), std::forward<Args>(args)...)
00100
00101
00102
          std::future<return_type> res = task->get_future();
00103
00104
              std::unique_lock<std::mutex> lock(queue_mutex);
00105
00106
              // don't allow enqueueing after stopping the pool
00107
00108
                  throw std::runtime_error("enqueue on stopped ThreadPool");
00109
00110
              tasks.emplace([task](){ (*task)(); });
00111
00112
          condition.notify_one();
00113
          return res;
00114 }
```

6.14.3.2 stopAndJoinAll()

```
void ThreadPool::stopAndJoinAll ( ) [inline]
```

Definition at line 122 of file threadpool.h.

Referenced by mfunc::Experiment< T >::testAllFunc(), and ~ThreadPool().

```
00123 {
00124
          {
00125
              std::unique lock<std::mutex> lock(queue mutex);
00126
              stop = true;
00127
00128
00129
          condition.notify_all();
00130
          for(std::thread &worker: workers)
00131
              worker.join();
00132 }
```

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

· include/threadpool.h

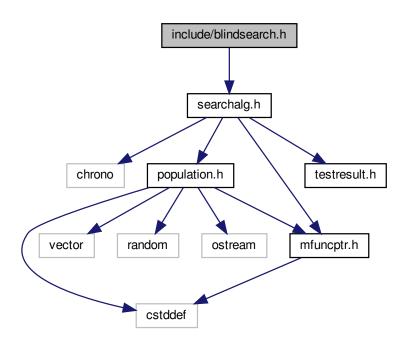
Chapter 7

File Documentation

7.1 include/blindsearch.h File Reference

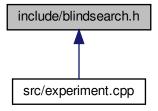
Implements the BlindSearch class, which inherits SearchAlgorithm. BlindSearch::run executes the blind search algorithm on a given population.

#include "searchalg.h"
Include dependency graph for blindsearch.h:



84 File Documentation

This graph shows which files directly or indirectly include this file:



Classes

class mdata::BlindSearch< T >

The BlindSearch class implements the Blind Search algorithm, which is ran using the overridden SearchAlgorithm ::run() function.

Namespaces

• mdata

7.1.1 Detailed Description

Implements the BlindSearch class, which inherits SearchAlgorithm. BlindSearch::run executes the blind search algorithm on a given population.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-19

Copyright

Copyright (c) 2019

Definition in file blindsearch.h.

7.2 blindsearch.h 85

7.2 blindsearch.h

```
00013 #ifndef __BLINDSEARCH_H
00014 #define ___BLINDSEARCH_H
00015
00016 #include "searchalg.h"
00017
00018 namespace mdata
00019 {
00026
          template<class T>
00027
          class BlindSearch : public SearchAlgorithm<T>
00028
00029
              // Declaration needed due to template base class
              using SearchAlgorithm<T>::startTimer;
00030
00031
              using SearchAlgorithm<T>::stopTimer;
00032
        public:
00033
     virtual TestResult<T> run(mfunc::mfuncPtr<T> funcPtr, const T
fMin, const T fMax, Population<T>* const pop, const T alpha)
00044
00045
              {
00046
                   // Get population size and dimensions
                  size_t popSize = pop->getPopulationSize();
size_t dimSize = pop->getDimensionsSize();
00047
00048
00049
00050
                  // Make sure funcPtr is valid;
return with error code 1
00052
                  if (funcPtr == nullptr) return TestResult<T>(1, 0, 0.0); // Invalid function id,
00053
                  // Start recording execution time
00054
                  startTimer();
00055
00056
                  // Generate values for population vector matrix
                  pop->generate(fMin, fMax);
00058
00059
                  // For each population vector, calculate the fitness using the funcPtr
00060
                  for (size_t sol = 0; sol < popSize; sol++)</pre>
00061
00062
                       // Populate fitness values using given math function pointer
00063
                       if (!pop->calcFitness(sol, funcPtr))
                           return TestResult<T>(2, 0, 0.0); // Invalid fitness index, return with
00064
      error code 2
00065
00066
                  // Return best fitness value in population
00067
                   return TestResult<T>(0, *pop->getBestFitnessPtr(),
00068
     stopTimer());
00069
00070
00071 }
00072
00073 #endif
00075 // ==========
00076 // End of blindsearch.h
00077 // ===========
```

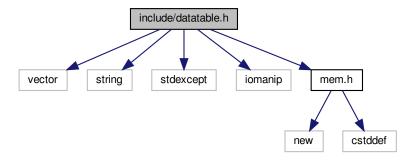
7.3 include/datatable.h File Reference

Header file for the DataTable class, which represents a spreadsheet/table of values that can easily be exported to a *.csv file.

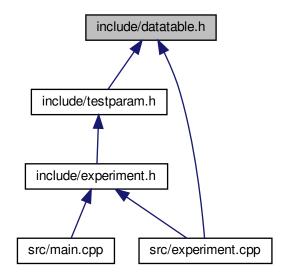
```
#include <vector>
#include <string>
#include <stdexcept>
#include <iomanip>
#include "mem.h"
```

86 File Documentation

Include dependency graph for datatable.h:



This graph shows which files directly or indirectly include this file:



Classes

• class mdata::DataTable < T >

The DataTable class is a simple table of values with labeled columns.

Namespaces

mdata

7.4 datatable.h

7.3.1 Detailed Description

Header file for the DataTable class, which represents a spreadsheet/table of values that can easily be exported to a * csv file

Author

```
Andrew Dunn (Andrew.Dunn@cwu.edu)
```

Version

0.2

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file datatable.h.

7.4 datatable.h

```
00001
00013 #ifndef __DATATABLE_H
00014 #define __DATATABLE_H
00015
00016 #include <vector>
00017 #include <string>
00018 #include <stdexcept>
00019 #include <iomanip>
00020 #include "mem.h"
00021
00022 namespace mdata
00023 {
00049
          template <class T>
00050
          class DataTable
00051
00052
          public:
00060
              DataTable(size_t _rows, size_t _cols) : rows(_rows), cols(_cols), dataMatrix(nullptr)
00061
00062
                  if (rows == 0)
00063
                      throw std::length_error("Table rows must be greater than 0.");
00064
                  else if (cols == 0)
00065
                      throw std::length_error("Table columns must be greater than 0.");
00066
00067
                  dataMatrix = util::allocMatrix<T>(rows, cols);
                  if (dataMatrix == nullptr)
    throw std::bad_alloc();
00068
00069
00070
00071
                  colLabels.resize(_cols, std::string());
00072
              }
00073
00077
              ~DataTable()
00078
              {
00079
                  util::releaseMatrix(dataMatrix, rows);
08000
00081
00088
              std::string getColLabel(size_t colIndex)
00089
00090
                  if (colIndex >= colLabels.size())
00091
                       throw std::out_of_range("Column index out of range");
00092
00093
                  return colLabels[colIndex];
```

88 File Documentation

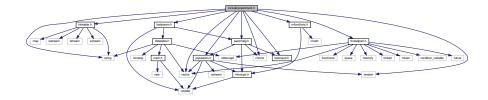
```
}
00095
00102
              void setColLabel(size_t colIndex, std::string newLabel)
00103
                   if (colIndex >= colLabels.size())
00104
                      throw std::out_of_range("Column index out of range");
00105
00106
00107
                  colLabels[colIndex] = newLabel;
00108
00109
              T getEntry(size_t row, size_t col)
00117
00118
                  if (dataMatrix == nullptr)
00119
00120
                      throw std::runtime_error("Data matrix not allocated");
00121
                  if (row >= rows)
00122
                      throw std::out_of_range("Table row out of range");
00123
                  else if (col >= cols)
                      throw std::out_of_range("Table column out of range");
00124
00126
                  return dataMatrix[row][col];
00127
              }
00128
00136
              void setEntry(size_t row, size_t col, T val)
00137
00138
                  if (dataMatrix == nullptr)
00139
                      throw std::runtime_error("Data matrix not allocated");
00140
                  if (row >= rows)
00141
                      throw std::out_of_range("Table row out of range");
00142
                  else if (col >= cols)
                      throw std::out_of_range("Table column out of range");
00143
00144
00145
                  dataMatrix[row][col] = val;
00146
00147
00155
              bool exportCSV(const char* filePath)
00156
00157
                  if (dataMatrix == nullptr) return false;
00159
                  using namespace std;
00160
                  ofstream outFile;
00161
                  outFile.open(filePath, ofstream::out | ofstream::trunc);
00162
                  if (!outFile.good()) return false;
00163
00164
                  // Print column labels
00165
                  for (unsigned int c = 0; c < cols; c++)
00166
00167
                      outFile << colLabels[c];</pre>
00168
                      if (c < cols - 1) outFile << ",";</pre>
00169
                  }
00170
                  outFile << endl;
00171
00172
00173
                  // Print data rows
00174
                  for (unsigned int r = 0; r < rows; r++)
00175
00176
                      for (unsigned int c = 0; c < cols; c++)
00178
                          outFile << std::setprecision(8) << dataMatrix[r][c];</pre>
00179
                          if (c < cols - 1) outFile << ",";
00180
                      outFile << endl:
00181
00182
00183
00184
                  outFile.close();
00185
00186
             }
00187
         private:
          size_t rows;
00188
00189
              size t cols:
00190
              std::vector<std::string> colLabels;
00191
              T** dataMatrix;
00193
00194 } // mdata
00195
00196 #endif
00197
00198 // ===========
00199 // End of datatable.h
00200 // ==========
```

7.5 include/experiment.h File Reference

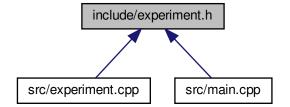
Header file for the Experiment class. Contains the basic logic and functions to run the cs471 project experiment.

```
#include <string>
#include <random>
#include <chrono>
#include <vector>
#include "mfunctions.h"
#include "inireader.h"
#include "population.h"
#include "threadpool.h"
#include "searchalg.h"
#include "testresult.h"
#include "testparam.h"
```

Include dependency graph for experiment.h:



This graph shows which files directly or indirectly include this file:



Classes

• struct mfunc::RandomBounds< T >

Simple struct for storing the minimum and maximum input vector bounds for a function.

class mfunc::Experiment< T >

Contains classes for running the CS471 project experiment.

Namespaces

• mfunc

90 File Documentation

7.5.1 Detailed Description

Header file for the Experiment class. Contains the basic logic and functions to run the cs471 project experiment.

Author

```
Andrew Dunn (Andrew. Dunn@cwu.edu)
```

Version

0.2

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file experiment.h.

7.6 experiment.h

```
00001
00013 #ifndef ___EXPERIMENT_H
00014 #define __EXPERIMENT_H
00016 #include <string>
00017 #include <random>
00018 #include <chrono>
00019 #include <vector>
00020 #include "mfunctions.h"
00021 #include "inireader.h"
00022 #include "population.h"
00023 #include "threadpool.h"
00024 #include "searchalg.h"
00025 #include "testresult.h"
00026 #include "testparam.h"
00027
00028 namespace mfunc
00029 {
00034
           template<class T>
00035
           struct RandomBounds
00036
00037
                T \min = 0.0;
00038
                T max = 0.0;
00039
00040
00051
           template<class T>
00052
           class Experiment
00053
           public:
00054
00055
               Experiment();
00056
                ~Experiment();
00057
                bool init(const char* paramFile);
                int testAllFunc();
00058
00059
                int testFuncThreaded(mdata::TestParameters<T> tParams);
00060
           private:
00061
               std::mutex popPoolMutex;
00062
                util::IniReader iniParams;
00063
                \verb|std::vector<mdata::Population<T>*>|populationsPool||
00064
                std::string resultsFile;
00065
                std::string execTimesFile;
00066
                RandomBounds<T>* vBounds;
00067
                ThreadPool* tPool;
```

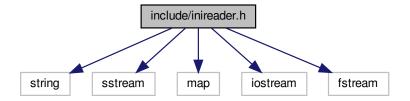
```
00068
               size_t iterations;
00069
               T alpha;
00070
               enums::Algorithm testAlg;
00071
00072
              mdata::Population<T>* popPoolRemove();
void popPoolAdd(mdata::Population<T>* popPtr);
00073
00074
00075
               bool parseFuncBounds();
00076
              bool allocatePopulationPool(size_t count, size_t popSize, size_t dimensions);
00077
00078
              void releasePopulationPool();
00079
08000
              bool allocateVBounds();
00081
              void releaseVBounds();
00082
00083
              bool allocateThreadPool(size_t numThreads);
00084
              void releaseThreadPool();
00085
00086 } // mfunc
00087
00088 #endif
00089
00090 // ========
00091 // End of experiment.h
00092 // ========
```

7.7 include/inireader.h File Reference

Header file for the IniReader class, which can open and parse simple *.ini files.

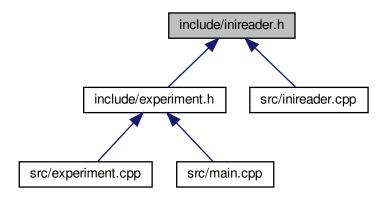
```
#include <string>
#include <sstream>
#include <map>
#include <iostream>
#include <fstream>
```

Include dependency graph for inireader.h:



92 File Documentation

This graph shows which files directly or indirectly include this file:



Classes

· class util::IniReader

The IniReader class is a simple *.ini file reader and parser.

Namespaces

• util

7.7.1 Detailed Description

Header file for the IniReader class, which can open and parse simple *.ini files.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file inireader.h.

7.8 inireader.h

7.8 inireader.h

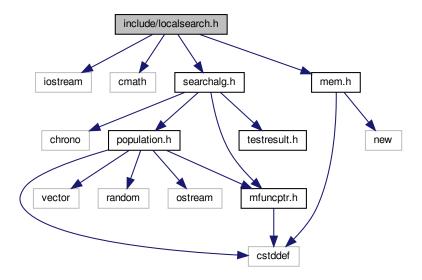
```
00001
00013 #ifndef ___INIREADER_H
00014 #define ___INIREADER_H
00015
00016 #include <string>
00017 #include <sstream>
00018 #include <map>
00019 #include <iostream>
00020 #include <fstream>
00021
00022 namespace util
00023 {
00046
          class IniReader
00047
00048
         public:
          IniReader();
00049
             ~IniReader();
00050
             bool openFile(std::string filePath);
00051
00052
             bool sectionExists(std::string section);
00053
             bool entryExists(std::string section, std::string entry);
00054
             std::string getEntry(std::string section, std::string entry);
00055
00056
             template <class T>
             T getEntryAs(std::string section, std::string entry)
00057
00058
             {
00059
                 std::stringstream ss(getEntry(section, entry));
00060
                 T retVal;
00061
                 ss >> retVal;
00062
                 return retVal;
00063
            }
       private:
00064
00065
            std::string file;
00066
              std::map<std::string, std::map<std::string>> iniMap;
00068
             bool parseFile();
00069
             void parseEntry(const std::string& sectionName, const std::string& entry);
00070
         };
00071 }
00072
00073 #endif
00074
00075 // ===========
00076 // End of inireader.h
00077 // ===========
```

7.9 include/localsearch.h File Reference

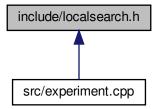
```
#include <iostream>
#include <cmath>
#include "searchalg.h"
#include "mem.h"
```

94 File Documentation

Include dependency graph for localsearch.h:



This graph shows which files directly or indirectly include this file:



Classes

• class mdata::LocalSearch< T>

The LocalSearch class implements the Local Search algorithm, which is ran using the overridden SearchAlgorithm ::run() function.

Namespaces

• mdata

Macros

• #define DEC_PRECISION 12

7.10 localsearch.h 95

7.9.1 Detailed Description

Author

Andrew Dunn (Andrew. Dunn@cwu.edu)

Version

0.1

Date

2019-04-19

Copyright

Copyright (c) 2019

Definition in file localsearch.h.

7.9.2 Macro Definition Documentation

7.9.2.1 DEC PRECISION

