



HÁSKÓLINN Í REYKJAVÍK
REYKJAVÍK UNIVERSITY



COMBINATORIAL PEDIGREE
INFERENCE FROM GENOMIC DATA
Research Report

Eiríkur Fannar Torfason¹
Jón Ingi Sveinbjörnsson¹

Computer Science Final Project

Supervisors: Jotun Hein², Dr. Rune Lyngsø²,
Dr. Bjarni Vilhjálmur Halldórsson¹
Evaluator: Dr. Yngvi Björnsson¹

School of Computer Science
Autumn 2008

1. Reykjavík University
2. University of Oxford

ABSTRACT / ÚRDRÁTTUR

A pedigree can be represented with a directed acyclic graph illustrating the ancestral relationships between multiple known individuals. In such a graph, individuals are represented by nodes and ancestral relationships are represented by directed edges (arcs) from parent to child. The nodes of the known individuals are labelled. We provide an algorithm for reconstructing pedigrees from data about pair-wise distances between labelled nodes, that is, the length of paths from a pair of known individuals to their common ancestors. We also explore a method to determine whether two pedigree graphs are isomorphic.

Hægt er að tákna skyldleika þekktra einstaklinga með stefndu neti. Í slíku neti tákna hnútar einstaklinga og örvar fjölskylduvensl frá foreldri til barns. Hnútar þekktra einstaklinga hafa merki (e. label). Við kynnum reiknirit sem endurbyggir slík net út frá upplýsingum um fjarlægðir milli merktra hnúta, þ.e. lengd stíga frá einhverju pari af þekktum einstaklingum til sameiginlegra forfeðra. Við könnum einnig aðferð til að úrskurða hvort tvö slík skyldleikanet séu einsmóta.

CONTENTS

1	Introduction	3
2	Background and Related Work	6
3	Topic Description / Motivation	9
4	Research Methods.....	10
4.1	Input data	10
4.2	Reconstruction algorithm.....	12
4.3	Pedigree Simulator	18
5	Results	22
5.1	Pedigree isomorphism	22
5.2	Optimizations.....	23
5.3	Number of solutions.....	32
5.4	Execution time	33
6	Conclusion and Future Works	36
6.1	Acknowledgements	37
7	References.....	38
8	Appendix.....	39
8.1	Number of solutions – Test results.....	39
8.2	Table reference	42
8.3	Figure reference.....	42
8.4	Chart reference.....	43

1 INTRODUCTION

Humans seem to have an inherent curiosity about their ancestry. The plethora of genealogical web sites available today supports this claim. deCODE genetics' Book of Icelanders (www.islendingabok.is) is a prime example of a web site designed to satisfy people's ancestral curiosities. Its database contains genealogical information on almost the entire population of Iceland, compiled from a variety of sources such as censuses, church records, obituaries etc. Users can see their own family trees as far as the records go and find the shortest pair-wise distance to a common ancestor between themselves and any other individual in the database. Other similar web sites include www.findmypast.com and www.genealogy.com. At least two web sites (www.familytreedna.com and dna.ancestry.com) offer DNA testing which can be used to trace the maternal and paternal lineages of the sample's donor.

Information about ancestry can also have its practical purposes. Genetic factors are known to underlie diseases and health. A family tree along with the health history of one's ancestors can be beneficial in identifying risk factors and taking preventive health care measures. In law enforcement and criminal justice system, DNA tests are frequently used to determine kinship. The results of such tests can be pivotal evidence in alimony cases for example.

A family tree usually has one of two forms. In one form the most recent individual is placed at the bottom as the root of the tree. Above are the individual's parents and above them are their parents and so on and so forth. Technically this type of "tree" doesn't meet the strict definition of a tree for very long since the likelihood of two branches merging increases as more generations are added to the "tree". This fact is however not all that important.

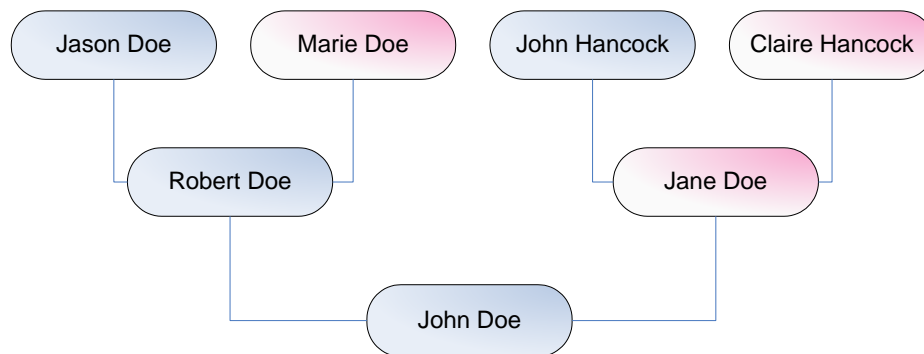


FIGURE 1 - AN ANCESTRAL FAMILY TREE.

The second form of family trees has one ancestor as the root but this time placed at the top. Below the ancestors are his/her children and below them are the grandchildren and so on and so forth.

