

Flourite Gerardium Rush

Generated by Doxygen 1.9.1

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 Algorithm_Parameters Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	5
3.1.2.1 crossover_rate	5
3.1.2.2 crossover_type_rate	6
3.1.2.3 early_stopping_generations	6
3.1.2.4 generation_step	6
3.1.2.5 mutation_max_step	6
3.1.2.6 mutation_rate	6
3.1.2.7 mutation_type_rate	6
3.1.2.8 population_size	6
3.2 CCircuit Class Reference	7
3.2.1 Detailed Description	7
3.2.2 Constructor & Destructor Documentation	8
3.2.2.1 CCircuit() [1/3]	8
3.2.2.2 CCircuit() [2/3]	8
3.2.2.3 CCircuit() [3/3]	8
3.2.3 Member Function Documentation	8
3.2.3.1 Check_Velocity()	8
3.2.3.2 FillIDs()	9
3.2.3.3 mark_units()	9
3.2.3.4 ResetFlowrates()	9
3.2.3.5 ReturnFlowrates()	10
3.2.3.6 ReturnProfit()	10
3.2.3.7 SetInlet()	11
3.2.3.8 SolveCCircuit()	11
3.2.3.9 SystemFlowrates()	11
3.2.4 Member Data Documentation	12
3.2.4.1 CCircuit_vector	12
3.2.4.2 num_units	12
3.2.4.3 recycle_mineral	12
3.2.4.4 recycle_waste	12
3.2.4.5 system_mineral_input	12
3.2.4.6 system_waste_input	12
3.2.4.7 units	13
3.3 Circuit_Parameters Struct Reference	13

3.3.1 Detailed Description	13
3.3.2 Member Data Documentation	13
3.3.2.1 max_iterations	13
3.3.2.2 tolerance	13
3.4 CUnit Class Reference	14
3.4.1 Detailed Description	15
3.4.2 Constructor & Destructor Documentation	15
3.4.2.1 CUnit() [1/2]	15
3.4.2.2 CUnit() [2/2]	15
3.4.3 Member Function Documentation	15
3.4.3.1 CheckMassBalance()	15
3.4.3.2 ConcentrationCalc()	16
3.4.3.3 OutletCalc()	16
3.4.3.4 PrintCUnit()	16
3.4.3.5 ReactionComp()	16
3.4.3.6 ResidenceTime()	17
3.4.3.7 ReturnOutlets()	17
3.4.3.8 SolveCUnit()	17
3.4.4 Member Data Documentation	18
3.4.4.1 Cgc	18
3.4.4.2 Cgi	18
3.4.4.3 Cgt	18
3.4.4.4 conc_num	18
3.4.4.5 Cwc	18
3.4.4.6 Cwi	18
3.4.4.7 Cwt	18
3.4.4.8 Fgc	19
3.4.4.9 Fgi	19
3.4.4.10 Fgt	19
3.4.4.11 Ftc	19
3.4.4.12 Fti	19
3.4.4.13 Ftt	19
3.4.4.14 Fwc	19
3.4.4.15 Fwi	19
3.4.4.16 Fwt	20
3.4.4.17 ID_num	20
3.4.4.18 ID_num_conc	20
3.4.4.19 ID_num_tail	20
3.4.4.20 mark	20
3.4.4.21 Rg	20
3.4.4.22 Rw	20
3.4.4.23 tails_num	20

3.4.4.24 tau	21
3.5 Individual Struct Reference	21
3.5.1 Detailed Description	21
3.5.2 Member Data Documentation	21
3.5.2.1 fitness_val	21
3.5.2.2 vector	21
4 File Documentation	23
4.1 include/CCircuit.h File Reference	23
4.2 include/CSimulator.h File Reference	23
4.2.1 Function Documentation	23
4.2.1.1 Evaluate_Circuit() [1/2]	24
4.2.1.2 Evaluate_Circuit() [2/2]	24
4.3 include/CUnit.h File Reference	24
4.4 include/Genetic_Algorithm.h File Reference	24
4.4.1 Function Documentation	25
4.4.1.1 AddChildren()	25
4.4.1.2 CalculateFitness()	26
4.4.1.3 Crossover()	26
4.4.1.4 Evaluate_circuit()	27
4.4.1.5 InitializePopulation()	27
4.4.1.6 MakeCDF()	27
4.4.1.7 Mutation()	28
4.4.1.8 NormalizeFitness()	28
4.4.1.9 optimize()	29
4.4.1.10 SelectParents()	29
4.4.1.11 SortParentsByFitness()	30
4.5 include/HyperPSearch.h File Reference	30
4.5.1 Function Documentation	30
4.5.1.1 generateRandomParameters()	30
4.5.1.2 randomSearch()	31
4.6 src/CCircuit.cpp File Reference	31
4.7 src/CMakeLists.txt File Reference	31
4.8 src/CSimulator.cpp File Reference	31
4.8.1 Function Documentation	32
4.8.1.1 Evaluate_Circuit() [1/2]	32
4.8.1.2 Evaluate_Circuit() [2/2]	32
4.8.2 Variable Documentation	32
4.8.2.1 default_circuit_parameters	32
4.9 src/CUnit.cpp File Reference	32
4.9.1 Function Documentation	33
4.9.1.1 isClose()	33

4.10 src/Genetic_Algorithm.cpp File Reference	33
4.10.1 Function Documentation	34
4.10.1.1 AddChildren()	34
4.10.1.2 CalculateFitness()	35
4.10.1.3 Crossover()	35
4.10.1.4 Evaluate_circuit()	35
4.10.1.5 InitializePopulation()	36
4.10.1.6 MakeCDF()	36
4.10.1.7 Mutation()	37
4.10.1.8 NormalizeFitness()	37
4.10.1.9 optimize()	38
4.10.1.10 SelectParents()	38
4.10.1.11 SortParentsByFitness()	38
4.11 src/HyperPSearch.cpp File Reference	39
4.11.1 Function Documentation	39
4.11.1.1 generateRandomParameters()	39
4.11.1.2 randomSearch()	40
4.12 src/main.cpp File Reference	40
4.12.1 Function Documentation	40
4.12.1.1 main()	40

Index	41
--------------	-----------

Chapter 1

Class Index

1.1 Class List

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

Algorithm_Parameters	
Contains parameters used to control the behavior of the genetic algorithm	5
CCircuit	7
Circuit_Parameters	13
CUnit	14
Individual	
Represents an individual in the population	21

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

include/CCircuit.h	23
include/CSimulator.h	23
include/CUnit.h	24
include/Genetic_Algorithm.h	24
include/HyperPSearch.h	30
src/CCircuit.cpp	31
src/CSimulator.cpp	31
src/CUnit.cpp	32
src/Genetic_Algorithm.cpp	33
src/HyperPSearch.cpp	39
src/main.cpp	40

Chapter 3

Class Documentation

3.1 Algorithm_Parameters Struct Reference

Contains parameters used to control the behavior of the genetic algorithm.

```
#include <Genetic_Algorithm.h>
```

Public Attributes

- int [population_size](#)
- double [mutation_rate](#)
- double [crossover_rate](#)
- int [generation_step](#)
- int [mutation_max_step](#)
- int [early_stopping_generations](#)
- std::vector< double > [mutation_type_rate](#)
- std::vector< double > [crossover_type_rate](#)

3.1.1 Detailed Description

Contains parameters used to control the behavior of the genetic algorithm.