```
#define DEC_PRECISION 12
```

Definition at line 21 of file localsearch.h.

7.10 localsearch.h

```
00001
00012 #ifndef __LOCALSEARCH_H
00013 #define __LOCALSEARCH_H
00014
00015 #include <iostream>
00016 #include <cmath>
00017 #include "searchalg.h"
00018 #include "mem.h"
00019
00020 // Precision when checking neighbor fitness in number of decimal places right side of zero. 00021 \#define DEC_PRECISION 12
00022
00023 namespace mdata
00024 {
00031
           template<class T>
           class LocalSearch : public SearchAlgorithm<T>
00032
00033
00034
                using SearchAlgorithm<T>::startTimer;
00035
               using SearchAlgorithm<T>::stopTimer;
               const T MIN_IMPROVEMENT = pow(static_cast<T>(10), static_cast<T>(-1 *
DEC_PRECISION));
00037
00038
           public:
00049
               virtual TestResult<T> run(mfunc::mfuncPtr<T> funcPtr, const T
      fMin, const T fMax, Population<T>* const pop, const T alpha)
00050
```

96 File Documentation

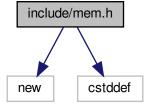
```
// Get population size and dimensions
00052
                   const size_t popSize = pop->getPopulationSize();
                   const size_t dimSize = pop->getDimensionsSize();
00053
00054
00055
                   // Make sure funcPtr is valid;
                   if (funcPtr == nullptr) return TestResult<T>(1, 0, 0.0); // Invalid function id,
00056
       return with error code 1
00057
00058
                   // Algorithm related variables
00059
                   bool stop = false;
                   size_t pIndex = 0;
00060
00061
00062
                   startTimer();
00063
00064
                   // Start recording execution time
00065
                   pop->generate(fMin, fMax);
00066
00067
                   for (size_t sol = 0; sol < popSize; sol++)</pre>
00068
00069
                        // Populate fitness values using given math function pointer
00070
                        if (!pop->calcFitness(sol, funcPtr))
00071
                            return TestResult<T>(2, 0, 0.0); // Invalid fitness index, return with
       error code 2
00072
00073
00074
                   \ensuremath{//} Get the index for the best fitness in the population
                   pIndex = pop->getBestFitnessIndex();
00075
00076
00077
                   \ensuremath{//} Get population vector and fitness of best solution
                   T* x = pop->getPopulationPtr(pIndex);
T xFit = pop->getFitness(pIndex);
00078
00079
08000
00081
                   // Create empty Y vector
                   T* y = util::allocArray<T>(dimSize);
T yFit = 0;
00082
00083
00084
                   // Create empty Z vector
T* z = util::allocArray<T>(dimSize);
00085
00086
00087
                   T zFit = 0;
88000
00089
                   if (x == nullptr || y == nullptr || z == nullptr)
00090
                   {
                        std::cerr << "Error in Local Search: Memory allocation failed" << std::endl;</pre>
00091
                        return TestResult<T>(3, 0, 0.0);
00092
00093
00094
00095
                   // Keep looping until search fails to improve
00096
                   while (!stop)
00097
                   {
00098
                        stop = true;
00099
00100
                        // Copy values from X vector into Y vector
00101
                        util::copyArray<T>(x, y, dimSize);
00102
                        // Loop through each dimension in vector
for (size_t a = 0; a < dimSize; a++)</pre>
00103
00104
00106
                             // Add alpha value to y[a]
00107
                            y[a] = x[a] + alpha;
00108
                            // Make sure y[a] is within the function bounds
00109
00110
                            lockBounds(y[a], fMin, fMax);
00111
00112
                            // Calculate fitness for y vector
00113
                            yFit = funcPtr(y, dimSize);
00114
00115
                            // Update Z[a] vector value based on the difference between Y and X \,
00116
                            z[a] = x[a] - (alpha * (yFit - xFit));
00117
00118
                             // Make sure z[a] is within the function bounds
00119
                            lockBounds(z[a], fMin, fMax);
00120
00121
                            y[a] = x[a]; // Reset y[a] to prepare for next loop
                        }
00122
00123
00124
                        zFit = funcPtr(z, dimSize);
00125
00126
                        // The following 'if' statement may cause extreme execution
                        // times for some functions due to floating point precision: // if (zFit < xFit)
00127
00128
00129
00130
                        // The replacement 'if' statement below places a limit
00131
                        // on the minimum acknowledged improvement fitness,
00132
                        // hopefully preventing extreme run-times:
00133
                        if (xFit - zFit > MIN_IMPROVEMENT)
00134
00135
                            // Z is an improvement on X, so keep searching
```

```
stop = false;
00137
                           // Swap Z and X for next loop
00138
                           T* tmp = x;
x = z;
xFit = zFit;
00139
00140
00141
00142
                           z = tmp;
00143
00144
                 }
00145
00146
                  // Return best result
00147
                   return TestResult<T>(0, xFit, stopTimer());
00148
00149
        private:
          void lockBounds(T& val, const T& min, const T& max)
00150
00151
                  if (val < min) val = min;
else if (val > max) val = max;
00152
00153
00154
00155
          };
00156 }
00157
00158 #endif
00159
00160 // =========
00161 // End of localsearch.h
00162 // ===========
```

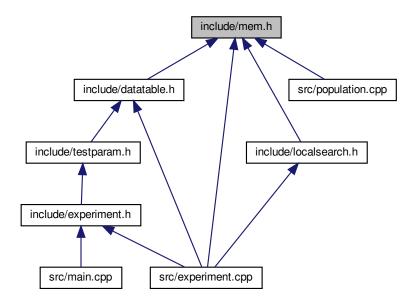
7.11 include/mem.h File Reference

Header file for various memory utility functions.

```
#include <new>
#include <cstddef>
Include dependency graph for mem.h:
```



This graph shows which files directly or indirectly include this file:



Namespaces

• util

Functions

```
    template < class T = double > void util::initArray (T *a, size_t size, T val)
```

Initializes an array with some set value.

• template<class T = double>

```
void util::initMatrix (T **m, size_t rows, size_t cols, T val)
```

Initializes a matrix with a set value for each entry.

template<class T = double>

```
void util::releaseArray (T *&a)
```

Releases an allocated array's memory and sets the pointer to nullptr.

• template<class T = double>

```
void util::releaseMatrix (T **&m, size_t rows)
```

Releases an allocated matrix's memory and sets the pointer to nullptr.

• template<class T = double>

```
T * util::allocArray (size_t size)
```

Allocates a new array of the given data type.

• template<class T = double>

```
T ** util::allocMatrix (size_t rows, size_t cols)
```

Allocates a new matrix of the given data type.

• template<class T = double>

```
void util::copyArray (T *src, T *dest, size_t size)
```

Copies the elements from one equal-sized array to another.

7.12 mem.h 99

7.11.1 Detailed Description

Header file for various memory utility functions.

Author

```
Andrew Dunn (Andrew . Dunn@cwu . edu)
```

Version

0.2

Date

2019-04-02

Copyright

Copyright (c) 2019

Definition in file mem.h.

7.12 mem.h

```
00001
00012 #ifndef __MEM_H
00013 #define __MEM_H
00015 #include <new> // std::nothrow
00016 #include <cstddef> // size_t definition
00017
00018 namespace util
00019 {
          template <class T = double>
00028
00029
          inline void initArray(T* a, size_t size, T val)
00030
00031
               if (a == nullptr) return;
00032
               for (size_t i = 0; i < size; i++)</pre>
00033
00034
00035
                   a[i] = val;
00036
00037
          }
00038
00048
          template <class T = double>
00049
          inline void initMatrix(T** m, size_t rows, size_t cols, T val)
00050
00051
               if (m == nullptr) return;
00052
               for (size_t i = 0; i < rows; i++)</pre>
00053
00054
00055
                   initArray(m[i], cols, val);
00056
00057
00058
00065
          template <class T = double >
          void releaseArray(T*& a)
00067
00068
               if (a == nullptr) return;
00069
00070
               delete[] a;
00071
00072
              a = nullptr;
00073
00081
          template <class T = double>
00082
          void releaseMatrix(T**& m, size_t rows)
```

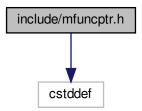
```
{
00084
              if (m == nullptr) return;
00085
              for (size_t i = 0; i < rows; i++)</pre>
00086
00087
00088
                  if (m[i] != nullptr)
                  {
00090
                      // Release each row
00091
                      releaseArray<T>(m[i]);
00092
00093
             }
00094
00095
              // Release columns
00096
              delete[] m;
00097
              m = nullptr;
00098
00099
00107
         template <class T = double>
00108
         inline T* allocArray(size_t size)
00109
00110
              return new(std::nothrow) T[size];
00111
00112
          template <class T = double>
00121
00122
          inline T** allocMatrix(size_t rows, size_t cols)
00123
00124
              T ** m = (T **) allocArray < T *> (rows);
00125
              if (m == nullptr) return nullptr;
00126
00127
              for (size_t i = 0; i < rows; i++)</pre>
00128
00129
                  m[i] = allocArray<T>(cols);
00130
                  if (m[i] == nullptr)
00131
00132
                      releaseMatrix<T>(m, rows);
00133
                      return nullptr;
00134
00135
             }
00136
00137
              return m;
00138
          }
00139
         template <class T = double>
00148
00149
         inline void copyArray(T* src, T* dest, size_t size)
00150
00151
              for (size_t i = 0; i < size; i++)</pre>
00152
                 dest[i] = src[i];
00153
         }
00154 }
00155
00156 #endif
00157
00158 // ==========
00159 // End of mem.h
00160 // ==========
```

7.13 include/mfuncptr.h File Reference

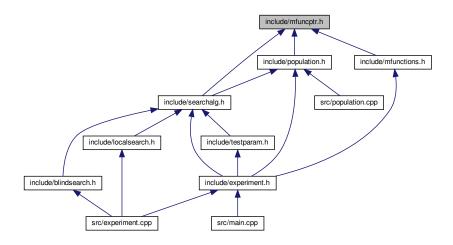
Contains the type definition for mfuncPtr, a templated function pointer to one of the math functions in mfunctions.h.

#include <cstddef>

Include dependency graph for mfuncptr.h:



This graph shows which files directly or indirectly include this file:



Namespaces

• mfunc

Typedefs

template < class T >
 using mfunc::mfuncPtr = T(*)(T *, size_t)

Function pointer that takes two arguments T* and size_t, and returns a T value.

7.13.1 Detailed Description

Contains the type definition for mfuncPtr, a templated function pointer to one of the math functions in mfunctions.h.

Author

```
Andrew Dunn (Andrew.Dunn@cwu.edu)
```

Version

0.1

Date

2019-04-19

Copyright

Copyright (c) 2019

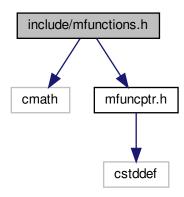
Definition in file mfuncptr.h.

7.14 mfuncptr.h

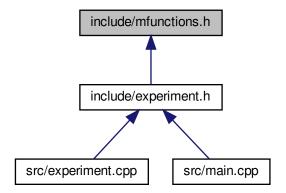
7.15 include/mfunctions.h File Reference

Contains various math function definitions.

```
#include <cmath>
#include "mfuncptr.h"
Include dependency graph for mfunctions.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct mfunc::FunctionDesc
 - get() returns a function's description Returns a C-string description for the given function id if the id is valid. Otherwise returns null
- struct mfunc::Functions< T >

Struct containing all static math functions. A function can be called directly by name, or indirectly using Functions::get or Functions::exec.

Namespaces

• mfunc

Macros

```
• #define USE MATH DEFINES
```

- #define _schwefelDesc "Schwefel's function"
- #define _dejongDesc "1st De Jong's function"
- #define _rosenbrokDesc "Rosenbrock"
- #define _rastriginDesc "Rastrigin"
- #define griewangkDesc "Griewangk"
- #define _sineEnvelopeSineWaveDesc "Sine Envelope Sine Wave"
- #define stretchedVSineWaveDesc "Stretched V Sine Wave"
- #define _ackleysOneDesc "Ackley's One"
- #define _ackleysTwoDesc "Ackley's Two"
- #define eggHolderDesc "Egg Holder"
- #define ranaDesc "Rana"
- #define _pathologicalDesc "Pathological"
- #define michalewiczDesc "Michalewicz"
- #define _mastersCosineWaveDesc "Masters Cosine Wave"
- #define _quarticDesc "Quartic"
- #define levyDesc "Levy"
- #define _stepDesc "Step"
- #define _alpineDesc "Alpine"

Variables

constexpr const unsigned int mfunc::NUM FUNCTIONS = 18

7.15.1 Detailed Description

Contains various math function definitions.

Author

Andrew Dunn (Andrew. Dunn@cwu.edu)

Version

0.1

Date

2019-03-29

Copyright

Copyright (c) 2019

Definition in file mfunctions.h.