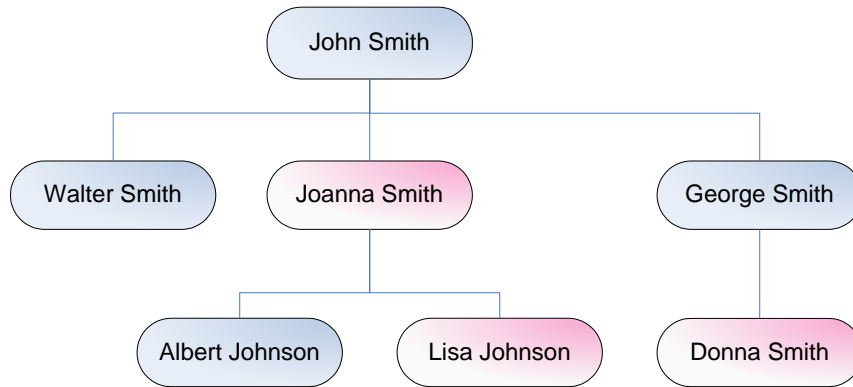


FIGURE 2 - A DESCENDANT FAMILY TREE.

Now, this second form of a family tree closely resembles phylogenetic trees that are used to describe the evolution of species. Phylogenetic trees are of a special interest to us because proven methods and models exist that make it possible to infer phylogenies of different species by comparing their DNA/RNA. Phylogenetic trees are then constructed from pair-wise distances to a common ancestor for all the sampled species. Lets for example assume that species A and B had a common ancestor 400,000 years ago while species A and C had a common ancestor one million years ago. We can then draw a phylogenetic tree for the three species like so:

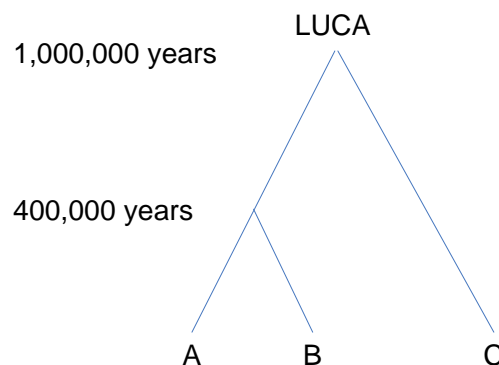


FIGURE 3 - A MOCK PHYLOGENETIC TREE. LUCA STANDS FOR LAST UNIVERSAL COMMON ANCESTOR.

In theory, a similar approach could be used when comparing segments of a chromosome sampled from two individuals of the same species. An approximation could be made of how long ago the two individuals shared an ancestor from whom they inherited this segment. Comparing a different pair of segments might yield another result which would indicate another common ancestor. As more and more comparisons are made the clearer the picture would become of how these two individuals are related through different ancestors. Then the question becomes this: If we were to compare enough segments from multiple individuals, could we reconstruct their pedigrees?

In this research project we aim to come up with at least a partial answer to this question. We investigate the possibility of inferring pedigrees in a population in very simple cases using mainly combinatorial approaches. We create a program that has two main functions:

- To simulate pedigrees in a population.
- To find the set of pedigrees compatible with the observed pair-wise distances between extant individuals in the population.

Due to the expected computational complexity of the problem, the number of extant individuals and pedigree depth (number of generations) will be kept very small.

2 BACKGROUND AND RELATED WORK

Let's start out by getting some terminology out of the way. When we talk about **extant individuals** we are referring to the known contemporary individuals. We assume these are the individuals whose genetic samples are being used in order to infer or reconstruct the pedigree. We probably know something about them, such as their gender, age, name etc. In a pedigree graph, the extant individuals are located at the bottom in the most recent generation. Even though the most immediate ancestors of the extant individuals may very well be alive and known, they are considered to be unknown as far as the reconstruction process is concerned. The extant individuals therefore have unique labels whereas the ancestors are unlabelled.

The most distant ancestors at the top end of a pedigree graph whose parents are not depicted are called the **founders**.

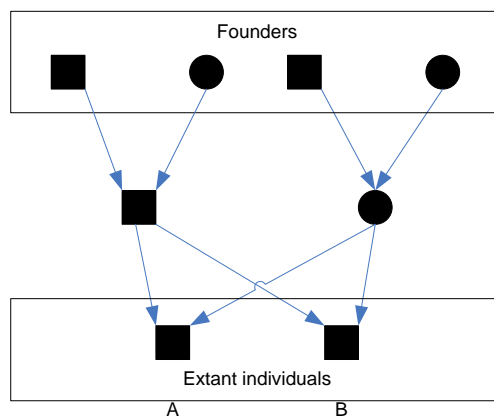


FIGURE 4 - FOUNDERS AT THE TOP AND EXTANT INDIVIDUALS AT THE BOTTOM

Pedigree depth is another term which we find helpful. The depth of a pedigree is the number of nodes along the longest path between a founder and an extant individual. In figure 4 the depth of the pedigree is 3.

In 2006 Mike Steel and Jotun Hein (1) had a paper published titled 'Reconstructing pedigrees: A combinatorial perspective'. As the title suggest, it is quite relevant to our project. In it the authors provide proof that population pedigrees can be deterministically reconstructed either using pair-wise amalgamation of the pedigrees of all pairs of extant individuals (Theorem 2.1) or by using 'link' and 'lasso' data (Theorem 3.1). Steel and Hein however assume that the gender of each ancestor is known. Neither theorem holds true if the genders are not known. They even provide a counter-example that demonstrates that pedigrees cannot be deterministically reconstructed using only the paths from one extant individual to another if the genders of the ancestral nodes that the paths lie through are not known.

