Ancestry and Relations

Abstract

In this paper, common trends amongst ancestry lines are analyzed and observed. The goal of the paper is to show how common ancestry forms in subsequent generations of offspring. The methods used include unique and non-unique testing. The affect of population growth on common ancestry is also observed using these methods.

Project 3
Mathematics 2130
Submitted by: John Hollett
Submitted to: Ivan Booth
November 13, 2015

1 Introduction

In this paper, how ancestry changes after each subsequent generation is discussed. Generations refers to individual sets of population. The tests range from basic conditions to more selective processes. These processes include random sampling with and without unique pairs. A pair refers to the duality between ancestors whether uniquely selected or not.

In non-unique tests, the participants of each generation may be assigned the same parent. In unique testing, each participant contains two different parents. The difference between these tests will be explored and compared.

A program was written in C$^{\#}$ for tracing and tracking ancestry. This utility is required to perform the tests. The following equation describes the approximate number of generations required to see a common ancestor. This simple simulation is tasked with the responsibility of networking relationships in a decipherable manner. In the book by Richard Dawkins [1], equation (1) was derived.

$$C_{Generation} = \log_2(N_{Population}) \tag{1}$$

2 Non-Unique Testing

2.1 Initial Test

This section includes sampling and analysis of non-unique ancestors of varying sizes for each generation. The figure (1) depicts the results of a test. An initial population of ten is selected, and a maximum of ten generations of humans. Each test is iterated 250 times.

In Figure (1), the axes described is consistent with all figures in the paper. The information shown is the number of common ancestors for each generation starting from the founding generation. It serves as a proof in determining the program works correctly. A smaller size test was created and carried out to satisfy this requirement.

$$C_{Generation} = \log_2(10) = 3.32$$

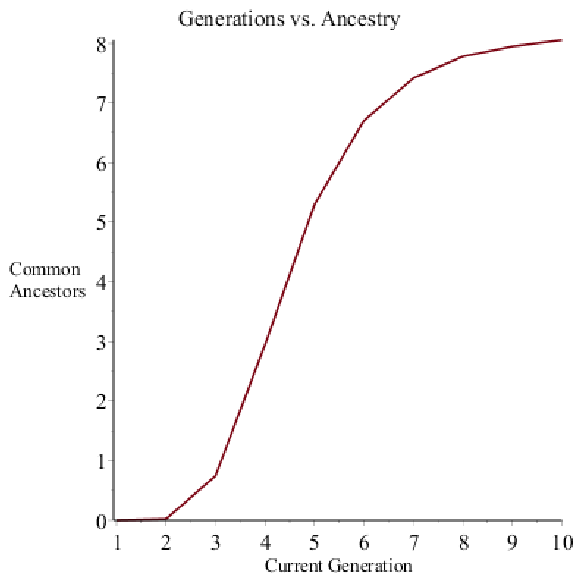
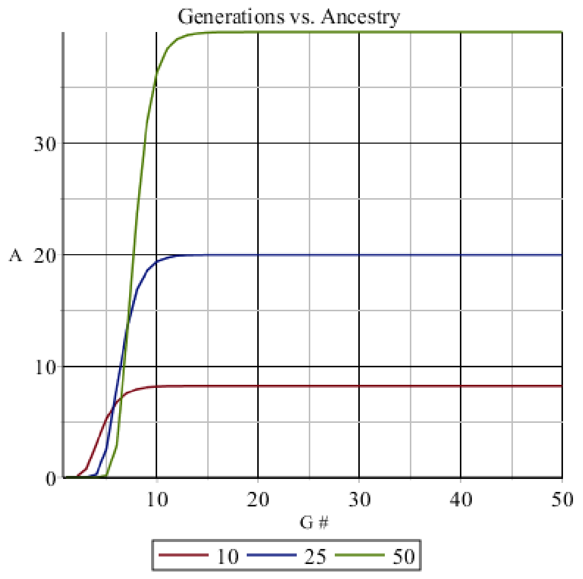


