Number Partition Problem Genetic Algorithm

Generated by Doxygen 1.8.12

Contents

	Null	iberPar	litionGA				ı
2	Clas	s Index					3
	2.1	Class	List		 	 	3
3	File	Index					5
	3.1	File Lis	st		 	 	5
4	Clas	s Docu	mentation	n			7
	4.1	GA::fir	alSolution	Struct Struct Reference	 	 	7
		4.1.1	Member	Data Documentation	 	 	7
			4.1.1.1	binarySolution	 	 	7
			4.1.1.2	fitness	 	 	7
			4.1.1.3	leftPartition	 	 	7
			4.1.1.4	rightPartition	 	 	7
	4.2	GA Cla	ass Refere	ence	 	 	8
		4.2.1	Construc	ctor & Destructor Documentation	 	 	8
			4.2.1.1	GA() [1/2]	 	 	8
			4.2.1.2	GA() [2/2]	 	 	8
			4.2.1.3	~GA()	 	 	9
		4.2.2	Member	Function Documentation	 	 	9
			4.2.2.1	displayPopulation()	 	 	9
			4.2.2.2	getFitness()	 	 	9
			4.2.2.3	getSortedList()	 	 	9
			4.2.2.4	run()	 	 	9

ii CONTENTS

			4.2.2.5	setBestSolution()	10
			4.2.2.6	tournament()	10
		4.2.3	Member	Data Documentation	10
			4.2.3.1	finalSolution	10
	4.3	Greedy	/ Class Re	ference	11
		4.3.1	Construc	tor & Destructor Documentation	11
			4.3.1.1	Greedy()	11
			4.3.1.2	~Greedy()	11
		4.3.2	Member	Function Documentation	11
			4.3.2.1	getFitness()	11
			4.3.2.2	run()	11
5	File	Docume	entation		13
	5.1	F:/Goo	ale Drive/0	CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/GA.cpp File Reference	13
	5.2		_	CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/GA.h File Reference .	13
	5.3		_	CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/Greedy.cpp File Refer-	
		ence .			14
	5.4	F:/Goo	gle Drive/0	CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/Greedy.h File Reference	14
	5.5	F:/Goo GA.cpp	•	e/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/NumberPartition← rence	14
		5.5.1	Detailed	Description	15
		5.5.2	DESCRI	PTION	15
		5.5.3	Function	Documentation	16
			5.5.3.1	check()	16
			5.5.3.2	helpMenu()	16
			5.5.3.3	main()	16
	5.6	F:/Goo	gle Drive/0	CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/stdafx.cpp File Reference	16
	5.7	F:/Goo	gle Drive/0	CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/stdafx.h File Reference	16
	5.8		•	CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/targetver.h File Refer-	17
	5.9	F:/Goo	gle Drive/0	CSCI443/Project 2/NumberPartitionGA/README.md File Reference	17
Inc	dex				19

NumberPartitionGA

This is a program to run two algorithms to solve the well-known Number Partition Problem (NPP) The NPP is taking a set of numbers and partition that set. Each element in the original set may be placed in either of the two sub partitions. The goal is to find a solution where the sum of the numbers in each partition is equal. For this particular algorithm, a fitness is represented by an integer. A perfect solution will have a value of 0. As this number increases, the solution is further from a perfect solution. For a particular set, there might not be a perfect solution. A "chromosome" is the term I will use for a particular solution. A chromosome is represented as a binary string of 1's and 0's. A 1 represents a value in one of the two partitions and a 0 represents a value in the other partition. Each position of the bits corresponds to the position of the SORTED ascending original list.

These two algorithms are as follows: A greedy algorithm called the longest processing time heuristic. This algorithm first takes the largest element in the original set and places it in the "left" partition. The sum for that partition is then calculated (one value at this point). The element is then removed from the original set. The repetitive process at this point is to take the highest value element in the original set and place it in the partition with the smaller sum. This is repeated until there are no longer any elements left in the original set.

The genetic algorithm (GA) is an algorithm that takes elements from biological evolution and mimics them to alter chromosomes that have a better fitness than their respective parents. The general steps I take for this algorithm are:

- Creating a finite set of randomly generated chromosomes (initial population)
- Using 2-tournament selection to choose two parents from the population
- Use 2-point crossover to splice the chromosomes from the two parents to create an offspring chromosome
- Simulate genetic mutation by making the probability of each bit in an offspring chromosome be the opposite of what it was. This probability is much higher in this algorithm than actaul biology. This is to help speed up the process. 10% probability is a good area to work with.