7.15.2 Macro Definition Documentation

```
7.15.2.1 _ackleysOneDesc
#define _ackleysOneDesc "Ackley's One"
Definition at line 27 of file mfunctions.h.
Referenced by mfunc::FunctionDesc::get().
7.15.2.2 _ackleysTwoDesc
#define _ackleysTwoDesc "Ackley's Two"
Definition at line 28 of file mfunctions.h.
Referenced by mfunc::FunctionDesc::get().
7.15.2.3 _alpineDesc
#define _alpineDesc "Alpine"
Definition at line 37 of file mfunctions.h.
Referenced by mfunc::FunctionDesc::get().
7.15.2.4 _dejongDesc
#define _dejongDesc "1st De Jong's function"
Definition at line 21 of file mfunctions.h.
Referenced by mfunc::FunctionDesc::get().
7.15.2.5 _eggHolderDesc
#define _eggHolderDesc "Egg Holder"
Definition at line 29 of file mfunctions.h.
```

Referenced by mfunc::FunctionDesc::get().

```
7.15.2.6 _griewangkDesc
#define _griewangkDesc "Griewangk"
Definition at line 24 of file mfunctions.h.
Referenced by mfunc::FunctionDesc::get().
7.15.2.7 _levyDesc
#define _levyDesc "Levy"
Definition at line 35 of file mfunctions.h.
Referenced by mfunc::FunctionDesc::get().
7.15.2.8 _mastersCosineWaveDesc
#define _mastersCosineWaveDesc "Masters Cosine Wave"
Definition at line 33 of file mfunctions.h.
Referenced by mfunc::FunctionDesc::get().
7.15.2.9 _michalewiczDesc
#define _michalewiczDesc "Michalewicz"
Definition at line 32 of file mfunctions.h.
Referenced by mfunc::FunctionDesc::get().
7.15.2.10 _pathologicalDesc
#define _pathologicalDesc "Pathological"
Definition at line 31 of file mfunctions.h.
```

Referenced by mfunc::FunctionDesc::get().

```
7.15.2.11 _quarticDesc
#define _quarticDesc "Quartic"
Definition at line 34 of file mfunctions.h.
Referenced by mfunc::FunctionDesc::get().
7.15.2.12 _ranaDesc
#define _ranaDesc "Rana"
Definition at line 30 of file mfunctions.h.
Referenced by mfunc::FunctionDesc::get().
7.15.2.13 _rastriginDesc
#define _rastriginDesc "Rastrigin"
Definition at line 23 of file mfunctions.h.
Referenced by mfunc::FunctionDesc::get().
7.15.2.14 _rosenbrokDesc
#define _rosenbrokDesc "Rosenbrock"
Definition at line 22 of file mfunctions.h.
Referenced by mfunc::FunctionDesc::get().
7.15.2.15 _schwefelDesc
#define _schwefelDesc "Schwefel's function"
Definition at line 20 of file mfunctions.h.
Referenced by mfunc::FunctionDesc::get().
```

7.15.2.16 _sineEnvelopeSineWaveDesc #define _sineEnvelopeSineWaveDesc "Sine Envelope Sine Wave" Definition at line 25 of file mfunctions.h. Referenced by mfunc::FunctionDesc::get(). 7.15.2.17 _stepDesc #define _stepDesc "Step" Definition at line 36 of file mfunctions.h. Referenced by mfunc::FunctionDesc::get(). 7.15.2.18 _stretchedVSineWaveDesc #define _stretchedVSineWaveDesc "Stretched V Sine Wave" Definition at line 26 of file mfunctions.h. Referenced by mfunc::FunctionDesc::get(). 7.15.2.19 _USE_MATH_DEFINES #define _USE_MATH_DEFINES Definition at line 15 of file mfunctions.h.

7.16 mfunctions.h

7.16 mfunctions.h

```
00001
00012 #ifndef __MFUNCIONS_H
00013 #define __MFUNCIONS_H
00014
00015 #define _USE_MATH_DEFINES
00016
00017 #include <cmath>
00018 #include "mfuncptr.h"
00019
00020 #define _schwefelDesc "Schwefel's function"
00021 #define _dejongDesc "1st De Jong's function"
00022 #define _rosenbrokDesc "Rosenbrock"
00023 #define _rastriginDesc "Rastrigin"
00024 #define _griewangkDesc "Griewangk"
00025 #define _sineEnvelopeSineWaveDesc "Sine Envelope Sine Wave"
00026 #define _stretchedVSineWaveDesc "Stretched V Sine Wave"
00027 #define _ackleysOneDesc "Ackley's One"
00028 #define _ackleysTwoDesc "Ackley's Two"
00029 #define _eggHolderDesc "Egg Holder"
00030 #define _ranaDesc "Rana"
00031 #define _pathologicalDesc "Pathological"
00032 #define _michalewiczDesc "Michalewicz"
00033 #define _mastersCosineWaveDesc "Masters Cosine Wave" 00034 #define _quarticDesc "Quartic"
00035 #define _levyDesc "Levy"
00036 #define _stepDesc "Step"
00037 #define _alpineDesc "Alpine"
00038
00042 namespace mfunc
00043 {
            constexpr const unsigned int NUM_FUNCTIONS = 18;
00047
00048
00056
            struct FunctionDesc
00057
00058
                 static const char* get(unsigned int f)
00059
                 {
00060
                      switch (f)
00061
00062
                           case 1:
00063
                              return _schwefelDesc;
00064
                           case 2:
00065
                               return _dejongDesc;
00066
                           case 3:
00067
                              return _rosenbrokDesc;
00068
                           case 4:
00069
                               return _rastriginDesc;
00070
                           case 5:
00071
                               return _griewangkDesc;
00072
                           case 6:
00073
                               return _sineEnvelopeSineWaveDesc;
00074
                           case 7:
00075
                              return _stretchedVSineWaveDesc;
00076
                           case 8:
00077
                               return _ackleysOneDesc;
00078
                           case 9:
00079
                              return _ackleysTwoDesc;
00080
                           case 10:
00081
                               return _eggHolderDesc;
00082
                           case 11:
00083
                               return _ranaDesc;
00084
                           case 12:
00085
                              return _pathologicalDesc;
00086
                           case 13:
00087
                              return _michalewiczDesc;
00088
                           case 14:
00089
                               return _mastersCosineWaveDesc;
00090
                           case 15:
00091
                               return _quarticDesc;
00092
                           case 16:
00093
                               return levyDesc;
00094
                           case 17:
00095
                               return _stepDesc;
00096
                           case 18:
00097
                               return _alpineDesc;
                           default:
00098
00099
                               return NULL;
00100
                      }
00101
00102
00103
00111
            template <class T>
00112
            struct Functions
00113
00114
                 static T schwefel(T* v, size_t n);
```

```
static T dejong(T* v, size_t n);
              static T rosenbrok(T* v, size_t n);
static T rastrigin(T* v, size_t n);
00116
00117
              static T griewangk(T* v, size_t n);
static T sineEnvelopeSineWave(T* v, size_t n);
static T stretchedVSineWave(T* v, size_t n);
00118
00119
00120
              static T ackleysOne(T* v, size_t n);
00121
00122
               static T ackleysTwo(T* v, size_t n);
00123
               static T eggHolder(T* v, size_t n);
               static T rana(T* v, size_t n);
00124
              static T pathological(T* v, size_t n);
static T mastersCosineWave(T* v, size_t n);
static T michalewicz(T* v, size_t n);
00125
00126
00127
00128
              static T quartic(T* v, size_t n);
              static T quartic(T* v, size_t n);
static T levy(T* v, size_t n);
static T step(T* v, size_t n);
static T alpine(T* v, size_t n);
static mfuncPtr<T> get(unsigned int f);
00129
00130
00131
00132
              static bool exec(unsigned int f, T* v, size_t n, T& outResult);
00133
00134
               static T nthroot(T x, T n);
00135
               static T w(T x);
00136
          };
00137 }
00138
00145 template <class T>
00146 T mfunc::Functions<T>::nthroot(T x, T n)
00147 {
00148
           return pow(x, static_cast<T>(1.0) / n);
00149 }
00150
00151 // ======
00152
00160 template <class T>
00161 T mfunc::Functions<T>::schwefel(T* v, size_t n)
00162 {
           T f = 0.0;
00163
00164
00165
           for (size_t i = 0; i < n; i++)</pre>
00166
          {
00167
               f \leftarrow (static\_cast < T > (-1.0) * v[i]) * sin(sqrt(std::abs(v[i])));
00168
00169
00170
           return (static_cast<T>(418.9829) * static cast<T>(n)) - f;
00171 }
00172
00173 // ===========
00174
00182 template <class T>
00183 T mfunc::Functions<T>::dejong(T* v, size_t n)
00184 {
00185
           T f = 0.0;
00186
00187
           for (size_t i = 0; i < n; i++)</pre>
00188
              f += v[i] * v[i];
00189
00190
          }
00191
00192
           return f:
00193 }
00194
00196
00204 template <class T>
00205 T mfunc::Functions<T>::rosenbrok(T* v, size_t n)
00206 {
00207
           T f = 0.0;
00208
           for (size_t i = 0; i < n - 1; i++)</pre>
00209
00210
          {
00211
               T a = ((v[i] * v[i]) - v[i+1]);
              T b = (static_cast<T>(1.0) - v[i]);
f += static_cast<T>(100.0) * a * a;
00212
00213
               f += b * b;
00214
          }
00215
00216
00217
           return f:
00218 }
00219
00221
00229 template <class T>
00230 T mfunc::Functions<T>::rastrigin(T* v, size_t n)
00231 {
00232
           T f = 0.0;
00233
           for (size_t i = 0; i < n; i++)</pre>
00234
00235
```

7.16 mfunctions.h

```
00236
                          f += (v[i] * v[i]) - (static\_cast < T > (10.0) * cos(static\_cast < T > (2.0) * static\_cast < T > (M_PI) * v[i]) 
          );
00237
00238
00239
                  return static cast<T>(10.0) * static cast<T>(n) * f;
00240 }
00241
00243
00251 template <class T>
00252 T mfunc::Functions<T>::griewangk(T* v, size_t n)
00253 {
00254
                  T sum = 0.0;
                  T product = 0.0;
00255
00256
00257
                  for (size_t i = 0; i < n; i++)</pre>
00258
00259
                         sum += (v[i] * v[i]) / static cast<T>(4000.0);
00260
00261
00262
                  for (size_t i = 0; i < n; i++)</pre>
00263
00264
                         product *= cos(v[i] / sqrt(static_cast<T>(i + 1.0)));
00265
00266
00267
                  return static_cast<T>(1.0) + sum - product;
00268 }
00269
00270 // ===========
00271
00279 template <class T>
00280 T mfunc::Functions<T>::sineEnvelopeSineWave(T* v, size_t n)
00281 {
00282
                  T f = 0.0;
00283
                  for (size_t i = 0; i < n - 1; i++)</pre>
00284
00285
                  {
00286
                         T = \sin(v[i] * v[i] + v[i+1] * v[i+1] - static_cast<T>(0.5));
00287
                         a *= a;
00288
                          \texttt{T b = (static\_cast<T>(1.0) + static\_cast<T>(0.001) * (v[i] * v[i] + v[i+1] * v[i+1])); } 
                        b *= b;
00289
00290
                         f += static cast<T>(0.5) + (a / b);
00291
                 }
00292
00293
                  return static_cast<T>(-1.0) * f;
00294 }
00295
00297
00305 template <class T>
00306 T mfunc::Functions<T>::stretchedVSineWave(T* v, size_t n)
00307 {
00308
                  T f = 0.0;
00309
                  for (size_t i = 0; i < n - 1; i++)</pre>
00310
00311
00312
                         T = nthroot(v[i]*v[i] + v[i+1]*v[i+1], static_cast<T>(4.0));
00313
                         T b = \sin(\text{static\_cast} < T) < (50.0) * nthroot(v[i] * v[i] + v[i+1] * v[i+1], static\_cast < T) < (10.0)));
00314
                        b *= b;
00315
                         f += a * b + static\_cast<T>(1.0);
00316
                  }
00317
00318
                  return f;
00319 }
00320
00321 // ==========
00322
00330 template <class T>
00331 T mfunc::Functions<T>::ackleysOne(T* v, size_t n)
00332 {
00333
                  T f = 0.0;
00334
00335
                  for (size_t i = 0; i < n - 1; i++)</pre>
00336
                          \texttt{T a = (static\_cast<T>(1.0) / pow(static\_cast<T>(M\_E), static\_cast<T>(0.2))) * sqrt(v[i]*v[i] + v[i+1] * v[
00337
          1]*v[i+1]);
00338
                         T b = static_cast<T>(3.0) * (cos(static_cast<T>(2.0) * v[i]) + sin(static_cast<T>(2.0) * v[i+1]));
00339
                         f += a + b;
00340
                  }
00341
00342
                  return f;
00343 }
00344
00345 // ==============
00346
00354 template <class T>
00355 T mfunc::Functions<T>::ackleysTwo(T* v, size_t n)
```

```
00356 {
 00357
                                                        T f = 0.0;
00358
00359
                                                        for (size_t i = 0; i < n - 1; i++)</pre>
00360
                                                                          T = static\_cast < T > (20.0) / pow(static\_cast < T > (M_E), static\_cast < T > (0.2) * sqrt((v[i] * v[i] + v[i+1]) 
00361
                                 ]*v[i+1]) / static_cast<T>(2.0)));
00362
                                                                              T b = pow(static_cast<T>(M_E), static_cast<T>(0.5) \star
00363
                                                                                                   (\cos(\text{static\_cast} < \text{T} > (2.0) * \text{static\_cast} < \text{T} > (M\_PI) * v[i]) + \cos(\text{static\_cast} < \text{T} > (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) * (2.0) 
                                 static_cast<T>(M_PI) * v[i+1])));
00364
                                                                              f += static_cast<T>(20.0) + static_cast<T>(M E) - a - b;
00365
 00366
 00367
 00368 }
 00369
00371
 00379 template <class T>
 00380 T mfunc::Functions<T>::eggHolder(T* v, size_t n)
 00381 {
00382
                                                        T f = 0.0;
00383
                                                        for (size_t i = 0; i < n - 1; i++)</pre>
00384
 00385
                                                                               \texttt{T a = static\_cast<T>(-1.0) * v[i] * sin(sqrt(std::abs(v[i] - v[i+1] - static\_cast<T>(47.0)))); } 
 00386
 00387
                                                                             \texttt{T b = (v[i+1] + static\_cast<T>(47)) * sin(sqrt(std::abs(v[i+1] + static\_cast<T>(47.0) + (v[i]/cast<T>(47.0) + (v[i]/cast<T>(47.
                                static_cast<T>(2.0))));
00388
                                                                          f += a - b;
00389
 00390
 00391
                                                       return f;
 00392 }
 00393
00394 // =======
00395
00403 template <class T>
 00404 T mfunc::Functions<T>::rana(T* v, size_t n)
 00405 {
 00406
                                                        T f = 0.0;
00407
00408
                                                        for (size t i = 0; i < n - 1; i++)
00409
                                                                             \texttt{T a = v[i]} * \sin(\operatorname{sqrt}(\operatorname{std}:\operatorname{abs}(\operatorname{v[i+1]} - \operatorname{v[i]} + \operatorname{static\_cast}(\operatorname{T}(1.0)))) * \cos(\operatorname{sqrt}(\operatorname{std}:\operatorname{abs}(\operatorname{v[i+1]} + \operatorname{static\_cast}(\operatorname{T}(1.0)))) * (\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname{std}(\operatorname
00410
                                 v[i] + static_cast<T>(1.0)));
00411
                                                                              T \ b = (v[i+1] \ + \ static\_cast < T > (1.0)) \ * \ cos(sqrt(std::abs(v[i+1] \ - \ v[i] \ + \ static\_cast < T > (1.0)))) \ * \ * \ track(theorem = 1) \ 
                                sin(sqrt(std::abs(v[i+1] + v[i] + static_cast<T>(1.0))));
 00412
                                                                           f += a + b;
00413
 00414
 00415
                                                      return f;
 00416 }
 00417
 00419
 00427 template <class T>
 00428 T mfunc::Functions<T>::pathological(T* v, size_t n)
 00429 {
 00430
                                                        T f = 0.0;
00431
00432
                                                        for (size t i = 0; i < n - 1; i++)
00433
                                                        {
 00434
                                                                             T a = \sin(\operatorname{sqrt}(\operatorname{static\_cast} < T > (100.0) *v[i] *v[i] + v[i+1] *v[i+1]));
 00435
                                                                              a = (a*a) - static\_cast<T>(0.5);
 00436
                                                                              T b = (v[i] * v[i] - static\_cast < T > (2) * v[i] * v[i+1] + v[i+1] * v[i+1]); 
                                                                          b = static_cast<T>(1.0) + static_cast<T>(0.001) * b*b;
00437
00438
                                                                            f += static_cast<T>(0.5) + (a/b);
00439
                                                       }
00440
 00441
                                                       return f;
 00442 }
00443
00445
 00453 template <class T>
 00454 T mfunc::Functions<T>::michalewicz(T* v, size_t n)
 00455 {
                                                        T f = 0.0;
 00456
00457
00458
                                                        for (size t i = 0: i < n: i++)
 00459
 00460
                                                                              f += \sin(v[i]) * pow(\sin(((i+1) * v[i]) * v[i]) / static_cast<T>(M_PI)), static_cast<T>(20));
 00461
 00462
00463
                                                        return -1.0 * f;
00464 }
00465
```