Figure 1: Founding Population = 10, Generations = 10, Non-unique

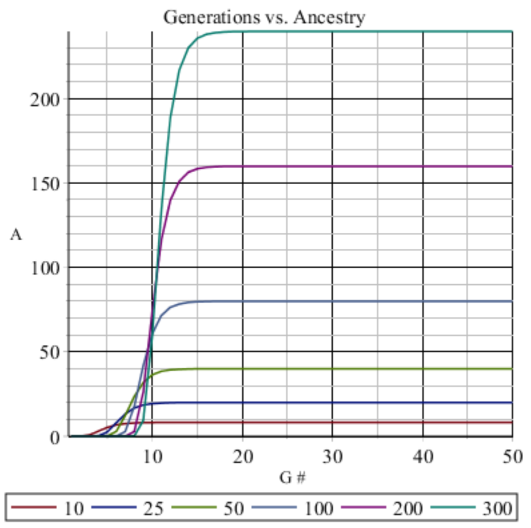
With the data depicted by figure (1), it is possible for one common ancestor to emerge after the second generation. Using equation (1), the result supports the graph. It shows the utility created performs sufficiently to produce proper data.

2.2 Further Testing

Figure 2: Non-unique Test



(a) Test Pop 10, 25, 50



(b) Test Pop 10, 25, 50, 100, 200, 300

In this section, each test is compared using varying initial population sizes. The number of maximum generations remains constant at fifty. Based on Figure (2a), trends can be observed from the changing initial size. Each test show common ancestry emerges only after the third generation.

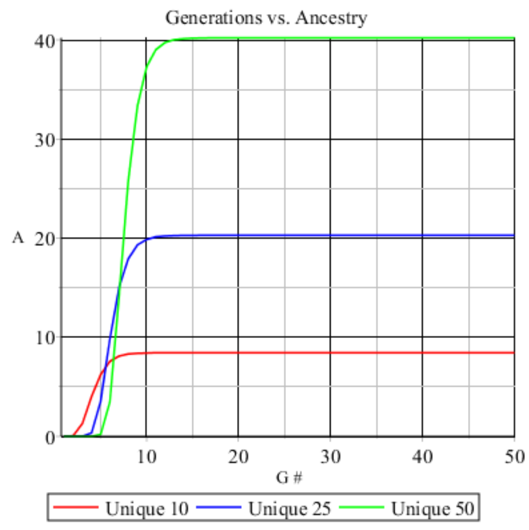
The tests using lower initial population show that the number of generations required to produce a non-changing common ancestor set changes from the initial population. This is also seen in larger founding population sizes. In Figure (2b), the data contradict perceived consistency in the number of generations required to observe a consistent ancestor set across all subsequent generations. Despite these results, the simulation returns the average number of generations it takes to form a common ancestry set for each founding population.

The average of the number of ancestors carried through subsequent generations depicted in the figures (2a) and (2b) is roughly 80% of the starting population. Inversely, 20% of the founding population drops from the ancestry tree outright. The data depicted in the next section addresses any influences uniquely selected ancestors exhibit.

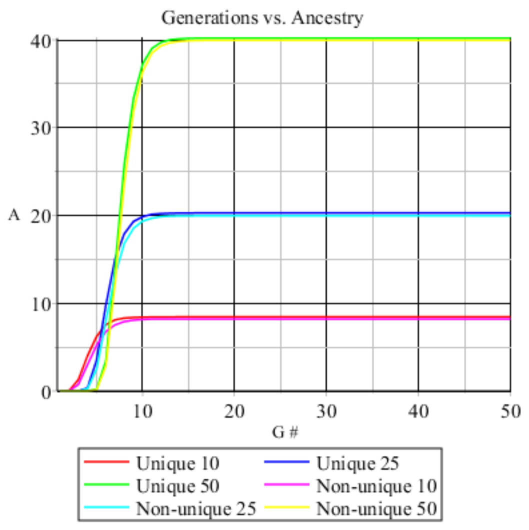
3 Unique Testing

The data prepared in the previous section contrast the differences of unique and non-unique tests. Tests were re-ran for populations of $10, 25, 50$ and $100, 200, 300$. The number of maximum generations for which the tests were run remains fifty for this section. Figure (3a) and (3b) shows the average number of common ancestors in each generation. Using unique selection, the analysis reveals similar results. Checking the results with equation (1) for each initial population supports figures (3a) and (3b).