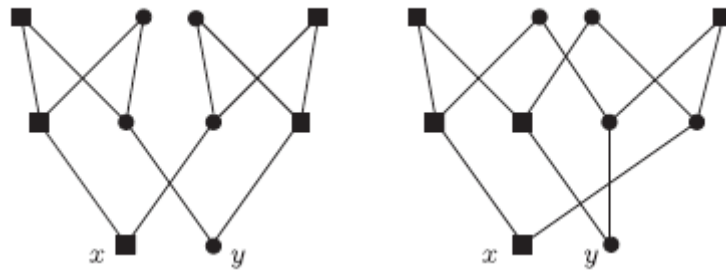


FIGURE 5 - STEEL'S AND HEIN'S COUNTER-EXAMPLE

Figure 5 depicts this counter-example. The two pedigrees in the figure show two individuals that share the same four grandparents. These two pedigrees are not the same, but they are similar in many other respects, e.g. the pedigrees both have the same embedded phylogenetic trees.

This counter-example tells us that if pedigrees are reconstructed using only pair-wise paths without gender information, multiple solutions may be found.

Another thing that's pointed out by Steel and Hein is that when pedigrees are reconstructed from genetic data, information about past individuals that did not have any children is lost since they did not contribute any genetic material to the extant individuals.

The paper contains several definitions, two of which we'll use without alteration. The first definition defines a pedigree.

A (strict) pedigree \mathbf{P} is an acyclic digraph, for which the vertex set \mathbf{V} is the disjoint union of two subsets \mathbf{M} and \mathbf{F} ('Male' and 'Female') and for which each vertex $\mathbf{v} \in \mathbf{V}$ satisfies the following condition:

(P) if \mathbf{v} has positive in-degree then \mathbf{v} has exactly two incoming arcs, say (\mathbf{u}, \mathbf{v}) and $(\mathbf{u}', \mathbf{v})$ where $\mathbf{u} \in \mathbf{M}$ and $\mathbf{u}' \in \mathbf{F}$.

Condition (P) formalizes the requirement that each individual in the set \mathbf{V} that has at least one parent in \mathbf{V} has exactly two, one male and one female. [...]. If \mathbf{X} is a subset of the vertices of \mathbf{P} that have no outgoing arcs, then we say that \mathbf{P} is a pedigree on \mathbf{X} .

Steel and Hein also define a general pedigree that, unlike strict pedigrees, allows individuals to have a single parent. Since our simulated and reconstructed pedigrees will adhere to the strict definition, we will from here on assume that when we write pedigree we mean strict pedigree and not general pedigree¹.

¹ Reconstructed pedigrees will not contain any gender information for non-extant individuals but they will be gender labellable.

The second definition defines pedigree isomorphism which is a fundamental concept when it comes to pedigree reconstruction. We already know that if ancestors are not gender labelled we may come up with multiple solutions but we're only interested in unique solutions. We don't want two results that are in essence the same pedigree only with some of the nodes in a different order.

It is important to formalize what it means for two pedigrees to 'be essentially the same' (isomorphic) where our set of extant individuals X is known (and so effectively labelled) but where we do not care about the labelling of the ancestral individuals. [...]. Two pedigrees on X , $P_1 = (V_1, A_1)$ and $P_2 = (V_2, A_2)$, are isomorphic, written $P_1 \cong P_2$ if there is a bijective map $\phi : V_1 \rightarrow V_2$ for which $(v_1, v_2) \in A_1 \Leftrightarrow (\phi(v_1), \phi(v_2)) \in A_2$ (i.e. ϕ is a digraph isomorphism) and ϕ is the identity map when restricted to X .

The gist of the definition is that pedigree isomorphism is exactly the same as isomorphism for directed graphs (digraphs) except that the extant individual nodes in the set X are uniquely labelled and thus put a restriction on the isomorphism.