7.16 mfunctions.h

```
00467
00475 template <class T>
00476 T mfunc::Functions<T>::mastersCosineWave(T* v, size_t n)
00477 {
00478
                   T f = 0.0;
00479
00480
                   for (size_t i = 0; i < n - 1; i++)</pre>
00481
                           \texttt{T a = pow(M\_E, static\_cast<T>(-1.0/8.0)*(v[i]*v[i] + v[i+1]*v[i+1] + static\_cast<T>(0.5)*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]*v[i+1]
00482
           ]));
00483
                          T b = \cos(\text{static cast} < T > (4) * <math>\operatorname{sgrt}(y[i] * y[i] + y[i+1] * y[i+1] + \operatorname{static cast} < T > (0.5) * y[i] * y[i+1]));
00484
                          f += a * b;
00485
00486
00487
                   return static_cast<T>(-1.0) * f;
00488 }
00489
00491
00499 template <class T>
00500 T mfunc::Functions<T>::quartic(T* v, size_t n)
00501 {
00502
                   T f = 0.0:
00503
00504
                   for (size_t i = 0; i < n; i++)</pre>
00505
00506
                           f += (i+1) * v[i] * v[i] * v[i] * v[i];
00507
                   }
00508
00509
                   return f:
00510 }
00511
00512 // =======
00513
00517 template <class T>
00518 T mfunc::Functions<T>::w(T x)
00519 {
00520
                   return static_cast<T>(1.0) + (x - static_cast<T>(1.0)) / static_cast<T>(4.0);
00521 }
00522
00530 template <class T>
00531 T mfunc::Functions<T>::levy(T* v, size_t n)
00532 {
00533
00534
00535
                   for (size_t i = 0; i < n - 1; i++)</pre>
00536
                          T = w(v[i]) - static_cast<T>(1.0);
00537
00538
                           a *= a;
00539
                           T b = sin(static_cast< T>(M_PI) * w(v[i]) + static_cast< T>(1.0));
00540
                           b *= b;
00541
                           T c = w(v[n - 1]) - static_cast<T>(1.0);
00542
00543
                          T d = sin(static\_cast<T>(2.0) * static\_cast<T>(M_PI) * w(v[n - 1]));
00544
                           d *= d;
00545
                           f += a * (static_cast<T>(1.0) + static_cast<T>(10.0) * b) + c * (static_cast<T>(1.0) + d);
00546
                   }
00547
00548
                   T = sin(static_cast<T>(M_PI) * w(v[0]));
                   return e*e + f;
00549
00550 }
00551
00552 // ========
00553
00561 template <class T>
00562 T mfunc::Functions<T>::step(T* v, size_t n)
00563 {
00564
                   T f = 0.0;
00565
00566
                   for (size_t i = 0; i < n; i++)</pre>
00567
00568
                          T = std::abs(v[i]) + static_cast<T>(0.5);
00569
                           f += a * a;
00570
                   }
00571
00572
                   return f;
00573 }
00574
00575 // =======
00576
00584 template <class T>
00585 T mfunc::Functions<T>::alpine(T* v, size_t n)
00586 {
00587
                   T f = 0.0;
00588
00589
                   for (size t i = 0; i < n; i++)
```

```
{
             f \leftarrow std::abs(v[i] + sin(v[i]) + static_cast<T>(0.1)*v[i]);
00592
00593
00594
         return f;
00595 }
00596
00598
00608 template <class T>
00609 mfunc::mfuncPtr<T> mfunc::Functions<T>::get(unsigned int f)
00610 {
00611
         switch (f)
00612
00613
             case 1:
00614
                return Functions<T>::schwefel;
00615
             case 2:
00616
               return Functions<T>::dejong;
00617
             case 3:
                return Functions<T>::rosenbrok;
00619
             case 4:
00620
                return Functions<T>::rastrigin;
             case 5:
00621
00622
               return Functions<T>::griewangk;
00623
             case 6:
00624
               return Functions<T>::sineEnvelopeSineWave;
00625
             case 7:
00626
                return Functions<T>::stretchedVSineWave;
00627
             case 8:
00628
               return Functions<T>::acklevsOne;
00629
             case 9:
00630
                return Functions<T>::ackleysTwo;
00631
             case 10:
00632
                return Functions<T>::eggHolder;
00633
             case 11:
                return Functions<T>::rana;
00634
00635
             case 12:
00636
               return Functions<T>::pathological;
             case 13:
00638
                return Functions<T>::michalewicz;
00639
             case 14:
00640
                return Functions<T>::mastersCosineWave;
00641
             case 15:
00642
               return Functions<T>::quartic;
             case 16:
00644
                return Functions<T>::levy;
00645
             case 17:
00646
               return Functions<T>::step;
             case 18:
00647
00648
                return Functions<T>::alpine;
             default:
00649
00650
               return nullptr;
00651
        }
00652 }
00653
00654 // ========
00666 template <class T>
00667 bool mfunc::Functions<T>::exec(unsigned int f, T* v, size_t n, T& outResult)
00668 {
00669
        auto fPtr = get(f);
if (fPtr == nullptr) return false;
00670
00671
00672
       outResult = fPtr(v, n);
00673
         return true;
00674 }
00675
00676 #endif
00677
00679 // End of mfunctions.h
00680 // ====
```

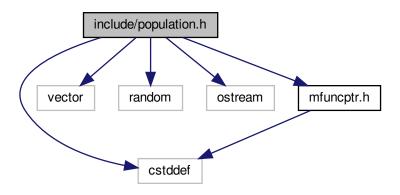
7.17 include/population.h File Reference

Header file for the Population class. Stores a population and resulting fitness values.

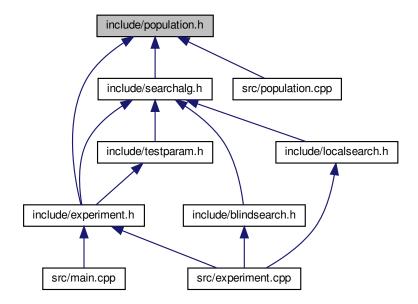
```
#include <cstddef>
#include <vector>
```

```
#include <random>
#include <ostream>
#include "mfuncptr.h"
```

Include dependency graph for population.h:



This graph shows which files directly or indirectly include this file:



Classes

class mdata::Population

Data class for storing a multi-dimensional population of data with the associated fitness.

Namespaces

• mdata

7.17.1 Detailed Description

Header file for the Population class. Stores a population and resulting fitness values.

Author

```
Andrew Dunn (Andrew. Dunn@cwu.edu)
```

Version

0.2

Date

2019-04-04

Copyright

Copyright (c) 2019

Definition in file population.h.

7.18 population.h

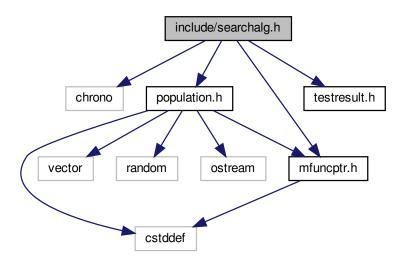
```
00001
00012 #ifndef ___POPULATION_H
00013 #define __POPULATION_H
00014
00015 #include <cstddef> // size_t definition
00016 #include <vector>
00017 #include <random>
00018 #include <ostream>
00019 #include "mfuncptr.h"
00020
00021 namespace mdata
00022 {
00029
            template<class T>
00030
           class Population
00031
00032
           public:
00033
                Population(size_t popSize, size_t dimensions);
00034
00035
                ~Population();
00036
                bool isReady();
                size_t getPopulationSize();
00037
00038
                 size_t getDimensionsSize();
00039
                T* getPopulationPtr(size_t popIndex);
00040
00041
                bool generate (T minBound, T maxBound);
                bool generate(1 minocuna, 1 manocuna,)
bool setFitness(size_t popIndex, T value);
bool calcFitness(size_t popIndex, mfunc::mfuncPtr<T> funcPtr);
00042
00043
00044
00045
                 T getFitness(size_t popIndex);
                T* getFitnessPtr(size_t popIndex);
std::vector<T> getAllFitness();
00046
00047
                 T* getBestFitnessPtr();
00048
00049
                size_t getBestFitnessIndex();
00050
```

```
00051
              void outputPopulation(std::ostream& outStream, const char* delim, const char*
     lineBreak);
00052
             void outputFitness(std::ostream& outStream, const char* delim, const char* lineBreak);
         private:
00053
         const size_t popSize;
const size_t popDim;
00054
00055
            T** popMatrix;
T* popFitness;
00058
00060
            std::random_device rdev;
00061
             std::mt19937 rgen;
             bool allocPopMatrix();
void releasePopMatrix();
00063
00064
00065
00066
             bool allocPopFitness();
00067
              void releasePopFitness();
00068
         };
00069 }
00070
00071 #endif
00072
00073 // ===
```

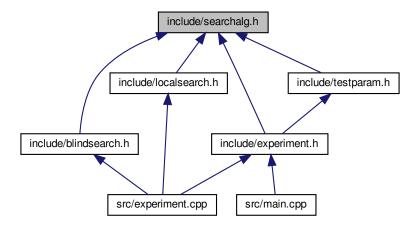
7.19 include/searchalg.h File Reference

Defines the SearchAlgorithm class, Algorithm enum, and AlgorithmNames struct. The SearchAlgorithm class serves as a base class for implemented search algorithms.

```
#include <chrono>
#include "population.h"
#include "testresult.h"
#include "mfuncptr.h"
Include dependency graph for searchalg.h:
```



This graph shows which files directly or indirectly include this file:



Classes

· struct enums::AlgorithmNames

Struct that contains constant string names for the different search algorithms.

class mdata::SearchAlgorithm< T >

The SearchAlgorithm class is used as a base class for other implemented search algorithms. Provides a common interface to run each algorithm.

Namespaces

- enums
- mdata

Enumerations

• enum enums::Algorithm { enums::BlindSearch = 0, enums::LocalSearch = 1, enums::Count = 2 } Enum of different available search algorithms.

7.19.1 Detailed Description

Defines the SearchAlgorithm class, Algorithm enum, and AlgorithmNames struct. The SearchAlgorithm class serves as a base class for implemented search algorithms.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

7.20 searchalg.h

Version

0.1

Date

2019-04-19

Copyright

Copyright (c) 2019

Definition in file searchalg.h.

7.20 searchalg.h

```
00013 #ifndef ___SEARCHALG_H
00014 #define ___SEARCHALG_H
00015
00016 #include <chrono>
00017 #include "population.h"
00018 #include "testresult.h"
00019 #include "mfuncptr.h"
00020
00021 using namespace std::chrono;
00022
00023 namespace enums
00024 {
00028
           enum Algorithm
00029
00030
                BlindSearch = 0,
00031
                LocalSearch = 1,
00032
                Count = 2
00033
           } ;
00039
           struct AlgorithmNames
00040
                static constexpr const char* BLIND_SEARCH = "Blind Search";
static constexpr const char* LOCAL_SEARCH = "Local Search";
00041
00042
00043
00044
                static const char* get(Algorithm alg)
00045
00046
                     switch (alg)
00047
                         case Algorithm::BlindSearch:
00048
00049
                            return BLIND_SEARCH;
00050
                         case Algorithm::LocalSearch:
00051
                             return LOCAL_SEARCH;
00052
                         default:
                             return "";
00053
00054
                             break;
00055
                    }
00056
                }
           };
00058 }
00059
00060 namespace mdata
00061 {
00069
           template<class T>
           class SearchAlgorithm
00071
00072
           public:
               SearchAlgorithm() : timeDiff(0.0) {}
virtual ~SearchAlgorithm() = 0;
virtual TestResult<T> run(mfunc::mfuncPtr<T> funcPtr, const T fMin,
00073
00074
00075
      const T fMax, Population<T>* const pop, const T alpha) = 0;
00076
        protected:
00077
                double timeDiff;
00078
                high_resolution_clock::time_point timer;
00079
00083
                void startTimer()
00084
00085
                     timer = high_resolution_clock::now();
```

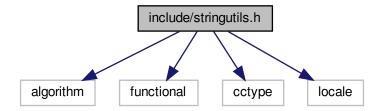
```
00086
               }
00087
00091
               double stopTimer()
00092
               {
00093
                    high_resolution_clock::time_point t_end = high_resolution_clock::now();
00094
                    return static_cast<double>(duration_cast<nanoseconds>(t_end - timer).count()) / 1000000.0;
00095
00096
           } ;
00097 }
00098
00099 // Trivial implementation of pure-virtual destructor 00100 // as required by the C++ language 00101 template<class T>
00102 mdata::SearchAlgorithm<T>::~SearchAlgorithm() { }
00103
00104 #endif
00105
00106 // =======
00107 // End of searchalg.h
00108 // =========
```

7.21 include/stringutils.h File Reference

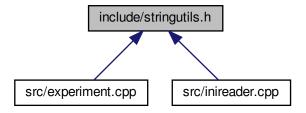
Contains various string manipulation helper functions.

```
#include <algorithm>
#include <functional>
#include <cctype>
#include <locale>
```

Include dependency graph for stringutils.h:



This graph shows which files directly or indirectly include this file:



7.22 stringutils.h

Namespaces

• util

7.21.1 Detailed Description

Contains various string manipulation helper functions.

