Flourite Gerardium Rush

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 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 ()
- CCircuit (int num_units)

  Constructor of the CCircuit class that initialises the vector of CUnit units.
- 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_Validity (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

*num_units_*	The number of units in the circuit.
*CCircuit_↩ vector_*	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

int	∗ 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

CCircuit_vector	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

int	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	*Flowrates_g*	A vector that will be filled with the mineral flow rates of each unit.
out	*Flowrates_w*	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: ProfitGerardium $*$ Fgi - PenalizationWaste $*$ 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 (Fgi and Fwi) as well as the total input flow rate (Fti) 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↩ _	The mineral input flow rate to be set for the system.
Sw↩ _	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

vector_size	the size of the circuit vector
circuit_vector	the pointer of the circuit vector array
parameters	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 &parameters)

  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 &parameters, 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 &parameters, 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

children	A vector of individuals that will be updated with valid children.
childX	The first child to be added if it's valid.
childY	The second child to be added if it's valid.
circuit	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

parents	A vector of individuals for which to calculate fitness.
func	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

selected_parentX	The first parent.
selected_parentY	The second parent.
gen	The random number generator.
uniform_real	A uniform real distribution.
parameters	The parameters of the genetic algorithm.
vector_size	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

vector_size	The size of the circuit vector.
circuit_vector	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

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

normalized	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,
            Algorithm_Parameters & parameters,
            int vector_size )
```

Perform mutation on two individuals.

Parameters

childX	The first individual.
childY	The second individual.
gen	The random number generator.
uniform_real	A uniform real distribution.
uniform_int	A uniform integer distribution.
uniform_mutation_step	A uniform real distribution used to generate mutation steps.
parameters	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

parents	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

vector_size	The size of the vectors representing individuals in the population.
func	A function that calculates fitness.
parameters	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

parents	A vector of individuals from which to select parents.
cdf	A pointer to the cumulative distribution function.
gen	The random number generator.
uniform_real	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

parents	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

vector_size	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

iterCount	The number of iterations to perform.
best_vector	A pointer to an array that will hold the best vector.
vector_size	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

vector_size	the size of the circuit vector
circuit_vector	the pointer of the circuit vector array
parameters	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

a	The first double value to compare.
b	The second double value to compare.
tolerance	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 &parameters)

  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 &parameters, 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 &parameters, 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

children	A vector of individuals that will be updated with valid children.
childX	The first child to be added if it's valid.
childY	The second child to be added if it's valid.
circuit	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

parents	A vector of individuals for which to calculate fitness.
func	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

selected_parentX	The first parent.
selected_parentY	The second parent.
gen	The random number generator.
uniform_real	A uniform real distribution.
parameters	The parameters of the genetic algorithm.
vector_size	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

vector_size	The size of the circuit vector.
circuit_vector	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,
            Algorithm_Parameters & parameters )
```

Initialize a population for the genetic algorithm.

Parameters

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

normalized	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

childX	The first individual.
childY	The second individual.
gen	The random number generator.
uniform_real	A uniform real distribution.
uniform_int	A uniform integer distribution.
uniform_mutation_step	A uniform real distribution used to generate mutation steps.
parameters	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

parents	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

vector_size	The size of the vectors representing individuals in the population.
func	A function that calculates fitness.
parameters	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

parents	A vector of individuals from which to select parents.
cdf	A pointer to the cumulative distribution function.
gen	The random number generator.
uniform_real	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

parents	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

vector_size	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

iterCount	The number of iterations to perform.
best_vector	A pointer to an array that will hold the best vector.
vector_size	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