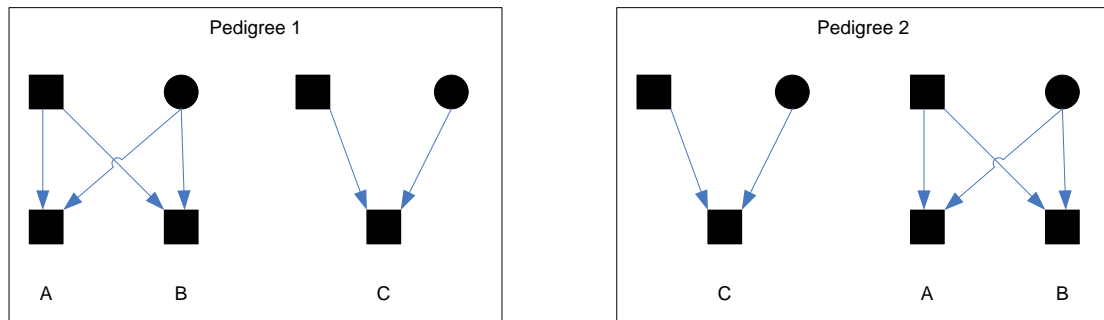


FIGURE 6 - ISOMORPHIC PEDIGREES

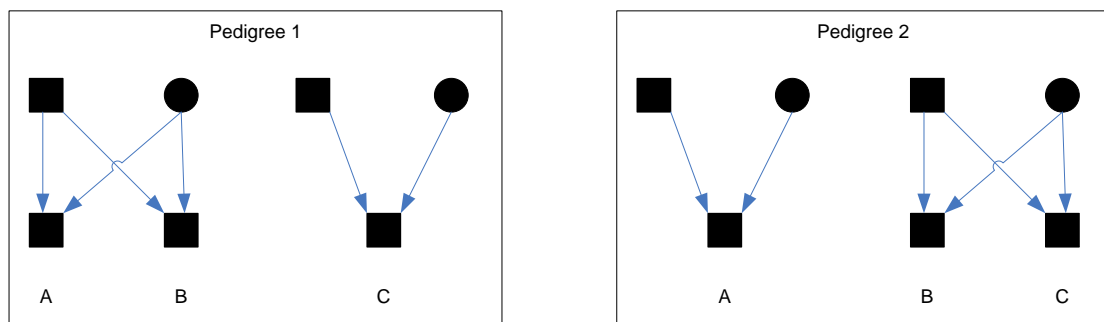


FIGURE 7 - AN EXAMPLE OF TWO GRAPHS THAT ARE DIGRAPH ISOMORPHIC BUT NOT PEDIGREE ISOMORPHIC

3 TOPIC DESCRIPTION / MOTIVATION

The original description for this project, written by Jotun Hein and Steffen Lauritzen, (2) provides the following motivation:

“Due to methodological advances and the phenomenal increase in genetic data from different species, phylogenetic analysis (the inference of evolutionary relationships between species) has risen to prominence and been put on much firmer statistical ground (Felsenstein, 2004; Semple and Steel, 2003). Similarly, considerable progress has been made in characterizing the genealogical relationships between segments of chromosome sampled from individuals from the same species, based on the pattern of mutational and recombinational diversity present in these segments. While such studies are important in their own right, the growing deluge of genomewide SNP genotype data (and in the near future sequence data) heralds the possibility of inferring the entire pedigree between a set of individuals. For any given contemporary individual, the ancestor(s) of a particular chromosomal segment represent only a fraction of all the ancestors from whom that individual inherited all his or her genetic material. However, when genotypes are obtained from different segments across the genome at a sufficiently dense rate, it becomes at least theoretically possible to reconstruct the genealogical links between contemporary individuals through all ancestors.”

To our best knowledge, this type of pedigree reconstruction is a relatively young subject with only a handful of published papers directly addressing it. This leaves ample room for research. Furthermore, pedigree reconstruction seems guaranteed to become more relevant in the years to come as theoretical possibilities are realised through advancements in methods and technology. When the subject’s young age and its future promise are considered it becomes apparent that this is a very interesting research topic indeed.

4 RESEARCH METHODS

In this chapter we describe the input data for the pedigree reconstruction algorithm. We provide insights into how this input data could theoretically be obtained by comparing genetic samples from a set of extant individuals. We then proceed to describe the reconstruction algorithm itself and provide a pseudo-code version of it. Finally we describe a random pedigree simulator that we created in order to generate sample input data for testing the reconstruction algorithm.

4.1 INPUT DATA

4.1.1 COMPARING DNA SEGMENTS

What can we learn about the common ancestors of a pair of individuals by comparing samples of their DNA? Imagine that we have DNA sequence data from two extant individuals and that we're focused in on a specific DNA segment that these two individuals have inherited from a common ancestor. The segments are not exactly the same. Mutations have occurred at certain locations. Now imagine that we have a model which can tell us the most likely sequence (and thus the number) of mutations that these two segments have undergone and that we know the mutation rate of the population. Then we can infer how many years ago the common ancestor was alive. If we then assume that each generation produces its offspring at a constant interval then we can infer how many generations we have to go back to find the common ancestor. If we come to the conclusion that the common ancestor for the segment lived 100 years ago and that the constant reproduction interval is 25 years, then we can assume that the two individuals shared a great-great-grandparent since

$$\frac{100 \text{ yrs}}{\frac{25 \text{ yrs}}{\text{gen}}} = 4 \text{ gen}$$

This hypothetical example of ours is a gross simplification of what is actually required to obtain information about common ancestors. We haven't even mentioned genetic recombination. The example is still good enough for our intents and purposes. The focus of our project is reconstructing pedigrees given data about common ancestors. How that data is obtained is outside the scope of this project.

4.1.2 RATIOS

What if we find that two separate segment pairs came from a common ancestor k generations ago, do we know whether it was inherited from the same ancestor or two distinct ancestors? To try to work that out we need more information. More specifically we need to know how many pairs were compared and how many of them gave the same result k .

If we compare 40 DNA segment pairs from two siblings, how many times would we expect to find that the common ancestor of a segment pair existed one generation ago? What if they are half-siblings? If you guessed 40 and 20 respectively, you'd be wrong. The correct answer is 20 ($\frac{1}{2}$) for full-siblings and 10 ($\frac{1}{4}$) for half-siblings. The reason for this is that each segment could have been inherited from either the mother or the father

with equal probability. If we compare segments from a pair of individuals labelled A and B we have four distinct possibilities:

1. A inherited it from his/her father and B inherited it from his/her father.
2. A inherited it from his/her father and B inherited it from his/her mother.
3. A inherited it from his/her mother and B inherited it from his/her father.
4. A inherited it from his/her mother and B inherited it from his/her mother.