Author

```
Evan Teran (https://github.com/eteran)
```

Date

2019-04-01

Definition in file stringutils.h.

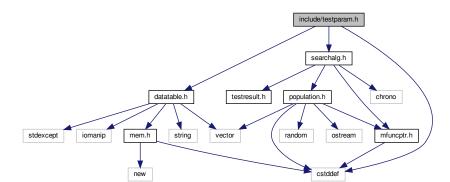
7.22 stringutils.h

```
00008 #ifndef __STRINGUTILS_H
00009 #define __STRINGUTILS_H
00010
00011 #include <algorithm>
00012 #include <functional>
00013 #include <cctype>
00014 #include <locale>
00015
00016 namespace util
00017 {
00018
00019
         // The string functions below were written by Evan Teran
00020
         // from Stack Overflow:
00021
         // https://stackoverflow.com/questions/216823/whats-the-best-way-to-trim-stdstring
00022
00023
00024
         // trim from start (in place)
00025
         static inline void s_ltrim(std::string &s) {
00026
             s.erase(s.begin(), std::find_if(s.begin(), s.end(),
00027
                     std::not1(std::ptr_fun<int, int>(std::isspace))));
00028
00029
00030
         // trim from end (in place)
00031
         static inline void s rtrim(std::string &s) {
00032
           s.erase(std::find_if(s.rbegin(), s.rend(),
00033
                     std::not1(std::ptr_fun<int, int>(std::isspace))).base(), s.end());
00034
00035
00036
         // trim from both ends (in place)
00037
         static inline void s_trim(std::string &s) {
00038
             s_ltrim(s);
00039
             s_rtrim(s);
00040
00041
         // trim from start (copying)
00042
00043
         static inline std::string s_ltrim_copy(std::string s) {
00044
          s ltrim(s);
00045
             return s;
00046
00047
         // trim from end (copying)
00048
00049
         static inline std::string s_rtrim_copy(std::string s) {
00050
             s rtrim(s);
00051
             return s;
00052
00053
         // trim from both ends (copying)
00054
00055
         static inline std::string s_trim_copy(std::string s) {
00056
            s_trim(s);
00057
             return s;
00058
00059 }
00060 #endif
00061
00062 // =========
00063 // End of stringutils.h
00064 // ==
```

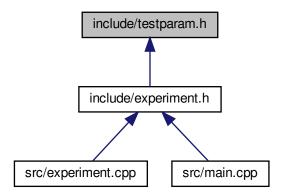
7.23 include/testparam.h File Reference

Contains the definition of the TestParameters struct, which is a data type used to transfer test parameters between functions.

```
#include <cstddef>
#include "datatable.h"
#include "searchalg.h"
Include dependency graph for testparam.h:
```



This graph shows which files directly or indirectly include this file:



Classes

struct mdata::TestParameters< T >

Packs together various test experiment parameters.

Namespaces

mdata

7.24 testparam.h

7.23.1 Detailed Description

Contains the definition of the TestParameters struct, which is a data type used to transfer test parameters between functions.

Author

```
Andrew Dunn (Andrew.Dunn@cwu.edu)
```

Version

0.1

Date

2019-04-20

Copyright

Copyright (c) 2019

Definition in file testparam.h.

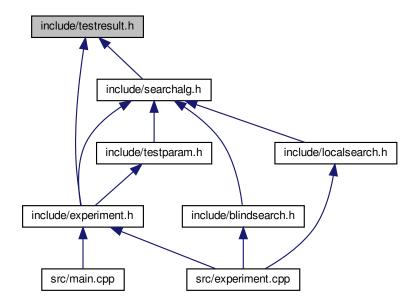
7.24 testparam.h

```
00001
00014 #ifndef __TESTPARAM_H
00015 #define __TESTPARAM_H
00017 #include <cstddef> // size_t definition
00018 #include "datatable.h"
00019 #include "searchalg.h"
00020
00021 namespace mdata
00022 {
00028
            template <class T>
00029
           struct TestParameters
00030
00031
                unsigned int funcId;
00032
                T alpha;
00033
               unsigned int resultsCol;
00034
               unsigned int execTimesCol;
               size_t resultsRow;
size_t execTimesRow;
00035
00036
00037
               DataTable<T>* resultsTable;
DataTable<T>* execTimesTable;
00038
00039
                enums::Algorithm alg;
00041
                TestParameters()
00042
00043
                    funcId = 1;
                    alpha = 0;
alg = enums::Algorithm::BlindSearch;
00044
00045
00046
                    resultsTable = nullptr;
00047
                    execTimesTable = nullptr;
00048
                    resultsCol = 0;
00049
                     execTimesCol = 0;
00050
                     resultsRow = 0;
00051
                     execTimesRow = 0:
00052
                }
00053
           };
00054 }
00055
00056 #endif
00057
00058 // =========
00059 // End of testparam.h
00060 // ==
```

7.25 include/testresult.h File Reference

Simple structure that packs together various return values for the search algorithms. functions.

This graph shows which files directly or indirectly include this file:



Classes

struct mdata::TestResult< T >

Namespaces

• mdata

7.25.1 Detailed Description

Simple structure that packs together various return values for the search algorithms. functions.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-19

Copyright

Copyright (c) 2019

Definition in file testresult.h.

7.26 testresult.h

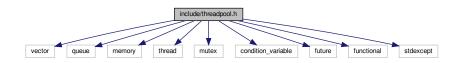
7.26 testresult.h

```
00001
00014 #ifndef ___TESTRESULT_H
00015 #define __TESTRESULT_H
00016
00017 namespace mdata
00018 {
00019
          template<class T>
00020
          struct TestResult
00021
          {
              const int err; // Error code. 0 = no error.
const T fitness; // Fitness result
00022
00023
00024
              const double execTime; // Algorithm execution time in miliseconds
00025
              TestResult(int _err, T _fitness, double _execTime) : err(_err), fitness(_fitness),
00026
     execTime(_execTime)
00027
00028
00029
00030 } // mdata
00031
00032 #endif
00033
00034 // ===
00035 // End of testresult.h
00036 // ==========
```

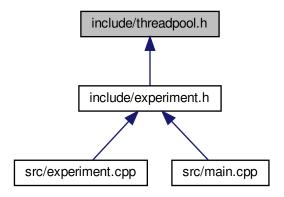
7.27 include/threadpool.h File Reference

```
#include <vector>
#include <queue>
#include <memory>
#include <thread>
#include <mutex>
#include <condition_variable>
#include <future>
#include <future>
#include <stdexcept>
```

Include dependency graph for threadpool.h:



This graph shows which files directly or indirectly include this file:



Classes

· class ThreadPool

7.28 threadpool.h

```
00001
00029 #ifndef __THREADPOOL_H
00030 #define __THREADPOOL_H
00031
00032 #include <vector>
00033 #include <queue>
00034 #include <memory>
00035 #include <thread>
00036 #include <mutex>
00037 #include <condition_variable>
00038 #include <future>
00039 #include <functional>
00040 #include <stdexcept>
00041
00042 class ThreadPool {
00043 public:
00044
          ThreadPool(size_t);
            template<class F, class... Args>
auto enqueue(F&& f, Args&&... args)
   -> std::future<typename std::result_of<F(Args...)>::type>;
00045
00046
00047
00048
            ~ThreadPool();
00049
00050
            void stopAndJoinAll();
00051 private:
           // need to keep track of threads so we can join them
00052
            std::vector< std::thread > workers;
00053
00054
            // the task queue
00055
            std::queue< std::function<void()> > tasks;
00056
00057
            // synchronization
            std::mutex queue_mutex;
std::condition_variable condition;
00058
00059
00060
            bool stop;
00061 };
00062
00063 // the constructor just launches some amount of workers 00064 inline ThreadPool::ThreadPool(size_t threads)
00065
            : stop(false)
00066 {
00067
            for(size_t i = 0;i<threads;++i)</pre>
```

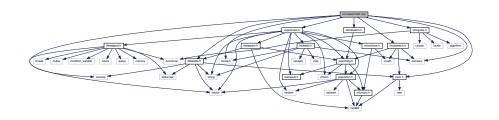
```
00068
              workers.emplace_back(
00069
                  [this]
00070
00071
                       for(;;)
00072
00073
                           std::function<void()> task;
00074
00075
00076
                               std::unique_lock<std::mutex> lock(this->queue_mutex);
00077
                               this->condition.wait(lock,
00078
                                   [this]{ return this->stop || !this->tasks.empty(); });
00079
                               if(this->stop && this->tasks.empty())
00080
                                   return;
00081
                               task = std::move(this->tasks.front());
00082
                               this->tasks.pop();
00083
00084
00085
                           task();
00086
00087
                  }
00088
00089 }
00090
00091 // add new work item to the pool
00092 template<class F, class... Args>
00093 auto ThreadPool::enqueue(F&& f, Args&&... args)
00094
           -> std::future<typename std::result_of<F(Args...)>::type>
00095 {
00096
          using return_type = typename std::result_of<F(Args...)>::type;
00097
00098
          auto task = std::make_shared< std::packaged_task<return_type()> >(
00099
                  std::bind(std::forward<F>(f), std::forward<Args>(args)...)
00100
00101
00102
          std::future<return_type> res = task->get_future();
00103
00104
              std::unique lock<std::mutex> lock(queue mutex);
00105
00106
              // don't allow enqueueing after stopping the pool
00107
00108
                  throw std::runtime_error("enqueue on stopped ThreadPool");
00109
              tasks.emplace([task](){ (*task)(); });
00110
00111
00112
          condition.notify_one();
00113
00114 }
00115
00116 // the destructor joins all threads \,
00117 inline ThreadPool::~ThreadPool()
00118 {
00119
          stopAndJoinAll();
00120 }
00121
00122 inline void ThreadPool::stopAndJoinAll()
00123 {
00125
              std::unique_lock<std::mutex> lock(queue_mutex);
00126
00127
00128
00129
          condition.notify_all();
00130
          for(std::thread &worker: workers)
00131
             worker.join();
00132 }
00133
00134 #endif
00135
00136 // ========
00137 // End of threadpool.h
00138 // ======
```

7.29 src/experiment.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include "experiment.h"
#include "datatable.h"
```

```
#include "blindsearch.h"
#include "localsearch.h"
#include "stringutils.h"
#include "mem.h"
```

Include dependency graph for experiment.cpp:



Macros

- #define INI_TEST_SECTION "test"
- #define INI_FUNC_RANGE_SECTION "function_range"
- #define INI_TEST_POPULATION "population"
- #define INI TEST DIMENSIONS "dimensions"
- #define INI_TEST_ITERATIONS "iterations"
- #define INI_TEST_NUMTHREADS "num_threads"
- #define INI_TEST_ALPHA "alpha"
- #define INI_TEST_ALGORITHM "algorithm"
- #define INI_TEST_RESULTSFILE "results_file"
- #define INI_TEST_EXECTIMESFILE "exec_times_file"

7.29.1 Macro Definition Documentation

7.29.1.1 INI_FUNC_RANGE_SECTION

```
#define INI_FUNC_RANGE_SECTION "function_range"
```

Definition at line 25 of file experiment.cpp.

7.29.1.2 INI_TEST_ALGORITHM

```
#define INI_TEST_ALGORITHM "algorithm"
```

Definition at line 31 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::init().

```
7.29.1.3 INI_TEST_ALPHA
#define INI_TEST_ALPHA "alpha"
Definition at line 30 of file experiment.cpp.
Referenced by mfunc::Experiment< T >::init().
7.29.1.4 INI_TEST_DIMENSIONS
#define INI_TEST_DIMENSIONS "dimensions"
Definition at line 27 of file experiment.cpp.
Referenced by mfunc::Experiment< T >::init().
7.29.1.5 INI_TEST_EXECTIMESFILE
#define INI_TEST_EXECTIMESFILE "exec_times_file"
Definition at line 33 of file experiment.cpp.
Referenced by mfunc::Experiment< T >::init().
7.29.1.6 INI_TEST_ITERATIONS
#define INI_TEST_ITERATIONS "iterations"
Definition at line 28 of file experiment.cpp.
Referenced by mfunc::Experiment< T >::init().
7.29.1.7 INI_TEST_NUMTHREADS
#define INI_TEST_NUMTHREADS "num_threads"
Definition at line 29 of file experiment.cpp.
```

Referenced by mfunc::Experiment< T >::init().

7.29.1.8 INI_TEST_POPULATION

```
#define INI_TEST_POPULATION "population"
```

Definition at line 26 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::init().

7.29.1.9 INI_TEST_RESULTSFILE

```
#define INI_TEST_RESULTSFILE "results_file"
```

Definition at line 32 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::init().

7.29.1.10 INI_TEST_SECTION

```
#define INI_TEST_SECTION "test"
```

Definition at line 24 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::init().

7.30 experiment.cpp

```
00013 #include <iostream>
00014 #include <fstream>
00015 #include <iomanip>
00015 #Include "experiment.h"
00017 #include "datatable.h"
00018 #include "blindsearch.h"
00019 #include "localsearch.h"
00020 #include "stringutils.h"
00021 #include "mem.h"
00022
00023 // Ini file string sections and keys
00024 #define INI_TEST_SECTION "test"
00025 #define INI_FUNC_RANGE_SECTION "function_range"
00026 #define INI_TEST_POPULATION "population" 00027 #define INI_TEST_DIMENSIONS "dimensions"
00028 #define INI_TEST_ITERATIONS "iterations"
00029 #define INI_TEST_NUMTHREADS "num_threads"
00030 #define INI_TEST_ALPHA "alpha"
00031 #define INI_TEST_ALGORITHM "algorithm"
00032 #define INI_TEST_RESULTSFILE "results_file"
00033 #define INI_TEST_EXECTIMESFILE "exec_times_file"
00034
00035 using namespace std;
00036 using namespace std::chrono;
00037 using namespace mfunc;
00038
00042 template<class T>
00043 Experiment<T>::Experiment()
            : vBounds(nullptr), tPool(nullptr), resultsFile(""), execTimesFile(""), iterations(0)
00044
00045 {
00046 }
00047
```

7.30 experiment.cpp 131