3.1.2 Member Data Documentation

3.1.2.1 crossover_rate

```
Algorithm_Parameters::crossover_rate
```

The chance that a crossover will occur during reproduction.

3.1.2.2 crossover_type_rate

`Algorithm_Parameters::crossover_type_rate`

A vector that determines the relative probabilities of different types of crossovers.

3.1.2.3 early_stopping_generations

`Algorithm_Parameters::early_stopping_generations`

The number of generations without improvement after which the algorithm should stop early.

3.1.2.4 generation_step

`Algorithm_Parameters::generation_step`

The maximum number of generations for which the algorithm should run.

3.1.2.5 mutation_max_step

`Algorithm_Parameters::mutation_max_step`

The maximum amount by which an individual's value can change during a mutation.

3.1.2.6 mutation_rate

`Algorithm_Parameters::mutation_rate`

The chance that a mutation will occur during reproduction.

3.1.2.7 mutation_type_rate

`Algorithm_Parameters::mutation_type_rate`

A vector that determines the relative probabilities of different types of mutations.

3.1.2.8 population_size

`Algorithm_Parameters::population_size`

Size of the population in each generation.

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

- [include/Genetic_Algorithm.h](#)

3.2 CCircuit Class Reference

```
#include <CCircuit.h>
```

Public Member Functions

- [CCircuit](#) ()
Constructor of the [CCircuit](#) class that initialises the vector of [CUnit](#) units.
- [CCircuit](#) (int num_units)
Constructor of the [CCircuit](#) class with specified number of units and a circuit vector.
- [CCircuit](#) (int num_units_, const std::vector< int > &CCircuit_vector_)
Constructor of the [CCircuit](#) class with specified number of units and a circuit vector.
- void [FillIDs](#) ()
Fills the IDs of the [CCircuit](#) units and initializes related data structures. This function fills the IDs of the [CCircuit](#) units and performs the following steps: Resizes the units, recycle_mineral, and recycle_waste vectors based on the number of units. Initializes the recycle_mineral and recycle_waste vectors with zeros. Assigns the appropriate IDs to each unit based on the CCircuit_vector. Sets the IDs for the concentrate and tailing outlets of the system.
- void [SystemFlowrates](#) (const double Fgi_, const double Fwi_)
Sets the system flow rates for mineral and waste inputs.
- void [SolveCCircuit](#) ()
Solves the [CCircuit](#) by iterating through its units.
- void [ReturnFlowrates](#) (std::vector< double > &Flowrates_g, std::vector< double > &Flowrates_w)
Returns the mineral and waste flow rates of each unit in the circuit.
- double [ReturnProfit](#) ()
Calculates and returns the profit based on the current unit's data.
- void [ResetFlowrates](#) ()
Resets the flow rates of all units in the circuit to zero.
- void [SetInlet](#) ()
Sets the inlet flow rates for each unit in the circuit.
- bool [Check_VValidity](#) (int *circuit_vector)
Function to check the validity of a circuit.
- void [mark_units](#) (int unit_num)
Function to mark all units accessible from the feed.

Public Attributes

- int num_units
- double system_mineral_input
- double system_waste_input
- std::vector< double > recycle_mineral
- std::vector< double > recycle_waste
- std::vector< int > CCircuit_vector
- std::vector< CUnit > units

3.2.1 Detailed Description

header file for the circuit

This header file defines the function that will be used to generate and validate the circuit

3.2.2 Constructor & Destructor Documentation

3.2.2.1 CCircuit() [1/3]

```
CCircuit::CCircuit ( )
```

3.2.2.2 CCircuit() [2/3]

```
CCircuit::CCircuit (
    int num_units )
```

Constructor of the [CCircuit](#) class that initialises the vector of [CUnit](#) units.

3.2.2.3 CCircuit() [3/3]

```
CCircuit::CCircuit (
    int num_units_,
    const std::vector< int > & CCircuit_vector_ )
```

Constructor of the [CCircuit](#) class with specified number of units and a circuit vector.

This constructor initializes a [CCircuit](#) object with the given number of units and circuit vector.

Parameters

<i>num_units_</i>	The number of units in the circuit.
<i>CCircuit_↵ vector_</i>	The circuit vector containing information about the circuit.

3.2.3 Member Function Documentation

3.2.3.1 Check_Validity()

```
bool CCircuit::Check_Validity (
    int * circuit_vector )
```

Function to check the validity of a circuit.

Parameters

<i>int</i>	* circuit_vector Pointer to a circuit vector that contains information about the circuit The function iterates through circuit to check its validity
------------	--

3.2.3.2 FillIDs()

```
void CCircuit::FillIDs ( )
```

Fills the IDs of the [CCircuit](#) units and initializes related data structures. This function fills the IDs of the [CCircuit](#) units and performs the following steps: Resizes the units, recycle_mineral, and recycle_waste vectors based on the number of units. Initializes the recycle_mineral and recycle_waste vectors with zeros. Assigns the appropriate IDs to each unit based on the CCircuit_vector. Sets the IDs for the concentrate and tailing outlets of the system.

Note

This function assumes that the [CCircuit](#), units, and related data structures have been properly initialized.

See also

[CCircuit](#)

Parameters

<i>CCircuit_vector</i>	A vector containing the IDs of the units in the CCircuit .
------------------------	--

3.2.3.3 mark_units()

```
void CCircuit::mark_units (
    int unit_num )
```

Function to mark all units accessible from the feed.

Parameters

<i>int</i>	unit_num Integer value representing the initial unit This function is recursively called to mark all units that can be accessed from the feed
------------	---

3.2.3.4 ResetFlowrates()

```
void CCircuit::ResetFlowrates ( )
```

Resets the flow rates of all units in the circuit to zero.

This function sets the flow rates of all units in the circuit to zero. It iterates through each unit and resets various flow rate variables to zero, including Fti, Fgi, Fwi, Ftc, Fgc, Fwc, Ftt, Fgt, Fwt, Rg, Rw, and tau.

3.2.3.5 ReturnFlowrates()

```
void CCircuit::ReturnFlowrates (
    std::vector< double > & Flowrates_g,
    std::vector< double > & Flowrates_w )
```

Returns the mineral and waste flow rates of each unit in the circuit.

This function populates the provided vectors with the mineral and waste flow rates of each unit in the circuit. The function resizes the `Flowrates_g` and `Flowrates_w` vectors to accommodate the required number of flow rates. Then, it iterates through each unit and assigns the mineral and waste flow rates to the corresponding elements in the vectors.

Parameters

out	<i>Flowrates_g</i>	A vector that will be filled with the mineral flow rates of each unit.
out	<i>Flowrates_w</i>	A vector that will be filled with the waste flow rates of each unit.

Note

The function assumes that the `Flowrates_g` and `Flowrates_w` vectors have been properly initialized before calling this function.

3.2.3.6 ReturnProfit()

```
double CCircuit::ReturnProfit ( )
```

Calculates and returns the profit based on the current unit's data.

This function calculates the profit based on the current unit's data and returns the result.

Returns

The calculated profit based on the formula: $\text{ProfitGerardium} * \text{Fgi} - \text{PenalizationWaste} * \text{Fwi}$.