All possibilities are equally probable. Assuming A and B are full-siblings then in two of them A and B inherit the segment from the same parent. If A and B are half-siblings then they either share a mother or a father. In either case only one possibility out of four would lead to the common parent. This tells us that no matter how many segment pairs we compare, we'd expect $\frac{1}{2}$ of them to come from a common parent for full-siblings and $\frac{1}{4}$ of them to come from a common parent for half-siblings.

How about grandparents? Most people have four distinct grandparents so a segment could have been inherited from any of them with equal probability ($\frac{1}{4}$). If two individuals share a single grandparent then the odds of them both inheriting the same segment from the common grandparent are:

$$\frac{1}{4} * \frac{1}{4} = \frac{1}{16}$$

This tells us that we expect every 1 out of 16 segment pairs to come from a common ancestor 2 generations ago if the two individuals share a single grandparent.

It's worth noting that finding the odds (ratios) for two individuals becomes increasingly more complex as we go further back in time. The number of common ancestors in each generation will increase and so will the number of related ancestors (common or not). Both of these factors affect the ratios. We cover this topic in more detail in subsection 4.2.3.

4.1.3 PUTTING IT ALL TOGETHER

We already mentioned that we assume that there's a constant reproduction time which results in non-overlapping generations. We now come to our second major assumption: infinite genomes. The reason for our second assumption is rather simple. If we have infinite genome lengths then our genome must contain genetic material from all our ancestors which in turn means that we can find the distances (number of generations) to all common ancestors for a pair of individuals. In addition, the observed ratios will approach their expected values as the number of segment pairs compared approaches infinity.

We've already established that by comparing genetic samples from a pair of individuals we can gather data about distances to common ancestors and the ratio of each distance. But we're not reconstructing the pedigree of two individuals; we want to reconstruct population pedigrees. That means that we need to gather distance and ratio data for all possible pairs of extant individuals in the population! Imagine that we have a tiny population of only four individuals labelled A, B, C and D. Then we need to gather

distance and ratio data for the pairs (A, B), (A, C), (A, D) (B, C), (B, D) and (C, D). Following is an example of what the input data could look like for this tiny population:

TABLE 1 - EXAMPLE INPUT DATA (LIMITED TO DISTANCE ≤ 2)

Pair	Distance	Ratio
A-B	1	0.500
A-D	1	0.250
B-D	1	0.250
C-D	1	0.250
A-C	2	0.250
B-C	2	0.250
C-D	2	0.125
A-B	2	0.125
A-D	2	0.125
B-D	2	0.125

4.2 RECONSTRUCTION ALGORITHM

4.2.1 OVERVIEW

Our pedigree reconstruction algorithm reconstruct one generation at a time, starting with the parents of the extant individuals. The algorithm generates all possible ways of grouping the extant individuals into sibling groups. It then merges and splits the parent nodes of the extant individuals in accordance with each generated sibling group configuration and then inspects its working copy pedigree to see whether it agrees with the input data. If it does then the working copy pedigree is cloned and added to a list of possible solutions. When the algorithm moves on to the next generation, all solutions from the previous generation are tried as “starting points”. The algorithm terminates when it has reconstructed the generation which is at an equal distance from the extant individuals as the longest distance in the input data.

This is in essence a brute-force algorithm which generates a whole lot of pedigrees and sees which ones agree with the input data. In section 5.2 we cover optimizations in the implementation of the algorithm, some of which rule out certain sibling group configurations which we know beforehand will not agree with the input data.

4.2.2 ENUMERATING SIBLING GROUP CONFIGURATIONS

The reconstruction algorithm must be able to enumerate all the different ways that the individuals in a particular generation can be partitioned into sibling groups. We do this by generating all partitions of a set. Following is the Wikipedia definition of a partition of a set (3):

A partition of a set X is a set of nonempty subsets of X such that every element x in X is in exactly one of these subsets.

Equivalently, a set P of nonempty sets is a partition of X if

The union of the elements of P is equal to X . (We say the elements of P cover X .)

The intersection of any two elements of P is empty. (We say the elements of P are pairwise disjoint.)

Let's take an example. Imagine that we have a generation of three individuals labelled A, B and C. These individuals make up the set $G = \{A, B, C\}$. The set G has five partitions, namely:

1. $\{\{A\}, \{B\}, \{C\}\}$
2. $\{\{A, B\}, \{C\}\}$
3. $\{\{A\}, \{B, C\}\}$
4. $\{\{A, C\}, \{B\}\}$
5. $\{\{A, B, C\}\}$

Individuals in the same subset are considered to be siblings. In partition number 1 there are no siblings since there's no subset that contains more than one individual. In partition 2 A and B are siblings while in partition 3 B and C are siblings and so on and so forth.

There's more to it though. We have to consider both half-siblings and full-siblings. In order to do that the algorithm employs a small trick. It starts by generating parent nodes for all the individuals in the set G . The parent nodes are assigned genders while the reconstruction is taking place so initially each individual in G has a father node and mother node. Having generated all the partitions of G the algorithm iterates through them in two loops with one loop nested in the other. In the outer loop the father nodes of those individuals that are in the same subset are merged into one while in the inner loop the mother nodes of those individuals that are in the same subset are merged.