```
00052 template<class T>
00053 Experiment<T>::~Experiment()
00054 {
00055
         releaseThreadPool();
00056
        releasePopulationPool();
00057
        releaseVBounds();
00059
00068 template<class T>
00069 bool Experiment<T>::init(const char* paramFile)
00070 {
00071
00072
        {
00073
             // Open and parse parameters file
00074
             if (!iniParams.openFile(paramFile))
00075
                cerr << "Experiment init failed: Unable to open param file: " << paramFile << endl;</pre>
00076
00077
                return false;
00079
00080
             // Extract test parameters from ini file
00081
             long numberSol = iniParams.getEntryAs<long>(INI_TEST_SECTION,
     INI_TEST_POPULATION);
00082
            long numberDim = iniParams.getEntryAs<long>(INI TEST SECTION,
     INI_TEST_DIMENSIONS);
            long numberIter = iniParams.getEntryAs<long>(INI_TEST_SECTION,
     INI_TEST_ITERATIONS);
00084
            long numberThreads = iniParams.getEntryAs<long>(
     INI_TEST_SECTION, INI_TEST_NUMTHREADS);
     alpha = iniParams.getEntryAs<T>(INI_TEST_SECTION, INI_TEST_ALPHA);
00085
00086
            unsigned int selectedAlg = iniParams.getEntryAs<unsigned int>(
     INI_TEST_SECTION, INI_TEST_ALGORITHM);
00087
            resultsFile = iniParams.getEntry(INI_TEST_SECTION,
     00088
     INI_TEST_EXECTIMESFILE);
00089
00090
            // Verify test parameters
00091
             if (numberSol <= 0)</pre>
00092
             {
                00093
00094
    paramFile << endl;</pre>
00095
               return false;
00096
00097
             else if (numberDim <= 0)</pre>
00098
                cerr << "Experiment init failed: Param file [test]->"
00099
                   << INI_TEST_DIMENSIONS << " entry missing or out of bounds: " <<
00100
    paramFile << endl;</pre>
00101
                return false;
00102
00103
             else if (numberIter <= 0)</pre>
00104
00105
                cerr << "Experiment init failed: Param file [test]->"
                    << INI_TEST_ITERATIONS << " entry missing or out of bounds: " <<
00106
return false;
00108
00109
            else if (numberThreads <= 0)</pre>
00110
00111
                cerr << "Experiment init failed: Param file [test]->"
                    << INI_TEST_NUMTHREADS << " entry missing or out of bounds: " <<
00112
     paramFile << endl;</pre>
00113
                return false;
00114
            else if (alpha == 0)
00115
00116
            {
00117
                cerr << "Experiment init failed: Param file [test]->"
                    << INI_TEST_ALPHA << " is missing or is equal to zero: " << paramFile << endl
00118
00119
                return false;
00120
            }
             else if (selectedAlg >= static_cast<unsigned int>(
00121
     enums::Algorithm::Count))
00122
       {
                00123
00124
      << endl:
00125
                return false;
00126
00127
00128
             // Cast iterations and test algorithm to correct types
00129
             iterations = (size_t)numberIter;
00130
             testAlg = static_cast<enums::Algorithm>(selectedAlg);
00131
```

```
// Print test parameters to console
                                    cout << "Population size: " << numberSol << endl;</pre>
00133
                                    cout << "Dimensions: " << numberDim << endl;</pre>
00134
                                    cout << "Iterations: " << iterations << endl;</pre>
00135
                                    cout << "Alpha value: " << alpha << endl;
cout << "Algorithm: " << enums::AlgorithmNames::get(testAlg) << endl;</pre>
00136
00137
00138
00139
                                    // Allocate memory for all population objects. We need one for each thread to prevent conflicts.
00140
                                    if (!allocatePopulationPool((size_t)numberThreads, (size_t)numberSol, (size_t)numberDim))
00141
                                    {
                                              cerr << "Experiment init failed: Unable to allocate populations." << endl;</pre>
00142
00143
                                              return false;
00144
                                   }
00145
00146
                                    // Allocate memory for function vector bounds
00147
                                    if (!allocateVBounds())
00148
00149
                                              cerr << "Experiment init failed: Unable to allocate vector bounds array." << endl;
00150
                                              return false;
00151
                                   }
00152
                                    // Fill function bounds array with data parsed from iniParams
00153
00154
                                    if (!parseFuncBounds())
00155
                                    {
00156
                                              cerr << "Experiment init failed: Unable to parse vector bounds array." << endl;
00157
                                             return false;
00158
00159
                                   // Allocate thread pool
00160
00161
                                    if (!allocateThreadPool((size_t)numberThreads))
00162
                                    {
00163
                                              cerr << "Experiment init failed: Unable to allocate thread pool." << endl;</pre>
00164
                                              return false;
00165
00166
                                    cout << "Started " << numberThreads << " worker threads ..." << endl;</pre>
00167
00168
                                    // Ready to run an experiment
00169
00170
                                    return true;
00171
00172
                         catch (const std::exception& ex)
00173
00174
                                    cerr << "Exception occurred while initializing experiment: " << ex.what() << endl;</pre>
00175
                                    return false;
00176
00177
                         catch (...)
00178
                                    cerr << "Unknown Exception occurred while initializing experiment." << endl;</pre>
00179
00180
                                    return false:
00181
                         }
00182 }
00183
00190 template<class T>
00191 int Experiment<T>::testAllFunc()
00192 {
00193
                          if (populationsPool.size() == 0) return 1;
00194
00195
                          // Construct results and execution times tables
00196
                         mdata::DataTable<T> resultsTable(iterations, (size_t)
              NUM FUNCTIONS);
00197
                         mdata::DataTable<T> execTimesTable(iterations, (size t)NUM FUNCTIONS);
00198
00199
                          // Prepare thread futures vector, used to ensure all async tasks complete
00200
                          // succesfully
00201
                         std::vector<std::future<int>> testFutures;
00202
00203
                          // Start recording total execution time
00204
                         high_resolution_clock::time_point t_start = high_resolution_clock::now();
00205
00206
                               For each of the NUM_FUNCTIONS functions, prepare a TestParameters
00207
                          \ensuremath{//} struct and queue an asynchronous test that will be picked up and
00208
                          // executed by one of the threads in the thread pool.
00209
                         for (unsigned int i = 0; i < NUM_FUNCTIONS; i++)</pre>
00210
00211
                                    // Update column labels for results and exec times tables
00212
                                    resultsTable.setColLabel((size_t)i, FunctionDesc::get(i + 1));
00213
                                    execTimesTable.setColLabel((size_t)i, FunctionDesc::get(i + 1));
00214
00215
                                    // Queue up a new function test for each iteration % \left( 1\right) =\left( 1\right) \left( 1\right) \left(
00216
                                    for (size t iter = 0; iter < iterations; iter++)</pre>
00217
00218
                                              mdata::TestParameters<T> curParam;
00219
                                              curParam.funcId = i + 1;
00220
                                              curParam.alpha = alpha;
                                              curParam.alg = testAlg;
00221
                                             curParam.resultsTable = &resultsTable;
curParam.execTimesTable = &execTimesTable;
00222
00223
```

7.30 experiment.cpp 133

```
curParam.resultsCol = i;
00225
                   curParam.execTimesCol = i;
00226
                   curParam.resultsRow = iter;
00227
                   curParam.execTimesRow = iter;
00228
00229
                   // Add function test to asvnc queue
00230
                   testFutures.emplace_back(
00231
                       tPool->enqueue(&Experiment<T>::testFuncThreaded, this
      , curParam)
00232
                   );
00233
              }
00234
          }
00235
00236
           // Get the total number of async tasks queued
00237
          const double totalFutures = static_cast<double>(testFutures.size());
00238
          int tensPercentile = -1;
00239
          std::chrono::microseconds waitTime(100);
00240
00241
           // Loop until all async tasks are completed and the thread futures
00242
          // array is empty
00243
          while (testFutures.size() > 0)
00244
00245
               // Sleep a little bit since the async thread tasks are higher priority
00246
               std::this_thread::sleep_for(waitTime);
00247
00248
               // Get iterator to first thread future
00249
               auto it = testFutures.begin();
00250
00251
               // Loop through all thread futures
00252
               while (it != testFutures.end())
00253
               {
00254
                   if (!it->valid())
00255
00256
                        // An error occured with one of the threads
00257
                        cerr << "Error: Thread future invalid.";</pre>
00258
                       tPool->stopAndJoinAll();
00259
                       return 1;
00260
00261
00262
                   // Get the status of the current thread future (async task)
00263
                   std::future_status status = it->wait_for(waitTime);
00264
                   if (status == std::future_status::ready)
00265
00266
                        // Task has completed, get return value
                       int errCode = it->get();
00267
00268
                        if (errCode)
00269
00270
                            // An error occurred while running the task.
00271
                            // Bail out of function
00272
                            tPool->stopAndJoinAll();
00273
                            return errCode;
00274
00275
00276
                        // Remove processed task future from vector
00277
                       it = testFutures.erase(it);
00278
00279
                        // Calculate the percent completed of all tasks, rounded to the nearest 10%
                       int curPercentile = static_cast<int>(((totalFutures - testFutures.size()) / totalFutures) *
00280
       10);
00281
                       if (curPercentile > tensPercentile)
00282
                       {
                           // Print latest percent value to the console
tensPercentile = curPercentile;
00283
00284
00285
                            cout << "~" << (tensPercentile * 10) << "% " << flush;
00286
00287
00288
                   else
00289
                   {
00290
                        // Async task has not yet completed, advance to the next one
00291
                       it++;
00292
00293
               }
00294
          }
00295
00296
           // Record total execution time and print it to the console
          high_resolution_clock::time_point t_end = high_resolution_clock::now();
00297
00298
           long double totalExecTime = static_cast<long double>(duration_cast<nanoseconds>(t_end - t_start).count(
      )) / 100000000.0L;
00299
          cout << endl << "Test finished. Total time: " << std::setprecision(7) << totalExecTime << " seconds." <</pre>
00300
      < endl;
00301
00302
           if (!resultsFile.empty())
00303
               // Export results table to a *.csv file
cout << "Exporting results to: " << resultsFile << endl;
resultsTable.exportCSV(resultsFile.c_str());</pre>
00304
00305
00306
```

```
00307
          }
00308
00309
          if (!execTimesFile.empty())
00310
              // Export exec times table to a *.csv file cout << "Exporting execution times to: " << execTimesFile << endl;
00311
00312
              execTimesTable.exportCSV(execTimesFile.c_str());
00313
00314
          }
00315
00316
          cout << flush;
00317
00318
          return 0:
00319 }
00320
00328 template<class T>
00329 int Experiment<T>::testFuncThreaded(
      mdata::TestParameters<T> tParams)
00330 {
00331
          mdata::SearchAlgorithm<T>* alg;
00332
00333
          // Construct a search algorithm object for the selected alg
00334
          switch (tParams.alg)
00335
              case enums::Algorithm::BlindSearch:
00336
00337
                  alg = new mdata::BlindSearch<T>();
00338
                  break;
00339
              case enums::Algorithm::LocalSearch:
00340
                 alg = new mdata::LocalSearch<T>();
00341
                  break;
00342
              default:
00343
                 cerr << "Invalid algorithm selected." << endl;
00344
                  return 1;
00345
00346
00347
          // Retrieve the function bounds
          const RandomBounds<T>& funcBounds = vBounds[tParams.funcId - 1];
00348
00349
00350
          // Retrieve the next available population object from the population pool
00351
          mdata::Population<T>* pop = popPoolRemove();
00352
00353
          \ensuremath{//} Run the search algorithm one and record the results
          auto tResult = alg->run(Functions<T>::get(tParams.funcId), funcBounds.
00354
     min, funcBounds.max, pop, tParams.alpha);
00355
00356
          // Place the population object back into the pool to be reused by anther thread
00357
          popPoolAdd(pop);
00358
00359
          if (tResult.err)
00360
          {
00361
              cerr << "Error while testing function " << tParams.funcId << endl;</pre>
00362
              return tResult.err;
00363
00364
00365
          // Update results table and execution times table with algorithm results
00366
          tParams.resultsTable->setEntry(tParams.resultsRow, tParams.
      resultsCol, tResult.fitness);
00367
         tParams.execTimesTable->setEntry(tParams.execTimesRow, tParams.
     execTimesCol, tResult.execTime);
00368
00369
          delete alg;
00370
          return 0;
00371 }
00372
00380 template<class T>
00381 mdata::Population<T>* Experiment<T>::popPoolRemove()
00382 {
00383
          mdata::Population<T>* retPop = nullptr;
00384
          std::chrono::microseconds waitTime(10);
00385
00386
          while (true)
00387
          {
00388
00389
                  std::lock_guard<std::mutex> lk(popPoolMutex);
00390
                  if (populationsPool.size() > 0)
00391
                   {
00392
                       retPop = populationsPool.back();
00393
                      populationsPool.pop_back();
00394
                   }
00395
              }
00396
              if (retPop != nullptr)
00397
00398
                  return retPop;
00399
00400
                  std::this_thread::sleep_for(waitTime);
00401
          }
00402 }
00403
```

7.30 experiment.cpp 135

```
00412 template<class T>
00413 void Experiment<T>::popPoolAdd(mdata::Population<T>* popPtr)
00414 {
00415
          if (popPtr == nullptr) return;
00416
00417
          std::lock guard<std::mutex> lk(popPoolMutex);
00418
00419
          populationsPool.push_back(popPtr);
00420 }
00421
00428 template<class T>
00429 bool Experiment<T>::parseFuncBounds()
00430 {
00431
          if (vBounds == nullptr) return false;
00432
          const string delim = ",";
const string section = "function_range";
00433
00434
00435
          string s_min;
00436
          string s_max;
00437
00438
           // Extract the bounds for each function
00439
          for (unsigned int i = 1; i <= NUM_FUNCTIONS; i++)</pre>
00440
00441
               // Get bounds entry from ini file for current function
00442
               string entry = iniParams.getEntry(section, to_string(i));
               if (entry.empty())
00444
00445
                   cerr << "Error parsing bounds for function: " << i << endl;</pre>
00446
                   return false;
00447
              }
00448
00449
               // Find index of ',' delimeter in entry string
00450
               auto delimPos = entry.find(delim);
00451
               if (delimPos == string::npos || delimPos >= entry.length() - 1)
00452
               {
                   cerr << "Error parsing bounds for function: " << i << endl;</pre>
00453
00454
                   return false;
00455
00456
00457
               // Split string and extract min/max strings
               s_min = entry.substr((size_t)0, delimPos);
s_max = entry.substr(delimPos + 1, entry.length());
00458
00459
00460
               util::s trim(s min):
00461
               util::s_trim(s_max);
00462
00463
               \ensuremath{//} Attempt to parse min and \max strings into double values
00464
               try
00465
               {
                   RandomBounds<T>& b = vBounds[i - 1];
00466
                   b.min = atof(s_min.c_str());
00467
00468
                   b.max = atof(s_max.c_str());
00469
00470
               catch(const std::exception& e)
00471
                   cerr << "Error parsing bounds for function: " << i << endl;</pre>
00472
00473
                   std::cerr << e.what() << '\n';
00474
                   return false;
00475
00476
00477
00478
          return true;
00479 }
00480
00488 template<class T>
00489 bool Experiment<T>::allocatePopulationPool(size_t count, size_t
      popSize, size_t dimensions)
00490 {
00491
          releasePopulationPool();
00492
00493
          std::lock_guard<std::mutex> lk(popPoolMutex);
00494
00495
00496
00497
               for (int i = 0; i < count; i++)</pre>
00498
00499
                   auto newPop = new(std::nothrow) mdata::Population<T>(popSize, dimensions);
00500
                   if (newPop == nullptr)
00501
                       std::cerr << "Error allocating populations." << '\n';
00502
00503
                       return false;
00504
00505
00506
                   populationsPool.push_back(newPop);
00507
               }
00508
00509
               return true;
00510
          }
```