3.2.3.7 SetInlet()

```
void CCircuit::SetInlet ( )
```

Sets the inlet flow rates for each unit in the circuit.

This function sets the inlet flow rates for each unit in the circuit based on the system feed ID and the provided recycle mineral and waste vectors. It iterates through each unit and checks if the unit's index matches the system feed ID. If a match is found, the mineral and waste input flow rates (F_{gi} and F_{wi}) as well as the total input flow rate (F_{ti}) are set to the sum of the corresponding recycle flow rates, system mineral input, and system waste input. For units that don't match the system feed ID, the inlet flow rates are set to the corresponding recycle flow rates.

Note

The function assumes that the `CCircuit_vector`, `recycle_mineral`, and `recycle_waste` vectors have been properly initialized before calling this function.

3.2.3.8 SolveCCircuit()

```
void CCircuit::SolveCCircuit ( )
```

Solves the [CCircuit](#) by iterating through its units.

This function solves the [CCircuit](#) by performing the following steps: Resets the flow rates. Sets the inlet. Initializes the `recycle_mineral` and `recycle_waste` vectors with zeros. Iterates through each unit in the circuit and solves it using `SolveCUnit()`. Updates the `recycle_mineral` and `recycle_waste` vectors based on the unit outputs.

Note

This function assumes that the [CCircuit](#) and its units have been properly initialized.

See also

[CCircuit](#), [SolveCUnit\(\)](#)

3.2.3.9 SystemFlowrates()

```
void CCircuit::SystemFlowrates (
    const double Fgi_,
    const double Fwi_ )
```

Sets the system flow rates for mineral and waste inputs.

This function sets the system flow rates for mineral and waste inputs based on the provided values.

Parameters

$Sg \leftrightarrow$ —	The mineral input flow rate to be set for the system.
$Sw \leftrightarrow$ —	The waste input flow rate to be set for the system.

3.2.4 Member Data Documentation

3.2.4.1 CCircuit_vector

```
std::vector<int> CCircuit::CCircuit_vector
```

3.2.4.2 num_units

```
int CCircuit::num_units
```

3.2.4.3 recycle_mineral

```
std::vector<double> CCircuit::recycle_mineral
```

3.2.4.4 recycle_waste

```
std::vector<double> CCircuit::recycle_waste
```

3.2.4.5 system_mineral_input

```
double CCircuit::system_mineral_input
```

3.2.4.6 system_waste_input

```
double CCircuit::system_waste_input
```

3.2.4.7 units

```
std::vector<CUnit> CCircuit::units
```

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

- include/CCircuit.h
- src/CCircuit.cpp

3.3 Circuit_Parameters Struct Reference

```
#include <CSimulator.h>
```

Public Attributes

- double [tolerance](#)
- int [max_iterations](#)

3.3.1 Detailed Description

header file for the circuit simulator

This header file defines the function that will be used to evaluate the circuit

3.3.2 Member Data Documentation

3.3.2.1 max_iterations

```
int Circuit_Parameters::max_iterations
```

3.3.2.2 tolerance

```
double Circuit_Parameters::tolerance
```

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

- include/CSimulator.h

3.4 CUnit Class Reference

```
#include <CUnit.h>
```

Public Member Functions

- void [ConcentrationCalc](#) ()
Calculates the concentrations of components in the [CUnit](#). This function calculates the concentrations of components in the [CUnit](#) by dividing the corresponding flow rates by the total flow rate.
- void [ResidenceTime](#) ()
Calculates the residence time of the [CUnit](#).
- void [ReactionComp](#) ()
Calculates the reaction components.
- void [OutletCalc](#) ()
Calculates the outlet components.
- int [CheckMassBalance](#) ()
Checks the mass balance of the unit.
- void [SolveCUnit](#) ()
Solves the [CUnit](#) by performing necessary calculations. This function solves the [CUnit](#) by performing the following steps: Calculates the residence time. Performs reaction composition calculations. Calculates the outlet values. Calculates the concentration.
- void [PrintCUnit](#) ()
Prints the [CUnit](#) information and performs a mass balance check. This function prints various components and flow rates at the inlet, concentrate, and tailing of the [CUnit](#). It also displays the spatial time and reaction rates. Finally, it calls the [CheckMassBalance\(\)](#) function to perform a mass balance check.
- std::vector< double > [ReturnOutlets](#) ()
Returns the outlet components.
- [CUnit](#) ()
- [CUnit](#) (int ID_num_, double Fgi_, double Fwi_)

Public Attributes

- int [ID_num](#)
- int [ID_num_conc](#)
- int [ID_num_tail](#)
- bool [mark](#)
- int [conc_num](#)
- int [tails_num](#)
- double [tau](#)
- double [Rg](#)
- double [Rw](#)
- double [Fti](#)
- double [Fgi](#)
- double [Fwi](#)
- double [Cgi](#)
- double [Cwi](#)
- double [Ftc](#)
- double [Fgc](#)
- double [Fwc](#)
- double [Cgc](#)
- double [Cwc](#)
- double [Ftt](#)
- double [Fgt](#)
- double [Fwt](#)
- double [Cgt](#)
- double [Cwt](#)

3.4.1 Detailed Description

header file for the units of the ircuit

This header file defines the function that will be used to initialize the units in the circuit

3.4.2 Constructor & Destructor Documentation

3.4.2.1 CUnit() [1/2]

```
CUnit::CUnit ( ) [inline]
```

3.4.2.2 CUnit() [2/2]

```
CUnit::CUnit (
    int ID_num_,
    double Fgi_,
    double Fwi_ ) [inline]
```

3.4.3 Member Function Documentation

3.4.3.1 CheckMassBalance()

```
int CUnit::CheckMassBalance ( )
```

Checks the mass balance of the unit.

This method checks the mass balance of the unit by comparing the inlet and outlet components and printing the results. It performs the following checks:

- Global Total mass balance
- Global Gerardium mass balance
- Global Waste mass balance
- Mass balance at the inlet
- Mass balance at the concentrate
- Mass balance at the tailing
- Global general mass balance

3.4.3.2 ConcentrationCalc()

```
void CUnit::ConcentrationCalc ( )
```

Calculates the concentrations of components in the [CUnit](#). This function calculates the concentrations of components in the [CUnit](#) by dividing the corresponding flow rates by the total flow rate.

Note

This function assumes that the [CUnit](#) and its related data have been properly initialized.

See also

[CUnit](#)

3.4.3.3 OutletCalc()

```
void CUnit::OutletCalc ( )
```

Calculates the outlet components.

This method calculates the outlet components based on the reaction rates and the input components.

3.4.3.4 PrintCUnit()

```
void CUnit::PrintCUnit ( )
```

Prints the [CUnit](#) information and performs a mass balance check. This function prints various components and flow rates at the inlet, concentrate, and tailing of the [CUnit](#). It also displays the spatial time and reaction rates. Finally, it calls the [CheckMassBalance\(\)](#) function to perform a mass balance check.

Note

This function assumes that the [CUnit](#) and its related data have been properly initialized.

See also

[CUnit](#), [CheckMassBalance\(\)](#)

3.4.3.5 ReactionComp()

```
void CUnit::ReactionComp ( )
```

Calculates the reaction components.

This method calculates the reaction components based on the reaction rate constants for Gerardium and waste.