Let's take an example. Suppose that the outer loop is in its first iteration while the inner loop is in its second iteration. That means that in the algorithm's working copy pedigree A and B have the same mother since A and B are in the same subset in partition 2. A, B and C will have distinct fathers since there's no subset with more than one individual in partition 1.

It's important to note that we have no knowledge about the gender of the ancestors of the extant individuals. The algorithm simply assumes that it knows the genders while the reconstruction is taking place. Once all possible solution pedigrees have been found, the gender labels of the ancestral nodes are removed. The reason for doing this is that it prevents the algorithm from creating and considering pedigrees which cannot be gender

labelled. Figure 8 shows an example of a pedigree graph which cannot be gender labelled.

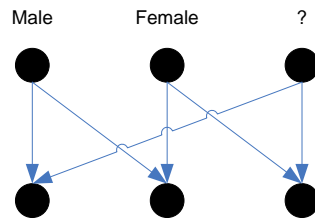


FIGURE 8 - A PEDIGREE GRAPH WHICH CANNOT BE GENDER LABELLED

4.2.3 CALCULATING THE RATIOS

Every time the algorithm changes to the working copy to a new configuration, it needs to be inspected to see whether it conforms to the input data. To do this we need a way to gather information about pair-wise distances and ratios from the working copy pedigree. It just so happens that there's a rather nifty approach to calculating the ratios. Let's start by laying some foundations to the approach.

Remember that we assume that all individuals have infinite genomes. That means that any given individual will be expected to have inherited half of his/her genetic material from each parent, one fourth from each grandparent, one eighth from each great-grandparent etc. The expected fraction of genetic material inherited from an ancestor k generations ago is therefore $\left(\frac{1}{2}\right)^k$. The previous formula however doesn't always hold true. Consider the following example:

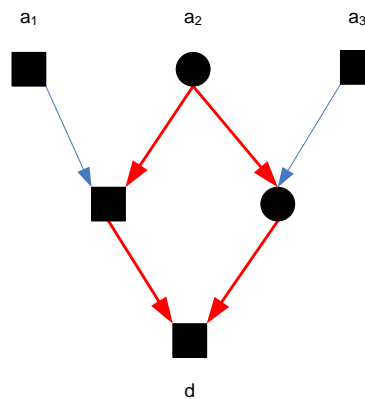


FIGURE 9 - AN INDIVIDUAL WITH ONLY THREE GRAND-PARENTS

Here d will have inherited $\frac{1}{4}$ of his genetic material from a_1 , $\frac{1}{4}$ from a_3 and $\frac{1}{2}$ from a_2 . We will therefore need to revise our formula. If a is an ancestor of d and k is the distance (number of generations) from a to d and p is the number of paths from a to d in the pedigree graph then the expected fraction of genetic material that d inherited from a is $p * \left(\frac{1}{2}\right)^k$.

So how does the number of paths from an ancestor to a pair of descendants affect the expected ratio between the pair in the input data? Let's consider the pedigree in figure 10.

example illustrates that it's only the number of node-disjoint path pairs that counts. Armed with this knowledge we can now present a method for calculating the ratio for a pair of extant individuals that share a common ancestor.

If d_1 and d_2 are extant descendant nodes, a is a common ancestor of d_1 and d_2 , k is the number of generations from a to either d_1 or d_2 , p_1 is the set of all paths from a to d_1 and p_2 is the set of all paths from a to d_2 then let q be the set of path pairs so that:

$$(x,y) \in q: \{x \in p_1; y \in p_2; x \text{ and } y \text{ are node-disjoint}\}.$$

We can then calculate the ratio for the pair (d_1, d_2) using the following formula:

$$|q| * \left(\frac{1}{2}\right)^k * \left(\frac{1}{2}\right)^k$$

If d_1 and d_2 have more than one common ancestor at distance k then this method is simply employed for each common ancestor and the results summed together to find the ratio.

4.2.4 PSEUDOCODE

```

ReconstructPedigree(distancesAndRatios)
  let i be the longest distance between any two pairs in the input data
  let S be the empty set
  add to S a pedigree containing only the extant individuals
  for generation g from 0 to i-1
    let R be the subset of input data where the distance equals g
    S = FindPedigreeSolutions(g, R, S)
  end for
  return S
end ReconstructPedigree

```

```

FindPedigreeSolutions(g, R, S)
  let S' be the empty set
  for each pedigree p in S
    let I be the set of individuals in generation g in p
    let P be the empty set
    for each individual i in I
      create and assign a mother to i
      add the mother to P
      create and assign a father to i
      add the father to P
    end for
    let SP be all the set-partitions of I
    for i = 0 to |SP|-1
      let p be the set-partition at index i in SP
      for each subset ss in p
        merge the fathers of all the individuals in ss
      end for
      for j = i to |SP|-1
        let p' be the set-partition at index j in SP
        for each subset ss' in p'
          merge the mothers of all the individuals in
            ss'
        end for
        let R' be the empty set
        for each parent p in P
          for all distinct pairs (j,k) of p's extant
            descendants
              let r = CalculateRatio(p, j, k)
              if (j,k) exists in R then
                increase the ratio of (j,k) in
                  R' by r
              else
                add (j,k) to R' with a ratio
                  of r
              end if
            end for
          if R equals R' then
            if the current pedigree is not
              isomorphic to a previous solution in
                S'
              clone the current pedigree and
                add to S'
            end if
          end if
        end for
      end for
      undo mergers of mothers of the individuals in I
    end for
    undo mergers of fathers of the individuals in I
  end for
  return S'
end FindPedigreeSolutions