```
catch(const std::exception& e)
00512
00513
              std::cerr << e.what() << '\n';
00514
              return false;
00515
         }
00516 }
00517
00521 template<class T>
00522 void Experiment<T>::releasePopulationPool()
00523 {
00524
          std::lock_quard<std::mutex> lk(popPoolMutex);
00525
00526
          if (populationsPool.size() == 0) return;
00527
00528
          for (int i = 0; i < populationsPool.size(); i++)</pre>
00529
              if (populationsPool[i] != nullptr)
00530
00531
              {
00532
                  delete populationsPool[i];
00533
                 populationsPool[i] = nullptr;
00534
00535
         }
00536
         populationsPool.clear();
00537
00538 }
00539
00547 template<class T>
00548 bool Experiment<T>::allocateVBounds()
00549 {
          vBounds = util::allocArray<RandomBounds<T>>(NUM_FUNCTIONS);
00550
00551
         return vBounds != nullptr;
00552 }
00553
00557 template<class T>
00558 void Experiment<T>::releaseVBounds()
00559 {
00560
          if (vBounds == nullptr) return;
00562
         util::releaseArray<RandomBounds<T>>(vBounds);
00563 }
00564
00573 template<class T>
00574 bool Experiment<T>::allocateThreadPool(size_t numThreads)
00575 {
00576
         releaseThreadPool();
00577
00578
         tPool = new(std::nothrow) ThreadPool(numThreads);
00579
         return tPool != nullptr;
00580 }
00581
00582 template<class T>
00583 void Experiment<T>::releaseThreadPool()
00584 {
00585
          if (tPool == nullptr) return;
00586
00587
         delete tPool;
         tPool = nullptr;
00589 }
00590
00591 // Explicit template specializations due to separate implementations in this CPP file
00592 template class mfunc::Experiment<float>;
00593 template class mfunc::Experiment<double>;
00594 template class mfunc::Experiment<long double>;
00596 // ==========
00597 // End of experiment.cpp
```

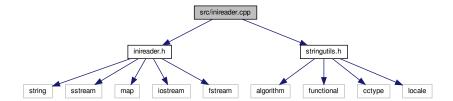
7.31 src/inireader.cpp File Reference

Implementation file for the IniReader class, which can open and parse simple *.ini files.

```
#include "inireader.h"
#include "stringutils.h"
```

7.32 inireader.cpp 137

Include dependency graph for inireader.cpp:



7.31.1 Detailed Description

Implementation file for the IniReader class, which can open and parse simple *.ini files.

Author

```
Andrew Dunn (Andrew. Dunn@cwu.edu)
```

Version

0.1

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file inireader.cpp.

7.32 inireader.cpp

```
00013 #include "inireader.h"
00014 #include "stringutils.h"
00015
00016 using namespace util;
00017
00021 IniReader::IniReader() : file(""), iniMap()
00022 {
00023 }
00024
00028 IniReader::~IniReader()
00029 {
00030
           iniMap.clear();
00031 }
00032
00040 bool IniReader::openFile(std::string filePath)
00041 {
00042
           file = filePath;
00043
           if (!parseFile())
00044
                return false;
```

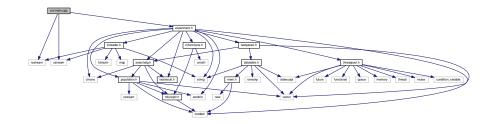
```
00045
00046
          return true;
00047 }
00048
00055 bool IniReader::sectionExists(std::string section)
00056 {
          return iniMap.find(section) != iniMap.end();
00058 }
00059
00067 bool IniReader::entryExists(std::string section, std::string entry)
00068 {
00069
          auto it = iniMap.find(section);
00070
          if (it == iniMap.end()) return false;
00071
00072
          return it->second.find(entry) != it->second.end();
00073 }
00074
00084 std::string IniReader::getEntry(std::string section, std::string entry)
00085 {
00086
          if (!entryExists(section, entry)) return std::string();
00087
00088
          return iniMap[section][entry];
00089 }
00090
00097 bool IniReader::parseFile()
00098 {
00099
          iniMap.clear();
00100
00101
          using namespace std;
00102
          ifstream inputF(file, ifstream::in);
00103
00104
          if (!inputF.good()) return false;
00105
00106
          string curSection;
00107
          string line;
00108
00109
          while (getline(inputF, line))
00110
          {
00111
              // Trim whitespace on both ends of the line
00112
              s_trim(line);
00113
              // Ignore empty lines and comments
00114
              if (line.empty() || line.front() == '#')
00115
00116
              {
00117
                  continue;
00118
00119
              else if (line.front() == '[' && line.back() == ']')
00120
              {
00121
                  // Line is a section definition
00122
                  // Erase brackets and trim to get section name
00123
                  line.erase(0, 1);
00124
                  line.erase(line.length() - 1, 1);
                  s_trim(line);
00125
00126
                  curSection = line;
00127
00128
              else if (!curSection.emptv())
00129
00130
                  // Line is an entry, parse the key and value
00131
                  parseEntry(curSection, line);
00132
              }
00133
          }
00134
00135
          // Close input file
00136
          inputF.close();
00137
          return true;
00138 }
00139
00144 void IniReader::parseEntry(const std::string& sectionName, const std::string& entry)
00145 {
00146
          using namespace std;
00147
00148
          // Split string around equals sign character
00149
          const string delim = "=";
00150
          string entryName;
00151
          string entryValue;
00152
00153
          // Find index of '='
00154
          auto delimPos = entry.find(delim);
00155
00156
          if (delimPos == string::npos || delimPos >= entry.length() - 1)
              return; // '=' is missing, or is last char in string
00157
00158
00159
          // Extract entry name/key and value
00160
          entryName = entry.substr((size_t)0, delimPos);
00161
          entryValue = entry.substr(delimPos + 1, entry.length());
00162
00163
          // Remove leading and trailing whitespace
```

```
00164
         s_trim(entryName);
00165
         s_trim(entryValue);
00166
         // We cannot have entries with empty keys
00167
00168
         if (entryName.empty()) return;
00169
00170
         // Add entry to cache
00171
         iniMap[sectionName][entryName] = entryValue;
00172 }
00173
00174 // =========
00175 // End of inireader.cpp
00176 //
```

7.33 src/main.cpp File Reference

Program entry point. Creates and runs CS471 project 2 experiment.

```
#include <iostream>
#include <sstream>
#include "experiment.h"
Include dependency graph for main.cpp:
```



Functions

template < class T >
 int runExp (const char *paramFile)

Runs the experiment using the given data type and parameter file. Currently supports three different data types: float, double, and long double.

• int main (int argc, char **argv)

7.33.1 Detailed Description

Program entry point. Creates and runs CS471 project 2 experiment.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.2

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file main.cpp.

7.33.2 Function Documentation

7.33.2.1 main()

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

Definition at line 46 of file main.cpp.

```
00047 {
00048
         // Make sure we have enough command line args
         if (argc <= 1)</pre>
00050
            cout << "Error: Missing command line parameter." << endl;
cout << "Proper usage: " << argv[0] << " [param file]" << endl;</pre>
00051
00052
            return EXIT_FAILURE;
00053
00054
00055
00056
         // Default data type is double
00057
         int dataType = 1;
00058
00059
         // User specified a data type, retrieve the value
00060
         if (argc > 2)
00061
00062
            std::stringstream ss(argv[2]);
00063
            ss >> dataType;
00064
             if (!ss) dataType = 1;
00065
         }
00066
00067
         // Verify specified data type switch
00068
         if (dataType < 0 || dataType > 2)
00069
00070
             cout << dataType << " is not a valid data type index. Value must be between 0 and 2." << endl;
00071
            dataType = 1;
00072
00073
00074
         // Run experiment with correct data type and return success code
00075
         switch (dataType)
00076
00077
            case 0:
00078
               return runExp<float>(argv[1]);
00079
            case 1:
08000
               return runExp<double>(argv[1]);
00081
            case 2:
00082
               return runExp<long double>(argv[1]);
            default:
00083
               return EXIT_FAILURE;
00084
00085
         }
00086 }
```

7.33.2.2 runExp()

Runs the experiment using the given data type and parameter file. Currently supports three different data types: float, double, and long double.

7.34 main.cpp 141

Template Parameters



Parameters

```
paramFile
```

Returns

int

Definition at line 29 of file main.cpp.

References mfunc::Experiment< T >::init(), and mfunc::Experiment< T >::testAllFunc().

```
00030 {
00031
            // Create an instance of the project 1 experiment class
00032
           mfunc::Experiment<T> ex;
00033
00034
           // Print size of selected data type in bits
           cout << "Float size: " << (sizeof(T) * 8) << "-bits" << endl;
cout << "Input parameters file: " << paramFile << endl;
cout << "Initializing experiment ..." << endl;</pre>
00035
00036
00037
00038
00039
            // If experiment initialization fails, return failure
           if (!ex.init(paramFile))
00041
               return EXIT_FAILURE;
00042
00043
               return ex.testAllFunc();
00044 }
```

7.34 main.cpp

```
00001
00013 #include <iostream>
00014 #include <sstream>
00015 #include "experiment.h"
00016
00017 using namespace std;
00018
00028 template<class T>
00029 int runExp(const char* paramFile)
00030 {
00031
            // Create an instance of the project 1 experiment class
00032
           mfunc::Experiment<T> ex;
00033
00034
           // Print size of selected data type in bits
           cout << "Float size: " << (sizeof(T) * 8) << "-bits" << endl;
cout << "Input parameters file: " << paramFile << endl;
cout << "Initializing experiment ..." << endl;</pre>
00035
00036
00037
00038
           // If experiment initialization fails, return failure if (!ex.init(paramFile)) \,
00039
00040
              return EXIT_FAILURE;
00041
00042
           else
00043
              return ex.testAllFunc();
00044 }
00045
00046 int main(int argc, char** argv)
00047 {
00048
            // Make sure we have enough command line args
00049
           if (argc <= 1)
00050
               cout << "Error: Missing command line parameter." << endl;
cout << "Proper usage: " << argv[0] << " [param file]" << endl;
return EXIT_FAILURE;
00051
00052
00053
00054
00055
```

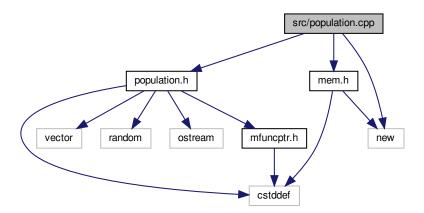
```
// Default data type is double
00057
        int dataType = 1;
00058
        // User specified a data type, retrieve the value
00059
00060
        if (argc > 2)
00061
00062
           std::stringstream ss(argv[2]);
00063
           ss >> dataType;
00064
           if (!ss) dataType = 1;
00065
00066
00067
        // Verify specified data type switch
00068
        if (dataType < 0 || dataType > 2)
00069
00070
            cout << dataType << " is not a valid data type index. Value must be between 0 and 2." << endl;
00071
           dataType = 1;
00072
00073
        // Run experiment with correct data type and return success code
00075
        switch (dataType)
00076
00077
           case 0:
00078
              return runExp<float>(argv[1]);
00079
           case 1:
08000
             return runExp<double>(argv[1]);
           case 2:
00082
              return runExp<long double>(argv[1]);
           default:
00083
00084
            return EXIT_FAILURE;
00085
        }
00086 }
00087
00088 // ===========
00089 // End of main.cpp \,
00090 // =====
```

7.35 src/population.cpp File Reference

Implementation file for the Population class. Stores a population and fitness values.

```
#include "population.h"
#include "mem.h"
#include <new>
```

Include dependency graph for population.cpp:



7.35.1 Detailed Description

Implementation file for the Population class. Stores a population and fitness values.

7.36 population.cpp 143

Author

```
Andrew Dunn (Andrew . Dunn@cwu . edu)
```

Version

0.2

Date

2019-04-04

Copyright

Copyright (c) 2019

Definition in file population.cpp.

7.36 population.cpp

```
00001
00012 #include "population.h"
00013 #include "mem.h"
00014 #include <new>
00015
00016 using namespace mdata;
00017 using namespace util;
00018
00026 template <class T>
00027 Population<T>::Population (size_t pSize, size_t dimensions) : popMatrix(nullptr),
popSize(pSize), popDim(dimensions)
00028 {
00029
          if (!allocPopMatrix() || !allocPopFitness())
00030
              throw std::bad_alloc();
00031 }
00032
00038 template <class T>
00039 Population<T>::~Population()
00040 {
00041
          releasePopMatrix();
00042
          releasePopFitness();
00043 }
00044
00052 template <class T>
00053 bool Population<T>::isReady()
00054 {
00055
          return popMatrix != nullptr && popFitness != nullptr;
00056 }
00057
00064 template <class T>
00065 size_t Population<T>::getPopulationSize()
00066 {
00067
          return popSize;
00068 }
00069
00076 template <class T>
00077 size_t Population<T>::getDimensionsSize()
00078 {
00079
          return popDim;
00080 }
00081
00089 template <class T>
00090 T* Population<T>::getPopulationPtr(size_t popIndex)
00091 {
00092
          if (popFitness == nullptr || popIndex >= popSize) return nullptr;
00093
00094
          return popMatrix[popIndex];
00095 }
00096
00107 template <class T>
```