Figure 3: Unique and Non-Unique Analysis



(a) Unique



(b) Unique and Non-unique

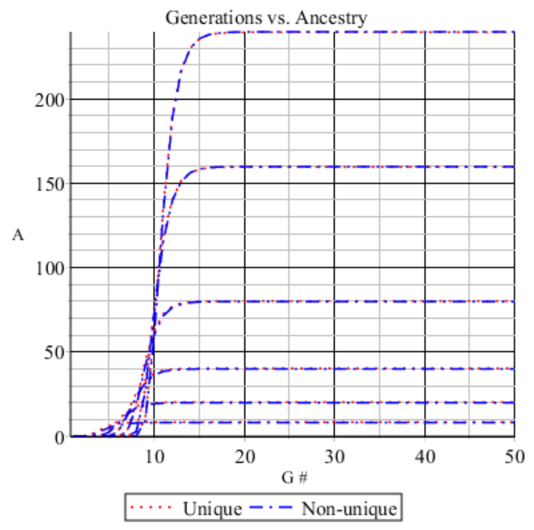


Figure 4: Unique vs. non-unique

$$C_{Generation} = \log_2(50) = 5.64$$

Figure (3b) depicts a unique ancestor set formed related to the founding size as seen in the previous section. This indicates that the difference between the two methods has negligible affect on the data.

Figure (4) shows increasing initial populations. The increase in population in subsequent generations is a constant rate of N. The results of unique and non-unique tests show there is no significant difference between the two. The unique test results, on average, produce negligibly greater numbers of ancestors. Meanwhile the number of generations required to find an ancestor is greater although it is insignificant.

4 Population Growth

The tests carried out in this section simulates population growth. Growth is determined by a percentage that is constant amongst each generation. The population of any subsequent generation is determined by applying the percentage to the prior generation.

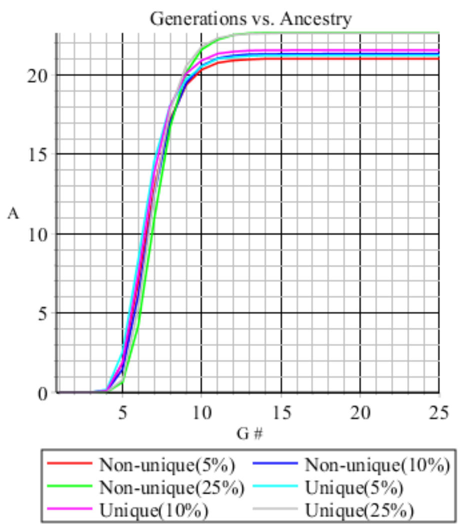


Figure 5: Pop Growth 5%, 10% ,25%

For the tests completed in this section, the populations will grow in size for each generation. The initial population for each test remains constant while the percentages of growth vary.

In figure (5), despite population growth, the average number of ancestors remains relatively consistent in all tests. A significant difference is seen when the common ancestry set is found.

With increasing population growth, the number of common ancestors increases. This result reflects the improved probability of selecting, and thus creating, more common ancestors from the founding set.

$$C_{\text{Generation}} = \log_2(25) = 4.64$$

The first common ancestor occurs at approximately the 5th generation. Following the 5th generation, the common ancestor set expands until the complete set is found. Using equation (1), the result above agrees with figure (5).

5 Technical Details

Explanation of the details of the testing methods is outlined in this section. The data collected were created using a simulation. The C$^{\#}$ programming language was used to create an iterative function to simulate data.

Two programmed objects named "Human" and "Generation" were used to correlate coherent data. A set of "Human" elements is stored in a generation. Previous sets are accessible in each "Generation", and the newly created "Generation" replaces the last as the current set. This is explained using sequences in mathematics . A sequence defined using the initial element A_1 and subsequent elements are described relative to the first in the sequence. The following example describes this process:

$$A_1 = \text{Generation}_0$$
$$A_N = \text{Generation}_{N-1}, \text{Where } N \geq 2$$
$$\text{Where Generation}_I \text{ is the population set at index I}$$

A "Human" is defined as an object containing two ancestors. For subsequent generations, each "Human" contains ancestry inherited from its parents derived from the original population set. This process is described as:

$$\begin{aligned}
&P = \{p_0..p_N\}, \text{Where P denotes original ancestry set}\\
&A = \{a \in P | a_0, a_1, ..., a_N\}\\
&B = \{b \in P | b_0, b_1, ..., b_N\}\\
&C = A \cup B, \text{Where C is the ancestry set of the current Human}
\end{aligned} \tag{2}$$

Testing ancestry in the current generation, the simulation processes the elements in the set. Each "Human" and its ancestry set is intersected. If the result created is not an empty set, there is at least one common ancestor in the current generation.