```

```

CalculateRatio(ancestor, descendantA, descendantB)
  let PA be the set of all paths from ancestor to descendantA
  let PB be the set of all paths from ancestor to descendantB
  let ratio = 0
  let g be the number of generations from ancestor to descendants
  let d = 2 to the power of g
  for each path p in PA
    for each path p' in PB
      if p and p' are node disjoint then
        ratio = ratio + (1/d * 1/d)
      end if
    end for
  end for
  return ratio
end CalculateRatio

```

4.3 PEDIGREE SIMULATOR

This section describes the functions and implementation of the pedigree simulator. We created the simulator in order to generate sample input data that we could then use to test the pedigree reconstruction algorithm. We'll start with some screenshots and basic explanations of the GUI that we created for the simulator. In following subsection we'll go into more of the implementation details.

4.3.1 THE GUI

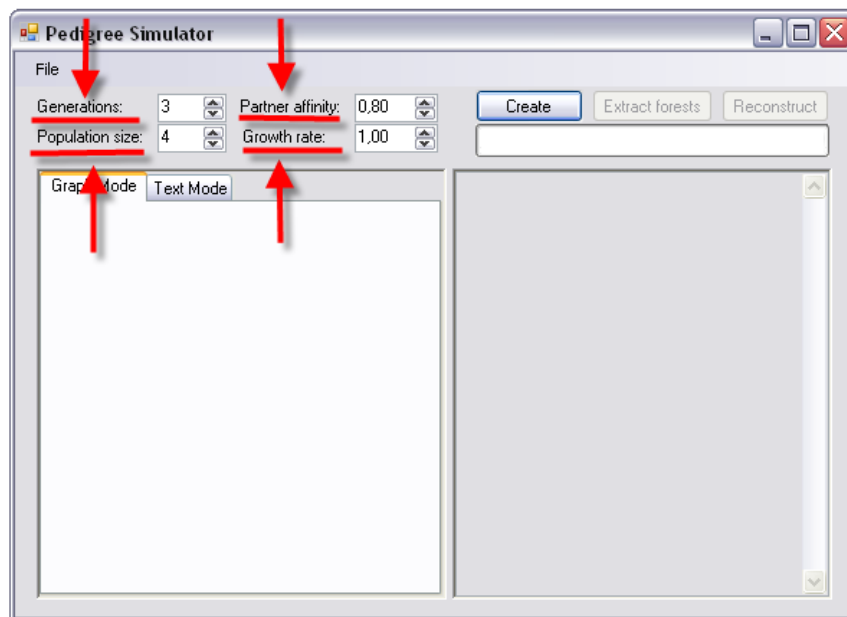


FIGURE 12 - PEDIGREE SIMULATOR

In figure 12 we see the simulator just after startup. The underlined parameters have been populated with their default values. Modifying them allows the user to affect how the pedigree will be simulated.

- **Generations:** The number of generations to have in the simulated pedigree.
- **Population size:** The number of individual in the founder generation.
- **Partner affinity:** Controls how likely a parent is to stick with the same partner for procreation.
- **Growth rate:** This parameter controls whether the population size stays constant, shrinks or grows. It also controls how fast the population size shrinks

or grows (exponentially). A value less than 1 will make the population shrink, a value of 1 means the population size will be constant and a value greater than 1 will make the population grow.

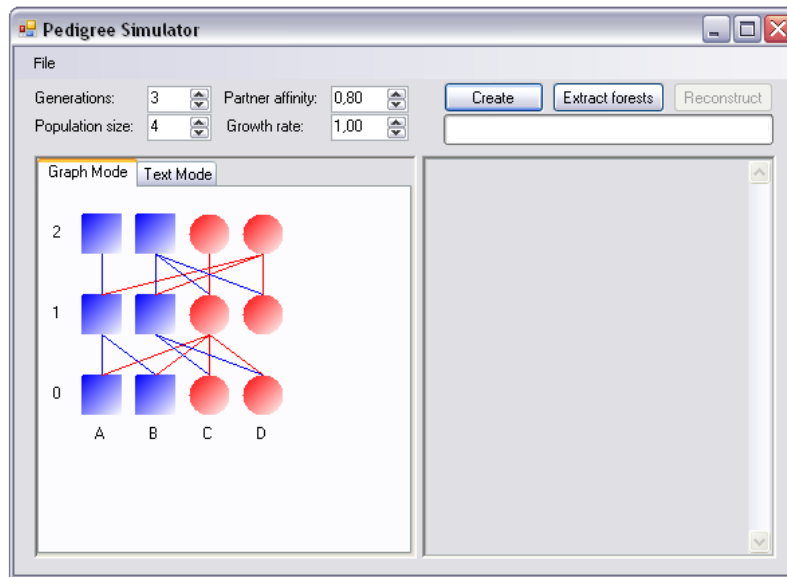


FIGURE 13 - SIMULATED PEDIGREE

In figure 13 we see a simulated pedigree, the red circles are females and the blue boxes are males. The connection from mother to child is a red line and the connection from father to child is a blue line. Here it is possible to see the numbering of the generations, extant individual are at the bottom in generation 0 and the founders are at the top in generation 2.