After the finite number of generations are completed, the algorithm might have found multiple solutions. All of the best solutions are then shown with their respective fitness (abs(sumLeft - sumRight)), binary chromosome, and numeric partitions.

2 NumberPartitionGA

Class Index

2.1 Class List

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

GA::final	Sc	olui	lo	nS	trι	ıct				 							-			 								1
GA										 										 							- 1	ξ
Greedy										 										 							11	1

4 Class Index

File Index

3.1 File List

Here is a list of all files with brief descriptions:

F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/GA.cpp	13
F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/GA.h	13
F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/Greedy.cpp	14
F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/Greedy.h	14
F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/NumberPartitionGA.cpp	
Use a greedy algorithm and genetic algorithm to solve the number partition problem	14
F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/stdafx.cpp	16
F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/stdafx.h	16
F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/targetver.h	17

6 File Index

Class Documentation

4.1 GA::finalSolutionStruct Struct Reference

```
#include <GA.h>
```

Public Attributes

- int fitness
- std::vector< std::string > binarySolution
- std::vector< int > leftPartition
- std::vector< int > rightPartition

4.1.1 Member Data Documentation

4.1.1.1 binarySolution

std::vector<std::string> GA::finalSolutionStruct::binarySolution

4.1.1.2 fitness

int GA::finalSolutionStruct::fitness

4.1.1.3 leftPartition

std::vector<int> GA::finalSolutionStruct::leftPartition

4.1.1.4 rightPartition

std::vector<int> GA::finalSolutionStruct::rightPartition

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

• F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/GA.h

8 Class Documentation

4.2 GA Class Reference

```
#include <GA.h>
```

Collaboration diagram for GA:

Classes

· struct finalSolutionStruct

Public Member Functions

• GA ()

Default constructor for genetic algorithm. This should not be used by default.

• GA (std::vector< int > startingList, int INIT_POP_SIZE)

Constructor for genetic algorithm.

• ∼GA ()

Default deconstructor for GA class.

• void run (int itterations)

run executes the genetic algorithm.

std::vector< std::string > tournament (std::vector< std::vector< std::string >> &L, int &popSize)

tournament picks two randomly selected parents from the population. The fitness of each chromosome is compared. The one with the lowest fitness (better fitness) is the chosen and placed back into the population. The loser chromosome is then discarded from the popultion.

void displayPopulation (const std::vector< std::vector< std::string >> vector_const)

Displays to the console each chromosome and their binary representation. This can be called at any time to view this information.

- int getFitness (std::vector< std::string > chromosome)
- std::vector< int > getSortedList ()
- void setBestSolution (std::vector< std::vector< std::string >> &L)

Compares all of the chromosomes in the current population and picks the first best solution from the population. This is then stored into a data structure called final Solution.

Public Attributes

· struct GA::finalSolutionStruct finalSolution

4.2.1 Constructor & Destructor Documentation

```
4.2.1.1 GA() [1/2] GA::GA ( )
```

Default constructor for genetic algorithm. This should not be used by default.

Constructor for genetic algorithm.

4.2 GA Class Reference 9

Parameters

startingList	A vector which contains the origonal number set
popsize	<int> How many chromosomes are generated and kept in the population</int>

```
4.2.1.3 \sim GA()
```

```
GA::∼GA ( )
```

Default deconstructor for GA class.

4.2.2 Member Function Documentation

4.2.2.1 displayPopulation()

Displays to the console each chromosome and their binary representation. This can be called at any time to view this information.

Parameters

vector_const	The vector containing the population of chromosomes. This is constant because we are only	
	viewing data.	

Returns

void

4.2.2.2 getFitness()

4.2.2.3 getSortedList()

```
std::vector< int > GA::getSortedList ( )
4.2.2.4 run()
```

run executes the genetic algorithm.

10 Class Documentation

Parameters

	itterations	<int> How many time the algorithm will loop (number of generations)</int>
--	-------------	---

Returns

void

Here is the caller graph for this function:

4.2.2.5 setBestSolution()

Compares all of the chromosomes in the current population and picks the first best solution from the population. This is then stored into a data structure called final Solution.