New generations are created using parameters describing the number of new elements, previous generation and uniqueness. Creating a new population requires the prior generation. The two methods for creating each "Human" in the population is determined by reading the uniqueness parameter. A non-unique "Human" is created by selecting two random parents. Unique describes the same process as non-unique while requiring the parents to be unequal.

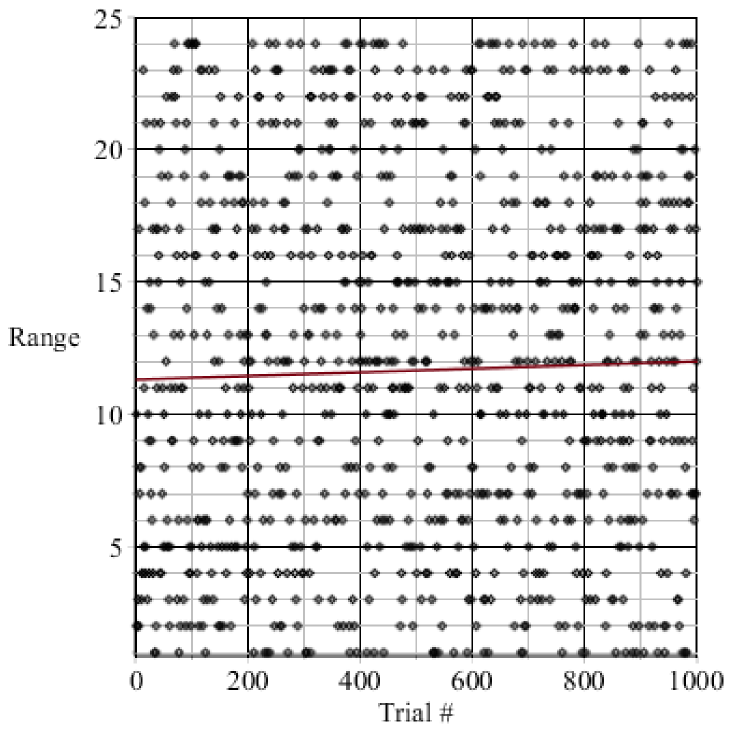


Figure 6: Random Distribution

Figure (6) depicts the method used to randomly select a parent from a population of 25. While there is no definitive trend in this data, the line of best indicates gradual error. As the number of trials approaches completion, the trend line rests in the middle of the range. This suggests that most of the error is eliminated through increased testing. The error related to by the method used for selecting a parent is negligible but, other studies use more reliable methods.

In a population set containing 25 elements, the randomly selected values lies between 0 and 24. The maximum value in figure (6) is 24. A set of data containing 25 elements in the language chosen begins at 0 and ends at 24. The actual range is $0, ..., 24$ inclusive, resulting in 25 elements.

6 Conclusion

The tests performed in this paper show common ancestry occurs consistently after the number of generations found by equation (1) . The figures show that the initial population affects presence of a common ancestor. Following the observation of a common ancestor, the figures depict consistently increasing traces of the founding members in each ”Human”.

In a paper on ancestry written by Douglas Rhode [2], the results of tests were far more advanced and complex than those used in this test. The paper takes into account the various levels of segregation in the population, such as migration and life span.

While this paper does not detail such in depth analysis, real world application of the results does merit a meaningful conclusion. It provided evidence that common ancestry occurs and it should not surprise the reader that he/she may be distantly related to the writer and beyond.