3.4.3.6 ResidenceTime()

```
void CUnit::ResidenceTime ( )
```

Calculates the residence time of the [CUnit](#).

This function calculates the residence time of the [CUnit](#) based on the given parameters: gamma, V, pg, pw, Fgi, and Fwi.

Note

This function assumes that the [CUnit](#) and its related data have been properly initialized.

See also

[CUnit](#)

3.4.3.7 ReturnOutlets()

```
std::vector< double > CUnit::ReturnOutlets ( )
```

Returns the outlet components.

This method returns a vector containing the outlet components: Gerardium component (Fgc), waste component (Fwc), total Gerardium component (Fgt), and total waste component (Fwt).

Returns

Vector containing the outlet components.

3.4.3.8 SolveCUnit()

```
void CUnit::SolveCUnit ( )
```

Solves the [CUnit](#) by performing necessary calculations. This function solves the [CUnit](#) by performing the following steps: Calculates the residence time. Performs reaction composition calculations. Calculates the outlet values. Calculates the concentration.

Note

This function assumes that the [CUnit](#) and its related data have been properly initialized.

See also

[CUnit](#), [ResidenceTime\(\)](#), [ReactionComp\(\)](#), [OutletCalc\(\)](#), [ConcentrationCalc\(\)](#)

3.4.4 Member Data Documentation

3.4.4.1 Cgc

`double CUnit::Cgc`

3.4.4.2 Cgi

`double CUnit::Cgi`

3.4.4.3 Cgt

`double CUnit::Cgt`

3.4.4.4 conc_num

`int CUnit::conc_num`

3.4.4.5 Cwc

`double CUnit::Cwc`

3.4.4.6 Cwi

`double CUnit::Cwi`

3.4.4.7 Cwt

`double CUnit::Cwt`

3.4.4.8 Fgc

```
double CUnit::Fgc
```

3.4.4.9 Fgi

```
double CUnit::Fgi
```

3.4.4.10 Fgt

```
double CUnit::Fgt
```

3.4.4.11 Ftc

```
double CUnit::Ftc
```

3.4.4.12 Fti

```
double CUnit::Fti
```

3.4.4.13 Ftt

```
double CUnit::Ftt
```

3.4.4.14 Fwc

```
double CUnit::Fwc
```

3.4.4.15 Fwi

```
double CUnit::Fwi
```

3.4.4.16 Fwt

```
double CUnit::Fwt
```

3.4.4.17 ID_num

```
int CUnit::ID_num
```

3.4.4.18 ID_num_conc

```
int CUnit::ID_num_conc
```

3.4.4.19 ID_num_tail

```
int CUnit::ID_num_tail
```

3.4.4.20 mark

```
bool CUnit::mark
```

3.4.4.21 Rg

```
double CUnit::Rg
```

3.4.4.22 Rw

```
double CUnit::Rw
```

3.4.4.23 tails_num

```
int CUnit::tails_num
```

3.4.4.24 tau

```
double CUnit::tau
```

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

- [include/CUnit.h](#)
- [src/CUnit.cpp](#)

3.5 Individual Struct Reference

Represents an individual in the population.

```
#include <Genetic_Algorithm.h>
```

Public Attributes

- `std::vector< int >` [vector](#)
- `double` [fitness_val](#)

3.5.1 Detailed Description

Represents an individual in the population.

3.5.2 Member Data Documentation

3.5.2.1 fitness_val

```
Individual::fitness_val
```

The fitness value of the individual, calculated using a provided fitness function.

3.5.2.2 vector

```
Individual::vector
```

A vector representing the state of the individual.

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

- [include/Genetic_Algorithm.h](#)

Chapter 4

File Documentation

4.1 include/CCircuit.h File Reference

```
#include "CUnit.h"
#include <vector>
#include <iostream>
#include <algorithm>
```

Include dependency graph for CCircuit.h: This graph shows which files directly or indirectly include this file:

Classes

- class [CCircuit](#)

4.2 include/CSimulator.h File Reference

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

Classes

- struct [Circuit_Parameters](#)

Functions

- double [Evaluate_Circuit](#) (int vector_size, int *circuit_vector, struct [Circuit_Parameters](#) parameters)
- double [Evaluate_Circuit](#) (int vector_size, int *circuit_vector)

4.2.1 Function Documentation

4.2.1.1 Evaluate_Circuit() [1/2]

```
double Evaluate_Circuit (
    int vector_size,
    int * circuit_vector )
```

4.2.1.2 Evaluate_Circuit() [2/2]

```
double Evaluate_Circuit (
    int vector_size,
    int * circuit_vector,
    struct Circuit_Parameters parameters )
```

Parameters

<i>vector_size</i>	the size of the circuit vector
<i>circuit_vector</i>	the pointer of the circuit vector array
<i>parameters</i>	circuit parameters: tolerance and maximum iterations

Returns

The monetary value of the final concentrate.

4.3 include/CUnit.h File Reference

```
#include <iostream>
#include <utility>
#include <vector>
#include <cmath>
```

Include dependency graph for CUnit.h: This graph shows which files directly or indirectly include this file:

Classes

- class [CUnit](#)

4.4 include/Genetic_Algorithm.h File Reference

```
#include <stdio.h>
#include <cmath>
#include <array>
#include <algorithm>
#include <vector>
#include <random>
#include <iostream>
#include <chrono>
#include <numeric>
#include <memory>
#include "CCircuit.h"
```

Include dependency graph for Genetic_Algorithm.h: This graph shows which files directly or indirectly include this file:

Classes

- struct [Algorithm_Parameters](#)
Contains parameters used to control the behavior of the genetic algorithm.
- struct [Individual](#)
Represents an individual in the population.

Functions

- double [Evaluate_circuit](#) (int vector_size, int *circuit_vector)
Evaluate the performance of a given circuit vector.
- std::vector< [Individual](#) > [InitializePopulation](#) (int vector_size, std::mt19937 &gen, std::uniform_int_distribution<> &uniform_int, [CCircuit](#) &circuit, [Algorithm_Parameters](#) ¶meters)
Initialize a population for the genetic algorithm.
- void [CalculateFitness](#) (std::vector< [Individual](#) > &parents, double(&func)(int, int *))
Calculate the fitness of each individual in a population.
- void [SortParentsByFitness](#) (std::vector< [Individual](#) > &parents)
Sorts individuals in a population by fitness.
- std::unique_ptr< std::vector< double > > [NormalizeFitness](#) (std::vector< [Individual](#) > &parents)
Normalize the fitness values in a population.
- std::unique_ptr< std::vector< double > > [MakeCDF](#) (std::unique_ptr< std::vector< double > > &normalized)
Create a cumulative distribution function from a normalized distribution.
- std::pair< [Individual](#), [Individual](#) > [SelectParents](#) (std::vector< [Individual](#) > &parents, std::unique_ptr< std::vector< double > > &cdf, std::mt19937 &gen, std::uniform_real_distribution<> &uniform_real)
Select two parents for reproduction using a cumulative distribution function.
- std::pair< [Individual](#), [Individual](#) > [Crossover](#) ([Individual](#) &selected_parentX, [Individual](#) &selected_parentY, std::mt19937 &gen, std::uniform_real_distribution<> &uniform_real, [Algorithm_Parameters](#) ¶meters, int vector_size)
Perform a crossover operation on two parents to produce offspring.
- void [Mutation](#) ([Individual](#) &childX, [Individual](#) &childY, std::mt19937 &gen, std::uniform_real_distribution<> &uniform_real, std::uniform_int_distribution<> &uniform_int, std::uniform_real_distribution<> &uniform_mutation_step, [Algorithm_Parameters](#) ¶meters, int vector_size)
Perform mutation on two individuals.
- void [AddChildren](#) (std::vector< [Individual](#) > &children, [Individual](#) &childX, [Individual](#) &childY, [CCircuit](#) &circuit)
Add valid children to the population.
- double [optimize](#) (int vector_size, int *best_vector, double(&func)(int, int *), [Algorithm_Parameters](#) parameters)
Run the genetic algorithm to optimize the circuit.