Parameters

L a vector containing the current population of chromosomes

Returns

void

4.2.2.6 tournament()

tournament picks two randomly selected parents from the population. The fitness of each chromosome is compared. The one with the lowest fitness (better fitness) is the chosen and placed back into the population. The loser chromosome is then discarded from the popultion.

Parameters

L	a vector containing the current chromosome population
popSize	current population size vector <string> Returns the winner chromosome from the tournament selection</string>

4.2.3 Member Data Documentation

4.2.3.1 finalSolution

```
struct GA::finalSolutionStruct GA::finalSolution
```

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

- F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/GA.h
- F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/GA.cpp

4.3 Greedy Class Reference

```
#include <Greedy.h>
```

Public Member Functions

- Greedy ()
- \sim Greedy ()

Default deconstructor for Greedy.

- void run (std::vector< int > &L, bool displayTwoLists)

Starts the main algorithm for the greedy algorithm.

• int getFitness ()

4.3.1 Constructor & Destructor Documentation

```
4.3.1.1 Greedy()
```

```
Greedy::Greedy ( )
```

Default constructor for Greedy.

```
4.3.1.2 \sim Greedy()
```

```
Greedy::\simGreedy ( )
```

Default deconstructor for Greedy.

4.3.2 Member Function Documentation

```
4.3.2.1 getFitness()
```

```
int Greedy::getFitness ( ) [inline]
```

Here is the caller graph for this function:

```
4.3.2.2 run()
```

```
void Greedy::run (
          std::vector< int > & L,
          bool displayTwoLists )
```

Starts the main algorithm for the greedy algorithm.

12 Class Documentation

Parameters

L	is a vector containing the origonal number set	l
displayTwoLists	<bool> Displays extra information about the final results</bool>	

Returns

void

Here is the caller graph for this function:

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

- F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/Greedy.h
- F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/Greedy.cpp

File Documentation

5.1 F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/GA.cpp File Reference

```
#include "stdafx.h"
#include "GA.h"
#include <list>
#include <time.h>
#include <iostream>
#include <string>
#include <vector>
#include <functional>
#include <algorithm>
#include <random>
Include dependency graph for GA.cpp:
```

5.2 F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/GA.h File Reference

```
#include <vector>
```

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

Classes

- class GA
- struct GA::finalSolutionStruct

14 File Documentation

5.3 F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/Greedy.cpp File Reference

```
#include "stdafx.h"
#include "Greedy.h"
#include <list>
#include <iostream>
#include <string>
#include <functional>
#include <vector>
#include <algorithm>
```

Include dependency graph for Greedy.cpp:

5.4 F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/Greedy.h File Reference

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

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

Classes

· class Greedy

5.5 F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/Number PartitionGA.cpp File Reference

Use a greedy algorithm and genetic algorithm to solve the number partition problem.

```
#include "stdafx.h"
#include <iostream>
#include <list>
#include <time.h>
#include <string>
#include <algorithm>
#include "Greedy.h"
#include "GA.h"
#include <vector>
#include <conio.h>
```

Include dependency graph for NumberPartitionGA.cpp:

Functions

· void check ()

Keeps the console from closing after main has finished.

• void helpMenu ()

Displays the help menu to the console.

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

5.5.1 Detailed Description

Use a greedy algorithm and genetic algorithm to solve the number partition problem.

Author

Steven Scholz

Date

9/21/17

Version

1.0

5.5.2 DESCRIPTION

This is a program to run two algorithms to solve the well-known Number Partition Problem (NPP) The NPP is taking a set of numbers and partition that set. Each element in the original set may be placed in either of the two sub partitions. The goal is to find a solution where the sum of the numbers in each partition is equal. For this particular algorithm, a fitness is represented by an integer. A perfect solution will have a value of 0. As this number increases, the solution is further from a perfect solution. For a particular set, there might not be a perfect solution. A "chromosome" is the term I will use for a particular solution. A chromosome is represented as a binary string of 1's and 0's. A 1 represents a value in one of the two partitions and a 0 represents a value in the other partition. Each position of the bits corresponds to the position of the SORTED ascending original list.

These two algorithms are as follows: A greedy algorithm called the longest processing time heuristic. This algorithm first takes the largest element in the original set and places it in the "left" partition. The sum for that partition is then calculated (one value at this point). The element is then removed from the original set. The repetitive process at this point is to take the highest value element in the original set and place it in the partition with the smaller sum. This is repeated until there are no longer any elements left in the original set.