7 C# Code

Listing 1: Ancestry.cs

```
 using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

namespace Math2130Project3 {

  public class Generation {
    public int CurrentGeneration { get; private set; }
    public int Number { get; private set; }
    public Generation Previous { get; private set; }
    public Human[] Population { get; private set; }

    public Generation( Generation p , int N , bool unique ) {
      this.CurrentGeneration = 1;
      this.Previous = p;
      this.Number = N;
      this.Population = new Human[N];
      if ( Previous != null ) {
        this.CurrentGeneration = this.Previous.CurrentGeneration + 1;
        if ( !unique ) {
          for ( int i = 0; i < N; i++ ) {
            this.Population[i] = new Human( Previous.getRandom() ,
              Previous.getRandom() , this , i );
          }
        } else {
          for ( int i = 0; i < N; i++ ) {
            this.Population[i] = new Human( Previous.Population , this ,
              i );
          }
        }
      } else {
        for ( int i = 0; i < N; i++ ) {
          this.Population[i] = new Human( null , null , this , i );
        }
      }
    }

    public static int AncestorDifference( Generation origin , Generation
      current ) {
      List<Human> ancestors = origin.Population.ToList();
      List<Human> currentAncestors = current.UniqueAncestors();
      return ancestors.Intersect( currentAncestors ).ToList().Count;
    }

    public int TotalUnique() {
      return this.UniqueAncestors().Count;
    }

    public Human getRandom() {
            Random rndNum = new Random(int.Parse(Guid.NewGuid().ToString().
                Substring(0, 8), System.Globalization.NumberStyles.HexNumber));

            int rnd = rndNum.Next(0, Number);
            return Population[rnd];
        }

        public List<Human> UniqueAncestors() {
            List<Human> list = new List<Human>();
            bool start = true;
            foreach ( Human h in this.Population ) {
                if ( list.Count == 0 & start ) {
                    list = h.Ancestors;
                    start = false;
                } else {
                    list = list.Intersect( h.Ancestors ).ToList();
                }
            }
            return list;
        }

        public override string ToString() {
            string x = "";
            foreach ( Human h in Population ) {
                x += "(" + this.CurrentGeneration + "," + h.ID + ")";
            }
            x = x.Replace( ")(" , "), (" );
            return "[" + x + "]";
        }
    }

    public class Human {
        public int ID;
        public Generation Current { get; private set; }
        public Human[] Ancestor;
        public List<Human> Ancestors { get; private set; }
        public Human this[int i] { get { return Ancestor[i]; } }

        public Human( Human a , Human b , Generation g , int ID ) {
            this.ID = ID;
            if ( a == null || b == null ) {
                this.Ancestor = null;
                this.Ancestors = new List<Human>();
            } else if ( a.Ancestor == null && b.Ancestor == null ) {
                this.Ancestors = new List<Human>();
                this.Ancestor = new Human[] { a , b };
                this.Ancestors = this.Ancestor.ToList();
            } else {
                this.Ancestors = new List<Human>();
                this.Ancestor = new Human[] { a , b };
                this.Ancestors = a.Ancestors.Union( b.Ancestors ).ToList();
            }
            this.Current = g;
        }
        public Human( Human[] population , Generation g , int ID ) {
            this.ID = ID;
            Human a = g.Previous.getRandom();
            Human b = a;
            Human unique = null;
            while ( a.Equals( b ) ) {
                if ( !( unique = g.Previous.getRandom() ).Equals( a ) ) {
                    b = unique;
                }
            }
            if ( a == null || b == null ) {
                this.Ancestor = null;
                this.Ancestors = new List<Human>();
            } else if ( a.Ancestor == null && b.Ancestor == null ) {
                this.Ancestors = new List<Human>();
                this.Ancestor = new Human[] { a , b };
                this.Ancestors = this.Ancestor.ToList();
            } else {
                this.Ancestors = new List<Human>();
                this.Ancestor = new Human[] { a , b };

                this.Ancestors = a.Ancestors.Union( b.Ancestors ).ToList();
            }
            this.Current = g;
        }

        public override string ToString() {
            if ( this.Ancestor != null && this.Ancestor != null ) {
                return "[" + "(" + this.Current.CurrentGeneration + "," + this.ID