4.4.1 Function Documentation

4.4.1.1 AddChildren()

```
void AddChildren (
    std::vector< Individual > & children,
    Individual & childX,
    Individual & childY,
    CCircuit & circuit )
```

Add valid children to the population.

Parameters

<i>children</i>	A vector of individuals that will be updated with valid children.
<i>childX</i>	The first child to be added if it's valid.
<i>childY</i>	The second child to be added if it's valid.
<i>circuit</i>	The circuit that is being optimized.

4.4.1.2 CalculateFitness()

```
void CalculateFitness (
    std::vector< Individual > & parents,
    double(&)(int, int *) func )
```

Calculate the fitness of each individual in a population.

Parameters

<i>parents</i>	A vector of individuals for which to calculate fitness.
<i>func</i>	A function that calculates fitness.

4.4.1.3 Crossover()

```
std::pair<Individual, Individual> Crossover (
    Individual & selected_parentX,
    Individual & selected_parentY,
    std::mt19937 & gen,
    std::uniform_real_distribution<> & uniform_real,
    Algorithm_Parameters & parameters,
    int vector_size )
```

Perform a crossover operation on two parents to produce offspring.

Parameters

<i>selected_parentX</i>	The first parent.
<i>selected_parentY</i>	The second parent.
<i>gen</i>	The random number generator.
<i>uniform_real</i>	A uniform real distribution.
<i>parameters</i>	The parameters of the genetic algorithm.
<i>vector_size</i>	The size of the vectors representing individuals in the population.

Returns

A pair of individuals representing the offspring.

4.4.1.4 Evaluate_circuit()

```
double Evaluate_circuit (
    int vector_size,
    int * circuit_vector )
```

Evaluate the performance of a given circuit vector.

Parameters

<i>vector_size</i>	The size of the circuit vector.
<i>circuit_vector</i>	A pointer to the circuit vector.

Returns

A double value representing the performance of the circuit vector.

4.4.1.5 InitializePopulation()

```
std::vector<Individual> InitializePopulation (
    int vector_size,
    std::mt19937 & gen,
    std::uniform_int_distribution<> & uniform_int,
    CCircuit & circuit,
    Algorithm_Parameters & parameters )
```

Initialize a population for the genetic algorithm.

Parameters

<i>vector_size</i>	The size of the vectors representing individuals in the population.
<i>gen</i>	The random number generator.
<i>uniform_int</i>	A uniform integer distribution.
<i>circuit</i>	The circuit that is being optimized.
<i>parameters</i>	The parameters of the genetic algorithm.

Returns

A vector of individuals representing the initial population.

4.4.1.6 MakeCDF()

```
std::unique_ptr<std::vector<double> > MakeCDF (
    std::unique_ptr< std::vector< double >> & normalized )
```

Create a cumulative distribution function from a normalized distribution.

Parameters

<i>normalized</i>	A pointer to the normalized distribution.
-------------------	---

Returns

A pointer to the cumulative distribution function.

4.4.1.7 Mutation()

```
void Mutation (
    Individual & childX,
    Individual & childY,
    std::mt19937 & gen,
    std::uniform_real_distribution<> & uniform_real,
    std::uniform_int_distribution<> & uniform_int,
    std::uniform_real_distribution<> & uniform_mutation_step,
    AlgorithmParameters & parameters,
    int vector_size )
```

Perform mutation on two individuals.

Parameters

<i>childX</i>	The first individual.
<i>childY</i>	The second individual.
<i>gen</i>	The random number generator.
<i>uniform_real</i>	A uniform real distribution.
<i>uniform_int</i>	A uniform integer distribution.
<i>uniform_mutation_step</i>	A uniform real distribution used to generate mutation steps.
<i>parameters</i>	The parameters of the genetic.

4.4.1.8 NormalizeFitness()

```
std::unique_ptr<std::vector<double> > NormalizeFitness (
    std::vector< Individual > & parents )
```

Normalize the fitness values in a population.

Parameters

<i>parents</i>	A vector of individuals whose fitness values will be normalized.
----------------	--

Returns

A vector of normalized fitness values.

4.4.1.9 optimize()

```
double optimize (
    int vector_size,
    int * best_vector,
    double(&)(int, int *) func,
    Algorithm_Parameters parameters )
```

Run the genetic algorithm to optimize the circuit.

Parameters

<i>vector_size</i>	The size of the vectors representing individuals in the population.
<i>func</i>	A function that calculates fitness.
<i>parameters</i>	The parameters of the genetic algorithm.

Returns

The fitness value of the best individual found.

4.4.1.10 SelectParents()

```
std::pair<Individual, Individual> SelectParents (
    std::vector< Individual > & parents,
    std::unique_ptr< std::vector< double >> & cdf,
    std::mt19937 & gen,
    std::uniform_real_distribution<> & uniform_real )
```

Select two parents for reproduction using a cumulative distribution function.

Parameters

<i>parents</i>	A vector of individuals from which to select parents.
<i>cdf</i>	A pointer to the cumulative distribution function.
<i>gen</i>	The random number generator.
<i>uniform_real</i>	A uniform real distribution.

Returns

A pair of individuals that have been selected as parents.

4.4.1.11 SortParentsByFitness()

```
void SortParentsByFitness (
    std::vector< Individual > & parents )
```

Sorts individuals in a population by fitness.

Parameters

<i>parents</i>	A vector of individuals to be sorted.
----------------	---------------------------------------

4.5 include/HyperPSearch.h File Reference

```
#include "Genetic_Algorithm.h"
```

Include dependency graph for HyperPSearch.h: This graph shows which files directly or indirectly include this file:

Functions

- [Algorithm_Parameters generateRandomParameters](#) (int vector_size)
header file to perform hyperparamter search
- void [randomSearch](#) (int iterCount, int *best_vector, int vector_size)
Performs random search to find the best hyperparameters for genetic algorithm.

4.5.1 Function Documentation

4.5.1.1 generateRandomParameters()

```
Algorithm_Parameters generateRandomParameters (
    int vector_size )
```

header file to perform hyperparamter search

header file to perform hyperparamter search

This function generates a set of genetic algorithm parameters (both random and non-random). Random parameters: population size, mutation rate, crossover rate, mutation type rate, crossover type rate. Non-random parameters: generation step, mutation max step, and early stopping generations.

Parameters

<i>vector_size</i>	Genetic algorithm vector size for mutation max step and early stopping calculation.
--------------------	---

Returns

[Algorithm_Parameters](#) The randomly generated parameters.