```
00108 bool Population<T>::generate(T minBound, T maxBound)
00109 {
00110
          if (popMatrix == nullptr) return false;
00111
00112
          // Generate a new seed for the mersenne twister engine
          rgen = std::mt19937(rdev());
00113
00114
00115
          // Set up a normal (bell-shaped) distribution for the random number generator with the correct function
       bounds
00116
          std::uniform real distribution<double> dist((double)minBound, (double)maxBound);
00117
          // Generate values for all vectors in popMatrix
00118
00119
          for (size_t s = 0; s < popSize; s++)</pre>
00120
00121
              for (size_t d = 0; d < popDim; d++)</pre>
00122
                  T rand = (T) dist(rgen);
00123
00124
                  popMatrix[s][d] = rand;
00125
00126
          }
00127
00128
          // Reset popFitness values to 0
00129
          initArray<T>(popFitness, popSize, (T)0.0);
00130
00131
          return true;
00132 }
00133
00142 template<class T>
00143 bool Population<T>::setFitness(size_t popIndex, T value)
00144 {
00145
          if (popFitness == nullptr || popIndex >= popSize) return false;
00146
00147
          popFitness[popIndex] = value;
00148
00149
          return true;
00150 }
00151
00162 template<class T>
00163 bool Population<T>::calcFitness(size_t popIndex,
      mfunc::mfuncPtr<T> funcPtr)
00164 {
00165
          if (popFitness == nullptr || popIndex >= popSize) return false;
00166
00167
          popFitness[popIndex] = funcPtr(popMatrix[popIndex], popDim);
00168
00169
          return true;
00170 }
00171
00179 template<class T>
00180 T Population<T>::getFitness(size_t popIndex)
00181 {
00182
          if (popFitness == nullptr || popIndex >= popSize) return 0;
00183
00184
          return popFitness[popIndex];
00185 }
00186
00194 template<class T>
00195 T* Population<T>::getFitnessPtr(size_t popIndex)
00196 {
00197
          if (popFitness == nullptr || popIndex >= popSize) return 0;
00198
00199
          return &popFitness[popIndex];
00200 }
00201
00208 template<class T>
00209 std::vector<T> Population<T>::getAllFitness()
00210 {
00211
          return std::vector<T>(popFitness[0], popFitness[popSize]);
00212 }
00213
00220 template<class T>
00221 T* Population<T>::getBestFitnessPtr()
00222 {
          return &popFitness[getBestFitnessIndex()];
00223
00224 }
00225
00232 template<class T>
00233 size_t Population<T>::getBestFitnessIndex()
00234 {
00235
          size t bestIndex = 0:
00236
00237
          for (size_t i = 1; i < popSize; i++)</pre>
00238
          {
00239
              if (popFitness[i] < popFitness[bestIndex])</pre>
00240
                  bestIndex = i;
00241
          }
00242
```

7.36 population.cpp 145

```
00243
          return bestIndex;
00244 }
00245
00254 template<class T>
00255 void Population<T>::outputPopulation(std::ostream& outStream, const char*
      delim, const char* lineBreak)
00256 {
00257
          if (popMatrix == nullptr) return;
00258
          for (size_t j = 0; j < popSize; j++)</pre>
00259
00260
00261
              for (size_t k = 0; k < popDim; k++)</pre>
00262
              {
00263
                  outStream << popMatrix[j][k];</pre>
00264
                  if (k < popDim - 1)</pre>
                      outStream << delim;</pre>
00265
00266
              }
00267
00268
              outStream << lineBreak;</pre>
00269
         }
00270 }
00271
00280 template<class T>
00281 void Population<T>::outputFitness(std::ostream& outStream, const char* delim,
      const char* lineBreak)
00282 {
00283
          if (popFitness == nullptr) return;
00284
00285
          for (size_t j = 0; j < popSize; j++)</pre>
00286
00287
              outStream << popFitness[j];</pre>
00288
                  if (j < popSize - 1)
00289
                      outStream << delim;
00290
          }
00291
          if (lineBreak != nullptr)
00292
00293
             outStream << lineBreak;
00294 }
00295
00302 template <class T>
00303 bool Population<T>::allocPopMatrix()
00304 {
          if (popSize == 0 || popDim == 0) return false;
00305
00306
00307
          popMatrix = allocMatrix<T>(popSize, popDim);
00308
          initMatrix<T>(popMatrix, popSize, popDim, 0);
00309
          return popMatrix != nullptr;
00310
00311 }
00312
00318 template <class T>
00319 void Population<T>::releasePopMatrix()
00320 {
00321
          releaseMatrix<T>(popMatrix, popSize);
00322 }
00323
00330 template <class T>
00331 bool Population<T>::allocPopFitness()
00332 {
00333
          if (popSize == 0 || popDim == 0) return false;
00334
00335
          popFitness = allocArray<T>(popSize);
00336
          initArray<T>(popFitness, popSize, 0);
00337
00338
          return popFitness != nullptr;
00339 }
00340
00346 template <class T>
00347 void Population<T>::releasePopFitness()
00348 {
00349
          releaseArray<T>(popFitness);
00350 }
00351
00352 // Explicit template specializations due to separate implementations in this CPP file
00353 template class mdata::Population<float>;
00354 template class mdata::Population<double>;
00355 template class mdata::Population<long double>;
00356
00357 // ==========
00358 // End of population.cpp
00359 // =====
```

Index

_USE_MATH_DEFINES	ThreadPool, 80
mfunctions.h, 108	a aklaya Ona
_ackleysOneDesc	ackleysOne
mfunctions.h, 105	mfunc::Functions, 36 ackleysTwo
_ackleysTwoDesc	mfunc::Functions, 37
mfunctions.h, 105	
_alpineDesc	alg mdata::TestParameters, 75
mfunctions.h, 105	
_dejongDesc	Algorithm enums, 9
mfunctions.h, 105	allocArray
_eggHolderDesc	util, 12
mfunctions.h, 105	allocMatrix
_griewangkDesc	util, 13
mfunctions.h, 105	alpha
_levyDesc	mdata::TestParameters, 75
mfunctions.h, 106	alpine
_mastersCosineWaveDesc	mfunc::Functions, 37
mfunctions.h, 106	
_michalewiczDesc	BLIND_SEARCH
mfunctions.h, 106	enums::AlgorithmNames, 18
_pathologicalDesc	
mfunctions.h, 106	calcFitness
_quarticDesc	mdata::Population, 60
mfunctions.h, 106	copyArray
_ranaDesc	util, 13
mfunctions.h, 107	Count
_rastriginDesc	enums, 9
mfunctions.h, 107	DEC_PRECISION
_rosenbrokDesc	localsearch.h, 95
mfunctions.h, 107	DataTable
_schwefelDesc	mdata::DataTable, 22
mfunctions.h, 107	dejong
_sineEnvelopeSineWaveDesc	mfunc::Functions, 38
mfunctions.h, 107	
_stepDesc	eggHolder
mfunctions.h, 108	mfunc::Functions, 38
_stretchedVSineWaveDesc	enqueue
mfunctions.h, 108	ThreadPool, 81
~DataTable	entryExists
mdata::DataTable, 22	util::IniReader, 52
~Experiment	enums, 9
mfunc::Experiment, 27	Algorithm, 9
~IniReader	Count, 9
util::IniReader, 52	enums::AlgorithmNames, 17
~Population	BLIND_SEARCH, 18
mdata::Population, 60	get, 17
~SearchAlgorithm	LOCAL_SEARCH, 18
mdata::SearchAlgorithm, 72	err
\sim ThreadPool	mdata::TestResult. 78

exec	mdata::Population, 67
mfunc::Functions, 39	griewangk
execTime	mfunc::Functions, 41
mdata::TestResult, 79	
execTimesCol	INI_FUNC_RANGE_SECTION
mdata::TestParameters, 76	experiment.cpp, 128
execTimesRow	INI_TEST_ALGORITHM
mdata::TestParameters, 76	experiment.cpp, 128
execTimesTable	INI_TEST_ALPHA
mdata::TestParameters, 76	experiment.cpp, 128
Experiment	INI_TEST_DIMENSIONS
mfunc::Experiment, 27	experiment.cpp, 129
experiment.cpp	INI_TEST_EXECTIMESFILE
INI_FUNC_RANGE_SECTION, 128	experiment.cpp, 129
INI_TEST_ALGORITHM, 128	INI_TEST_ITERATIONS
INI TEST ALPHA, 128	experiment.cpp, 129
INI TEST DIMENSIONS, 129	INI_TEST_NUMTHREADS
INI TEST EXECTIMESFILE, 129	experiment.cpp, 129
INI TEST ITERATIONS, 129	INI_TEST_POPULATION
INI TEST NUMTHREADS, 129	experiment.cpp, 129
INI TEST POPULATION, 129	INI_TEST_RESULTSFILE
INI TEST RESULTSFILE, 130	experiment.cpp, 130
INI TEST SECTION, 130	INI_TEST_SECTION
exportCSV	experiment.cpp, 130
mdata::DataTable, 22	include/blindsearch.h, 83, 85
matabata table, EE	include/datatable.h, 85, 87
fitness	include/experiment.h, 88, 90
mdata::TestResult, 79	include/inireader.h, 91, 93
funcid	include/localsearch.h, 93, 95
mdata::TestParameters, 76	include/mem.h, 97, 99
mataroot aramotoro, ro	include/mfuncptr.h, 100, 102
generate	include/mfunctions.h, 102, 109
mdata::Population, 61	include/population.h, 114, 116
get	include/searchalg.h, 117, 119
enums::AlgorithmNames, 17	include/stringutils.h, 120, 121
mfunc::FunctionDesc, 34	include/testparam.h, 122, 123
mfunc::Functions, 40	include/testresult.h, 124, 125
getAllFitness	include/threadpool.h, 125, 126
mdata::Population, 62	IniReader
getBestFitnessIndex	util::IniReader, 52
mdata::Population, 62	init
getBestFitnessPtr	mfunc::Experiment, 28
mdata::Population, 63	•
getColLabel	initArray
mdata::DataTable, 24	util, 14
getDimensionsSize	initMatrix
mdata::Population, 63	util, 14
getEntry	isReady
mdata::DataTable, 25	mdata::Population, 67
util::IniReader, 53	LOCAL SEARCH
getEntryAs	enums::AlgorithmNames, 18
util::IniReader, 53	_
	levy
getFitness mdata::Population, 65	mfunc::Functions, 41
•	localsearch.h
getFitnessPtr	DEC_PRECISION, 95
mdata::Population, 65	main
getPopulationPtr	main
mdata::Population, 66	main.cpp, 140
getPopulationSize	main.cpp

main, 140	resultsTable, 77
runExp, 140	TestParameters, 75
mastersCosineWave	mdata::TestParameters $<$ T $>$, 74
mfunc::Functions, 42	mdata::TestResult
max	err, 78
mfunc::RandomBounds, 70	execTime, 79
mdata, 10	fitness, 79
mdata::BlindSearch	TestResult, 78
run, 20	mdata::TestResult< T >, 78
mdata::BlindSearch< T >, 19	mfunc, 10
mdata::DataTable	mfuncPtr, 11
∼DataTable, 22	NUM_FUNCTIONS, 11
DataTable, 22	mfunc::Experiment
exportCSV, 22	~Experiment, 27
getColLabel, 24	Experiment, 27
getEntry, 25	init, 28
setColLabel, 25	testAllFunc, 30
setEntry, 26	testFuncThreaded, 32
mdata::DataTable $<$ T $>$, 21	mfunc::Experiment< T >, 26
mdata::LocalSearch	mfunc::FunctionDesc, 33
run, 56	get, 34
mdata::LocalSearch< T >, 55	mfunc::Functions
mdata::Population	ackleysOne, 36
∼Population, 60	ackleysTwo, 37
calcFitness, 60	alpine, 37
generate, 61	dejong, 38
getAllFitness, 62	eggHolder, 38
getBestFitnessIndex, 62	exec, 39
getBestFitnessPtr, 63	get, 40
getDimensionsSize, 63	griewangk, 41
getFitness, 65	levy, 41
getFitnessPtr, 65	mastersCosineWave, 42
getPopulationPtr, 66	michalewicz, 43
getPopulationSize, 67	nthroot, 43
isReady, 67	pathological, 45
outputFitness, 68	quartic, 45
outputPopulation, 68	rana, 46
Population, 59	rastrigin, 47
setFitness, 69	rosenbrok, 47
mdata::Population< T >, 58	schwefel, 48
mdata::SearchAlgorithm	sineEnvelopeSineWave, 48
\sim SearchAlgorithm, 72	step, 49
run, 72	stretchedVSineWave, 50
SearchAlgorithm, 72	w, 50
startTimer, 73	mfunc::Functions $<$ T $>$, 35
stopTimer, 73	mfunc::RandomBounds
timeDiff, 73	max, 70
timer, 74	min, 70
mdata::SearchAlgorithm $<$ T $>$, 71	mfunc::RandomBounds $<$ T $>$, 70
mdata::TestParameters	mfuncPtr
alg, 75	mfunc, 11
alpha, 75	mfunctions.h
execTimesCol, 76	_USE_MATH_DEFINES, 108
execTimesRow, 76	_ackleysOneDesc, 105
execTimesTable, 76	_ackleysTwoDesc, 105
funcld, 76	_alpineDesc, 105
resultsCol, 77	_dejongDesc, 105
resultsRow, 77	_eggHolderDesc, 105

_griewangkDesc, 105 _levyDesc, 106	main.cpp, 140
_mastersCosineWaveDesc, 106	schwefel
_michalewiczDesc, 106	mfunc::Functions, 48
_pathologicalDesc, 106	SearchAlgorithm
_quarticDesc, 106	mdata::SearchAlgorithm, 72
_ ·	sectionExists
_ranaDesc, 107	util::IniReader, 54
_rastriginDesc, 107	setColLabel
_rosenbrokDesc, 107	
_schwefelDesc, 107	mdata::DataTable, 25
_sineEnvelopeSineWaveDesc, 107	setEntry
_stepDesc, 108	mdata::DataTable, 26
_stretchedVSineWaveDesc, 108	setFitness
michalewicz	mdata::Population, 69
mfunc::Functions, 43	sineEnvelopeSineWave
min	mfunc::Functions, 48
mfunc::RandomBounds, 70	src/experiment.cpp, 127, 130
	src/inireader.cpp, 136, 137
NUM_FUNCTIONS	src/main.cpp, 139, 141
mfunc, 11	src/population.cpp, 142, 143
nthroot	startTimer
mfunc::Functions, 43	mdata::SearchAlgorithm, 73
Thano anotions, To	step
openFile	mfunc::Functions, 49
util::IniReader, 54	stopAndJoinAll
	•
outputFitness	ThreadPool, 81
mdata::Population, 68	stopTimer
outputPopulation	mdata::SearchAlgorithm, 73
mdata::Population, 68	stretchedVSineWave
	mfunc::Functions, 50
pathological	
mfunc::Functions, 45	testAllFunc
Population	mfunc::Experiment, 30
mdata::Population, 59	testFuncThreaded
	mfunc::Experiment, 32
quartic	TestParameters
mfunc::Functions, 45	mdata::TestParameters, 75
	TestResult
rana	mdata::TestResult, 78
mfunc::Functions, 46	ThreadPool, 79
rastrigin	~ThreadPool, 80
mfunc::Functions, 47	enqueue, 81
releaseArray	stopAndJoinAll, 81
util, 15	ThreadPool, 80
releaseMatrix	timeDiff
util, 16	
resultsCol	mdata::SearchAlgorithm, 73
	timer
mdata::TestParameters, 77	mdata::SearchAlgorithm, 74
resultsRow	
mdata::TestParameters, 77	util, 11
resultsTable	allocArray, 12
mdata::TestParameters, 77	allocMatrix, 13
rosenbrok	copyArray, 13
mfunc::Functions, 47	initArray, 14
run	initMatrix, 14
mdata::BlindSearch, 20	releaseArray, 15
mdata::LocalSearch, 56	releaseMatrix, 16
mdata::SearchAlgorithm, 72	util::IniReader, 51
runExp	\sim IniReader, 52
•	•

```
entryExists, 52
getEntry, 53
getEntryAs, 53
IniReader, 52
openFile, 54
sectionExists, 54
```