The genetic algorithm (GA) is an algorithm that takes elements from biological evolution and mimics them to alter chromosomes that have a better fitness than their respective parents. The general steps I take for this algorithm are:

- · Creating a finite set of randomly generated chromosomes (initial population)
- Using 2-tournament selection to choose two parents from the population
- · Use 2-point crossover to splice the chromosomes from the two parents to create an offspring chromosome
- Simulate genetic mutation by making the probability of each bit in an offspring chromosome be the opposite of what it was. This probability is much higher in this algorithm than actaul biology. This is to help speed up the process. 10% probability is a good area to work with.

After the finite number of generations are completed, the algorithm might have found multiple solutions. All of the best solutions are then shown with their respective fitness (abs(sumLeft - sumRight)), binary chromosome, and numeric partitions.

You can use the argument -h to call the help menu for the different command line arguments

16 File Documentation

5.5.3 Function Documentation

```
5.5.3.1 check()

void check ( )
```

Keeps the console from closing after main has finished.

Returns

void

Here is the caller graph for this function:

```
5.5.3.2 helpMenu()
```

```
void helpMenu ( )
```

Displays the help menu to the console.

Returns

void

Here is the caller graph for this function:

```
5.5.3.3 main()
```

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

Here is the call graph for this function:

5.6 F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/stdafx.cpp File Reference

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

5.7 F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/stdafx.h File Reference

```
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
```

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

5.8	F:/Google Drive/CSCI443/Project 2/NumberPartitionGA/NumberPartitionGA/targetver.h
	File Reference

Fi	ile Reference
	de <sdkddkver.h> ependency graph for targetver.h: This graph shows which files directly or indirectly include this file:</sdkddkver.h>
5.9 F:	:/Google Drive/CSCI443/Project 2/NumberPartitionGA/README.md File Reference

18 File Documentation

Index

\sim GA	setBestSolution, 10
GA, 9	tournament, 10
\sim Greedy	getFitness
Greedy, 11	GA, 9
	Greedy, 11
binarySolution	getSortedList
GA::finalSolutionStruct, 7	GA, 9
	Greedy, 11
check	~Greedy, 11
NumberPartitionGA.cpp, 16	getFitness, 11
	Greedy, 11
displayPopulation	•
GA, 9	run, 11
F:/Google Drive/CSCI443/Project 2/NumberPartitionG	helpMenu
A/NumberPartitionGA/GA.cpp, 13	NumberPartitionGA.cpp, 16
F:/Google Drive/CSCI443/Project 2/NumberPartitionG←	
- · · · · · · · · · · · · · · · · · · ·	leftPartition
A/NumberPartitionGA/GA.h, 13	GA::finalSolutionStruct, 7
F:/Google Drive/CSCI443/Project 2/NumberPartitionG←	ŕ
A/NumberPartitionGA/Greedy.cpp, 14	main
F:/Google Drive/CSCI443/Project 2/NumberPartitionG←	NumberPartitionGA.cpp, 16
A/NumberPartitionGA/Greedy.h, 14	117
F:/Google Drive/CSCI443/Project 2/NumberPartition ←	NumberPartitionGA.cpp
GA/NumberPartitionGA/NumberPartitionG	check, 16
A.cpp, 14	helpMenu, 16
F:/Google Drive/CSCI443/Project 2/NumberPartitionG←	main, 16
A/NumberPartitionGA/stdafx.cpp, 16	,
F:/Google Drive/CSCI443/Project 2/NumberPartitionG←	rightPartition
A/NumberPartitionGA/stdafx.h, 16	GA::finalSolutionStruct, 7
F:/Google Drive/CSCI443/Project 2/NumberPartitionG←	run
A/NumberPartitionGA/targetver.h, 17	GA, 9
F:/Google Drive/CSCI443/Project 2/NumberPartitionG←	Greedy, 11
A/README.md, 17	Cirectly, 11
finalSolution	setBestSolution
	GA, 10
GA, 10	CA, 10
fitness	tournament
GA::finalSolutionStruct, 7	GA, 10
GA::finalSolutionStruct, 7	art, 10
binarySolution, 7	
fitness, 7	
•	
leftPartition, 7	
rightPartition, 7	
GA, 8	
~GA, 9	
displayPopulation, 9	
finalSolution, 10	
GA, 8	
getFitness, 9	
getSortedList, 9	
run, 9	