4.5.1.2 randomSearch()

```
void randomSearch (
    int iterCount,
    int * best_vector,
    int vector_size )
```

Performs random search to find the best hyperparameters for genetic algorithm.

This function performs a search by iteratively (and randomly) generating algorithm parameters, and evaluating the genetic algorithm output with those parameters. At the end, it outputs the best score, best hyperparameters, and best vector to the console and writes the best score to a file.

Parameters

<i>iterCount</i>	The number of iterations to perform.
<i>best_vector</i>	A pointer to an array that will hold the best vector.
<i>vector_size</i>	Genetic algorithm vector size.

4.6 src/CCircuit.cpp File Reference

```
#include "CCircuit.h"
#include <iostream>
Include dependency graph for CCircuit.cpp:
```

4.7 src/CMakeLists.txt File Reference**4.8 src/CSimulator.cpp File Reference**

```
#include "CCircuit.h"
#include "CSimulator.h"
#include <iostream>
#include <vector>
#include <cmath>
Include dependency graph for CSimulator.cpp:
```

Functions

- double [Evaluate_Circuit](#) (int vector_size, int *circuit_vector, struct [Circuit_Parameters](#) parameters)
- double [Evaluate_Circuit](#) (int vector_size, int *circuit_vector)

Variables

- struct `Circuit_Parameters` `default_circuit_parameters` = {0.01, 1000}

4.8.1 Function Documentation

4.8.1.1 Evaluate_Circuit() [1/2]

```
double Evaluate_Circuit (
    int vector_size,
    int * circuit_vector )
```

4.8.1.2 Evaluate_Circuit() [2/2]

```
double Evaluate_Circuit (
    int vector_size,
    int * circuit_vector,
    struct Circuit_Parameters parameters )
```

Parameters

<i>vector_size</i>	the size of the circuit vector
<i>circuit_vector</i>	the pointer of the circuit vector array
<i>parameters</i>	circuit parameters: tolerance and maximum iterations

Returns

The monetary value of the final concentrate.

4.8.2 Variable Documentation

4.8.2.1 default_circuit_parameters

```
struct Circuit_Parameters default_circuit_parameters = {0.01, 1000}
```

4.9 src/CUnit.cpp File Reference

```
#include "CUnit.h"
Include dependency graph for CUnit.cpp:
```

Functions

- bool `isClose` (double a, double b, double tolerance)

Checks if two double values are close within a specified tolerance. This function compares two double values and determines if they are close to each other within the specified tolerance. The comparison is performed using the absolute difference between the two values.

4.9.1 Function Documentation

4.9.1.1 isClose()

```
bool isClose (
    double a,
    double b,
    double tolerance )
```

Checks if two double values are close within a specified tolerance. This function compares two double values and determines if they are close to each other within the specified tolerance. The comparison is performed using the absolute difference between the two values.

Parameters

<i>a</i>	The first double value to compare.
<i>b</i>	The second double value to compare.
<i>tolerance</i>	The tolerance value within which the two values are considered close.

Returns

true if the absolute difference between the two values is less than or equal to the tolerance, false otherwise.

4.10 src/Genetic_Algorithm.cpp File Reference

```
#include "Genetic_Algorithm.h"
#include "CCircuit.h"
#include <stdio.h>
#include <cmath>
#include <array>
#include <algorithm>
#include <vector>
#include <random>
#include <iostream>
#include <chrono>
#include <numeric>
#include <string>
#include <fstream>
#include <memory>
#include "omp.h"
Include dependency graph for Genetic_Algorithm.cpp:
```

Functions

- double [Evaluate_circuit](#) (int vector_size, int *circuit_vector)
Evaluate the performance of a given circuit vector.
- std::vector< [Individual](#) > [InitializePopulation](#) (int vector_size, std::mt19937 &gen, std::uniform_int_distribution<> &uniform_int, [CCircuit](#) &circuit, [Algorithm_Parameters](#) ¶meters)
Initialize a population for the genetic algorithm.
- void [CalculateFitness](#) (std::vector< [Individual](#) > &parents, double(&func)(int, int *))
Calculate the fitness of each individual in a population.
- void [SortParentsByFitness](#) (std::vector< [Individual](#) > &parents)
Sorts individuals in a population by fitness.
- std::unique_ptr< std::vector< double > > [NormalizeFitness](#) (std::vector< [Individual](#) > &parents)
Normalize the fitness values in a population.
- std::unique_ptr< std::vector< double > > [MakeCDF](#) (std::unique_ptr< std::vector< double >> &normalized)
Create a cumulative distribution function from a normalized distribution.
- std::pair< [Individual](#), [Individual](#) > [SelectParents](#) (std::vector< [Individual](#) > &parents, std::unique_ptr< std::vector< double >> &cdf, std::mt19937 &gen, std::uniform_real_distribution<> &uniform_real)
Select two parents for reproduction using a cumulative distribution function.
- std::pair< [Individual](#), [Individual](#) > [Crossover](#) ([Individual](#) &selected_parentX, [Individual](#) &selected_parentY, std::mt19937 &gen, std::uniform_real_distribution<> &uniform_real, [Algorithm_Parameters](#) ¶meters, int vector_size)
Perform a crossover operation on two parents to produce offspring.
- void [Mutation](#) ([Individual](#) &childX, [Individual](#) &childY, std::mt19937 &gen, std::uniform_real_distribution<> &uniform_real, std::uniform_int_distribution<> &uniform_int, std::uniform_real_distribution<> &uniform_real, std::uniform_int_distribution<> &uniform_int, std::uniform_real_distribution<> &uniform_real, [Algorithm_Parameters](#) ¶meters, int vector_size)
Perform mutation on two individuals.
- void [AddChildren](#) (std::vector< [Individual](#) > &children, [Individual](#) &childX, [Individual](#) &childY, [CCircuit](#) &circuit)
Add valid children to the population.
- double [optimize](#) (int vector_size, int *best_vector, double(&func)(int, int *), [Algorithm_Parameters](#) parameters)
Run the genetic algorithm to optimize the circuit.

4.10.1 Function Documentation

4.10.1.1 AddChildren()

```
void AddChildren (
    std::vector< Individual > & children,
    Individual & childX,
    Individual & childY,
    CCircuit & circuit )
```

Add valid children to the population.

Parameters

<i>children</i>	A vector of individuals that will be updated with valid children.
<i>childX</i>	The first child to be added if it's valid.
<i>childY</i>	The second child to be added if it's valid.
<i>circuit</i>	The circuit that is being optimized.

4.10.1.2 CalculateFitness()

```
void CalculateFitness (
    std::vector< Individual > & parents,
    double(&)(int, int *) func )
```

Calculate the fitness of each individual in a population.

Parameters

<i>parents</i>	A vector of individuals for which to calculate fitness.
<i>func</i>	A function that calculates fitness.

4.10.1.3 Crossover()

```
std::pair<Individual, Individual> Crossover (
    Individual & selected_parentX,
    Individual & selected_parentY,
    std::mt19937 & gen,
    std::uniform_real_distribution<> & uniform_real,
    Algorithm_Parameters & parameters,
    int vector_size )
```

Perform a crossover operation on two parents to produce offspring.

Parameters

<i>selected_parentX</i>	The first parent.
<i>selected_parentY</i>	The second parent.
<i>gen</i>	The random number generator.
<i>uniform_real</i>	A uniform real distribution.
<i>parameters</i>	The parameters of the genetic algorithm.
<i>vector_size</i>	The size of the vectors representing individuals in the population.