                    + ")=>(" + this.Ancestor[0].Current.CurrentGeneration + "," +
                    this.Ancestor[0].ID + "), (" + this.Ancestor[1].Current.
                    CurrentGeneration + "," + this.Ancestor[1].ID + ")]";
            } else {
                return "[" + "(" + this.Current.CurrentGeneration + "," + this.ID
                    + ")]";
            }
        }
    }

    public class Tester {

        public static void Basic() {
            Generation g = new Generation(null, 5 ,false);
            g = new Generation( g , 5 , false );
            g = new Generation( g , 5 , false );
            g = new Generation( g , 5 , false );
            g = new Generation( g , 5 , false );
            g = new Generation( g , 5 , false );
            g = new Generation( g , 5 , false );
            Human[] humans = g.UniqueAncestors().ToArray();
            foreach ( Human h in humans ) {
                Console.Write( "|" + h + "|" );
                Console.WriteLine();
            }
        }

        public static double[] Iterative( int n , int gens , double percent ,
            bool unique ) {
            double[] total = new double[gens];
            Generation g = new Generation(null, n, unique);
            Generation origin = g;
            if ( n > 0 ) {
                g = new Generation( g , n = ( int )( Math.Ceiling( n * ( 1 +
                    percent ) ) ) , unique );
                for ( int i = 1; i < gens; i++ ) {
                    total[i - 1] = g.TotalUnique();
                    g = new Generation( g , n = ( int )( Math.Ceiling( n * ( 1 +
                        percent ) ) ) , unique );
                }
                total[gens - 1] = g.TotalUnique();
            }
            return total;
        }

        public static void Test( int N , int InitialPop , double percent , int G
            , bool unique ) {
            double[] avg = new double[G];
            for ( int i = 0; i < N; i++ ) {
                double[] result = Iterative( InitialPop , G , percent , unique );
                avg = avg.Zip( result , ( x , y ) => x + y / N ).ToArray();
            }
            string path = @"N="+N+",P="+InitialPop+"@"+(percent+1)+",G="+G+",U="+
                unique+".txt";
            StreamWriter file = null;
            if ( File.Exists( path ) ) {
                File.Delete( path );
                file = File.CreateText( path );
            } else {
                file = File.CreateText( path );
            }
            for ( int i = 0; i < G; i++ ) {
                file.WriteLine( "{0:0.000} {1:0.000}" , i + 1 , avg[i] );
            };
            file.Close();
        }

        public static void ProveRandom() {
            StreamWriter file = null;
            string path = @"RandomPlot25.txt";

            if ( File.Exists( path ) ) {
                File.Delete( path );
                file = File.CreateText( path );
            } else {
                file = File.CreateText( path );
            }
            for ( int i = 1; i <= 1000; i++ ) {
                Random rndNum = new Random(int.Parse(Guid.NewGuid().ToString().
                    Substring(0, 8), System.Globalization.NumberStyles.HexNumber));
                double rnd = rndNum.Next(0,25);

                file.WriteLine( "{0} {1}" , i , rnd );
            }

            file.Close();
        }

        public static void Main( string[] args ) {
            //Test( 250 , 10 , 0 , 10 , false );
            //Test( 250 , 10 , 0 , 50 , false );
            //Test( 250 , 25 , 0 , 50 , false );
            //Test( 250 , 50 , 0 , 50 , false );
            //Test( 250 , 100 , 0 , 50 , false );
            //Test( 250 , 200 , 0 , 50 , false );
            //Test( 250 , 300 , 0 , 50 , false );
            //Test( 250 , 10 , 0 , 50 , true );
            //Test( 250 , 25 , 0 , 50 , true );
            //Test( 250 , 50 , 0 , 50 , true );
            //Test( 250 , 100 , 0 , 50 , true );
            //Test( 250 , 200 , 0 , 50 , true );
            //Test( 250 , 300 , 0 , 50 , true );
            //Test( 250 , 25 , 0.05 , 25 , false );
            //Test( 250 , 25 , 0.10 , 25 , false );
            //Test( 250 , 25 , 0.25 , 25 , false );

            //Test( 250 , 25 , 0.05 , 25 , true );
            //Test( 250 , 25 , 0.10 , 25 , true );
            //Test( 250 , 25 , 0.25 , 25 , true );
            //Test( 250 , 10 , 0.0 , 20 , false );
        }
      }
    }
```

References

[1] Dawkins, Richard. "The Ancestor's Tale". Houghton Mifflin, 2004, Print.

[2] "On the Common Ancestors of All Living Humans" `http://tedlab.mit.edu/~dr/Papers/Rohde-MRCA-two.pdf`, Douglas L. T. Rhode, Massachusetts Institute of Technology, 11 November 2003, Web 9 November 2015