Returns

A pair of individuals representing the offspring.

4.10.1.4 Evaluate_circuit()

```
double Evaluate_circuit (
    int vector_size,
    int * circuit_vector )
```

Evaluate the performance of a given circuit vector.

Parameters

<i>vector_size</i>	The size of the circuit vector.
<i>circuit_vector</i>	A pointer to the circuit vector.

Returns

A double value representing the performance of the circuit vector.

4.10.1.5 InitializePopulation()

```
std::vector<Individual> InitializePopulation (
    int vector_size,
    std::mt19937 & gen,
    std::uniform_int_distribution<> & uniform_int,
    CCircuit & circuit,
    AlgorithmParameters & parameters )
```

Initialize a population for the genetic algorithm.

Parameters

<i>vector_size</i>	The size of the vectors representing individuals in the population.
<i>gen</i>	The random number generator.
<i>uniform_int</i>	A uniform integer distribution.
<i>circuit</i>	The circuit that is being optimized.
<i>parameters</i>	The parameters of the genetic algorithm.

Returns

A vector of individuals representing the initial population.

4.10.1.6 MakeCDF()

```
std::unique_ptr<std::vector<double> > MakeCDF (
    std::unique_ptr< std::vector< double >> & normalized )
```

Create a cumulative distribution function from a normalized distribution.

Parameters

<i>normalized</i>	A pointer to the normalized distribution.
-------------------	---

Returns

A pointer to the cumulative distribution function.

4.10.1.7 Mutation()

```
void Mutation (
    Individual & childX,
    Individual & childY,
    std::mt19937 & gen,
    std::uniform_real_distribution<> & uniform_real,
    std::uniform_int_distribution<> & uniform_int,
    std::uniform_real_distribution<> & uniform_mutation_step,
    Algorithm_Parameters & parameters,
    int vector_size )
```

Perform mutation on two individuals.

Parameters

<i>childX</i>	The first individual.
<i>childY</i>	The second individual.
<i>gen</i>	The random number generator.
<i>uniform_real</i>	A uniform real distribution.
<i>uniform_int</i>	A uniform integer distribution.
<i>uniform_mutation_step</i>	A uniform real distribution used to generate mutation steps.
<i>parameters</i>	The parameters of the genetic.

4.10.1.8 NormalizeFitness()

```
std::unique_ptr<std::vector<double> > NormalizeFitness (
    std::vector< Individual > & parents )
```

Normalize the fitness values in a population.

Parameters

<i>parents</i>	A vector of individuals whose fitness values will be normalized.
----------------	--

Returns

A vector of normalized fitness values.

4.10.1.9 optimize()

```
double optimize (
    int vector_size,
    int * best_vector,
    double(&)(int, int *) func,
    Algorithm_Parameters parameters )
```

Run the genetic algorithm to optimize the circuit.

Parameters

<i>vector_size</i>	The size of the vectors representing individuals in the population.
<i>func</i>	A function that calculates fitness.
<i>parameters</i>	The parameters of the genetic algorithm.

Returns

The fitness value of the best individual found.

4.10.1.10 SelectParents()

```
std::pair<Individual, Individual> SelectParents (
    std::vector< Individual > & parents,
    std::unique_ptr< std::vector< double >> & cdf,
    std::mt19937 & gen,
    std::uniform_real_distribution<> & uniform_real )
```

Select two parents for reproduction using a cumulative distribution function.

Parameters

<i>parents</i>	A vector of individuals from which to select parents.
<i>cdf</i>	A pointer to the cumulative distribution function.
<i>gen</i>	The random number generator.
<i>uniform_real</i>	A uniform real distribution.

Returns

A pair of individuals that have been selected as parents.

4.10.1.11 SortParentsByFitness()

```
void SortParentsByFitness (
    std::vector< Individual > & parents )
```

Sorts individuals in a population by fitness.

Parameters

<i>parents</i>	A vector of individuals to be sorted.
----------------	---------------------------------------

4.11 src/HyperPSearch.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <random>
#include <vector>
#include <filesystem>
#include "Genetic_Algorithm.h"
#include "CUnit.h"
#include "CCircuit.h"
#include "CSimulator.h"
#include "HyperPSearch.h"
Include dependency graph for HyperPSearch.cpp:
```

Functions

- [Algorithm_Parameters generateRandomParameters](#) (int vector_size)
Generates algorithm parameters randomly.
- void [randomSearch](#) (int iterCount, int *best_vector, int vector_size)
Performs random search to find the best hyperparameters for genetic algorithm.

4.11.1 Function Documentation

4.11.1.1 generateRandomParameters()

```
Algorithm_Parameters generateRandomParameters (
    int vector_size )
```

Generates algorithm parameters randomly.

header file to perform hyperparamter search

This function generates a set of genetic algorithm parameters (both random and non-random). Random parameters: population size, mutation rate, crossover rate, mutation type rate, crossover type rate. Non-random parameters: generation step, mutation max step, and early stopping generations.

Parameters

<i>vector_size</i>	Genetic algorithm vector size for mutation max step and early stopping calculation.
--------------------	---

Returns

[Algorithm_Parameters](#) The randomly generated parameters.

4.11.1.2 randomSearch()

```
void randomSearch (
    int iterCount,
    int * best_vector,
    int vector_size )
```

Performs random search to find the best hyperparameters for genetic algorithm.

This function performs a search by iteratively (and randomly) generating algorithm parameters, and evaluating the genetic algorithm output with those parameters. At the end, it outputs the best score, best hyperparameters, and best vector to the console and writes the best score to a file.

Parameters

<i>iterCount</i>	The number of iterations to perform.
<i>best_vector</i>	A pointer to an array that will hold the best vector.
<i>vector_size</i>	Genetic algorithm vector size.

4.12 src/main.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <filesystem>
#include "Genetic_Algorithm.h"
#include "CCircuit.h"
#include "CUnit.h"
#include "CSimulator.h"
#include "HyperPSearch.h"
```

Include dependency graph for main.cpp:

Functions

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

4.12.1 Function Documentation**4.12.1.1 main()**

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

Index

AddChildren
 Genetic_Algorithm.cpp, [34](#)
 Genetic_Algorithm.h, [25](#)
Algorithm_Parameters, [5](#)
 crossover_rate, [5](#)
 crossover_type_rate, [5](#)
 early_stopping_generations, [6](#)
 generation_step, [6](#)
 mutation_max_step, [6](#)
 mutation_rate, [6](#)
 mutation_type_rate, [6](#)
 population_size, [6](#)
CalculateFitness
 Genetic_Algorithm.cpp, [35](#)
 Genetic_Algorithm.h, [26](#)
CCircuit, [7](#)
 CCircuit, [8](#)
 CCircuit_vector, [12](#)
 Check_Validity, [8](#)
 FillIDs, [9](#)
 mark_units, [9](#)
 num_units, [12](#)
 recycle_mineral, [12](#)
 recycle_waste, [12](#)
 ResetFlowrates, [9](#)
 ReturnFlowrates, [10](#)
 ReturnProfit, [10](#)
 SetInlet, [10](#)
 SolveCCircuit, [11](#)
 system_mineral_input, [12](#)
 system_waste_input, [12](#)
 SystemFlowrates, [11](#)
 units, [12](#)
CCircuit_vector
 CCircuit, [12](#)
Cgc
 CUnit, [18](#)
Cgi
 CUnit, [18](#)
Cgt
 CUnit, [18](#)
Check_Validity
 CCircuit, [8](#)
CheckMassBalance
 CUnit, [15](#)
Circuit_Parameters, [13](#)
 max_iterations, [13](#)
 tolerance, [13](#)
conc_num
 CUnit, [18](#)
ConcentrationCalc
 CUnit, [15](#)
Crossover
 Genetic_Algorithm.cpp, [35](#)
 Genetic_Algorithm.h, [26](#)
crossover_rate
 Algorithm_Parameters, [5](#)
crossover_type_rate
 Algorithm_Parameters, [5](#)
CSimulator.cpp
 default_circuit_parameters, [32](#)
 Evaluate_Circuit, [32](#)
CSimulator.h
 Evaluate_Circuit, [23](#), [24](#)
CUnit, [14](#)
 Cgc, [18](#)
 Cgi, [18](#)
 Cgt, [18](#)
 CheckMassBalance, [15](#)
 conc_num, [18](#)
 ConcentrationCalc, [15](#)
 CUnit, [15](#)
 Cwc, [18](#)
 Cwi, [18](#)
 Cwt, [18](#)
 Fgc, [18](#)
 Fgi, [19](#)
 Fgt, [19](#)
 Ftc, [19](#)
 Fti, [19](#)
 Ftt, [19](#)
 Fwc, [19](#)
 Fwi, [19](#)
 Fwt, [19](#)
 ID_num, [20](#)
 ID_num_conc, [20](#)
 ID_num_tail, [20](#)
 mark, [20](#)
 OutletCalc, [16](#)
 PrintCUnit, [16](#)
 ReactionComp, [16](#)
 ResidenceTime, [16](#)
 ReturnOutlets, [17](#)
 Rg, [20](#)
 Rw, [20](#)
 SolveCUnit, [17](#)
 tails_num, [20](#)
 tau, [20](#)

- CUnit.cpp
 - isClose, 33
- Cwc
 - CUnit, 18
- Cwi
 - CUnit, 18
- Cwt
 - CUnit, 18
- default_circuit_parameters
 - CSimulator.cpp, 32
- early_stopping_generations
 - Algorithm_Parameters, 6
- Evaluate_Circuit
 - CSimulator.cpp, 32
 - CSimulator.h, 23, 24
- Evaluate_circuit
 - Genetic_Algorithm.cpp, 35
 - Genetic_Algorithm.h, 26
- Fgc
 - CUnit, 18
- Fgi
 - CUnit, 19
- Fgt
 - CUnit, 19
- FillIDs
 - CCircuit, 9
- fitness_val
 - Individual, 21
- Ftc
 - CUnit, 19
- Fti
 - CUnit, 19
- Ftt
 - CUnit, 19
- Fwc
 - CUnit, 19
- Fwi
 - CUnit, 19
- Fwt
 - CUnit, 19
- generateRandomParameters
 - HyperPSearch.cpp, 39
 - HyperPSearch.h, 30
- generation_step
 - Algorithm_Parameters, 6
- Genetic_Algorithm.cpp
 - AddChildren, 34
 - CalculateFitness, 35
 - Crossover, 35
 - Evaluate_circuit, 35
 - InitializePopulation, 36
 - MakeCDF, 36
 - Mutation, 37
 - NormalizeFitness, 37
 - optimize, 37
 - SelectParents, 38
 - SortParentsByFitness, 38
- Genetic_Algorithm.h
 - AddChildren, 25
 - CalculateFitness, 26
 - Crossover, 26
 - Evaluate_circuit, 26
 - InitializePopulation, 27
 - MakeCDF, 27
 - Mutation, 28
 - NormalizeFitness, 28
 - optimize, 29
 - SelectParents, 29
 - SortParentsByFitness, 29
- HyperPSearch.cpp
 - generateRandomParameters, 39
 - randomSearch, 40
- HyperPSearch.h
 - generateRandomParameters, 30
 - randomSearch, 31
- ID_num
 - CUnit, 20
- ID_num_conc
 - CUnit, 20
- ID_num_tail
 - CUnit, 20
- include/CCircuit.h, 23
- include/CSimulator.h, 23
- include/CUnit.h, 24
- include/Genetic_Algorithm.h, 24
- include/HyperPSearch.h, 30
- Individual, 21
 - fitness_val, 21
 - vector, 21
- InitializePopulation
 - Genetic_Algorithm.cpp, 36
 - Genetic_Algorithm.h, 27
- isClose
 - CUnit.cpp, 33
- main
 - main.cpp, 40
- main.cpp
 - main, 40
- MakeCDF
 - Genetic_Algorithm.cpp, 36
 - Genetic_Algorithm.h, 27
- mark
 - CUnit, 20
- mark_units
 - CCircuit, 9
- max_iterations
 - Circuit_Parameters, 13
- Mutation
 - Genetic_Algorithm.cpp, 37
 - Genetic_Algorithm.h, 28
- mutation_max_step

- Algorithm_Parameters, 6
- mutation_rate
 - Algorithm_Parameters, 6
- mutation_type_rate
 - Algorithm_Parameters, 6
- NormalizeFitness
 - Genetic_Algorithm.cpp, 37
 - Genetic_Algorithm.h, 28
- num_units
 - CCircuit, 12
- optimize
 - Genetic_Algorithm.cpp, 37
 - Genetic_Algorithm.h, 29
- OutletCalc
 - CUnit, 16
- population_size
 - Algorithm_Parameters, 6
- PrintCUnit
 - CUnit, 16
- randomSearch
 - HyperPSearch.cpp, 40
 - HyperPSearch.h, 31
- ReactionComp
 - CUnit, 16
- recycle_mineral
 - CCircuit, 12
- recycle_waste
 - CCircuit, 12
- ResetFlowrates
 - CCircuit, 9
- ResidenceTime
 - CUnit, 16
- ReturnFlowrates
 - CCircuit, 10
- ReturnOutlets
 - CUnit, 17
- ReturnProfit
 - CCircuit, 10
- Rg
 - CUnit, 20
- Rw
 - CUnit, 20
- SelectParents
 - Genetic_Algorithm.cpp, 38
 - Genetic_Algorithm.h, 29
- SetInlet
 - CCircuit, 10
- SolveCCircuit
 - CCircuit, 11
- SolveCUnit
 - CUnit, 17
- SortParentsByFitness
 - Genetic_Algorithm.cpp, 38
 - Genetic_Algorithm.h, 29
- src/CCircuit.cpp, 31
- src/CMakeLists.txt, 31
- src/CSimulator.cpp, 31
- src/CUnit.cpp, 32
- src/Genetic_Algorithm.cpp, 33
- src/HyperPSearch.cpp, 39
- src/main.cpp, 40
- system_mineral_input
 - CCircuit, 12
- system_waste_input
 - CCircuit, 12
- SystemFlowrates
 - CCircuit, 11
- tails_num
 - CUnit, 20
- tau
 - CUnit, 20
- tolerance
 - Circuit_Parameters, 13
- units
 - CCircuit, 12
- vector
 - Individual, 21