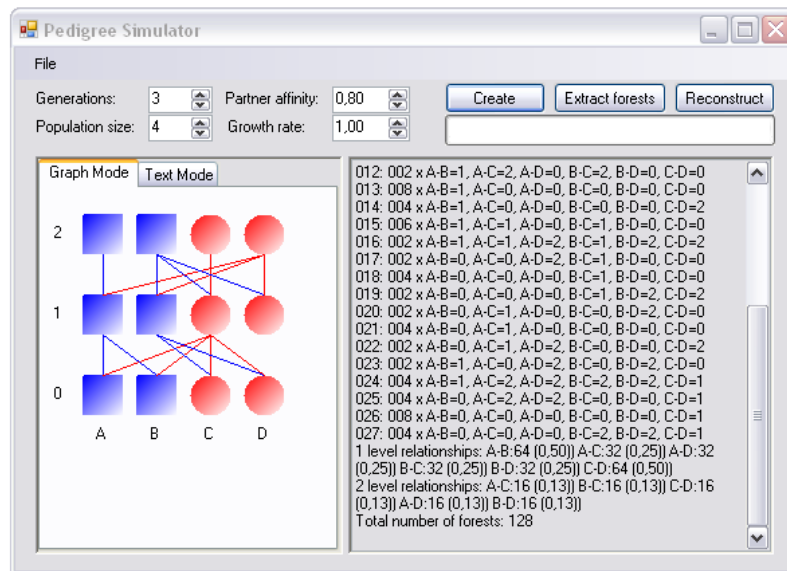


FIGURE 14 - RECONSTRUCTION DATA EXTRACTED FROM SIMULATED PEDIGREE

In figure 14 we see a screenshot that shows how the input data for the pedigree reconstruction algorithm has been extracted from the simulated pedigree and displayed in a textbox on the right. The last few lines in the textbox show the pair-wise distances and ratios (called *x level relationships* for reasons unknown).

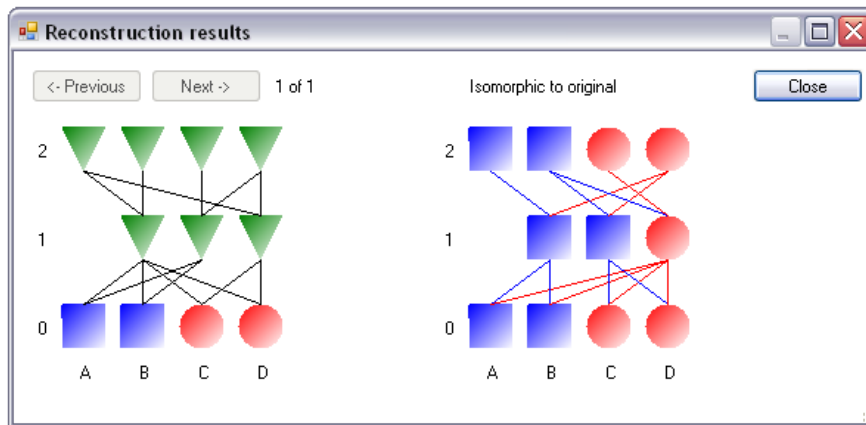


FIGURE 15 - RECONSTRUCTED PEDIGREE

This is the dialog that pops up when the program has finished reconstructing a pedigree. On the left is the reconstructed pedigree and on the right is the original pedigree. Individuals whose gender is unknown are represented by a green triangle.

4.3.2 IMPLEMENTATION DETAILS

4.3.2.1 The collection of individuals

Pedigree simulation starts by creating a collection of all the individuals that make up the pedigree. The total size depends on the population size, number of generations and the growth rate. The collection that we created is based on a one-dimensional array. We did create an interface for the collection to make it easier for us to change the underlying data structure if need be.

The collection contains several generations that are numbered from 0 to $N-1$ where 0 is the most recent generation, $N-1$ is the founder generation and N is the total number of generations.

Each generation contains a set of individuals that are numbered from 0 to $M-1$ where M is the total number of individuals in the generation. Males have the numbers 0 to $\frac{M}{2} - 1$ and females have the numbers $\frac{M}{2}$ to $M-1$.

Each individual has the following properties:

- Sex
- Father
- Mother
- Children

By storing information about parents and children we can traverse the lineages (paths) in both ascending and descending order.

4.3.2.2 Choosing parents

We created a single interface that is called `IParentChooser` so that so that multiple implementations of how parents are chosen could be used interchangeably without having to change anything in the simulator code.

Once the collection of individuals has been constructed, the simulator will choose parents for each individual starting with the individuals in the most recent generation and moving backwards in time from there.

The first implementation of that interface is the *Total Promiscuity Chooser*. It randomly selects a father and a mother from the preceding generation.

We also created another implementation of the interface, the *Total Monogamy Chooser*. That implementation starts by randomly partnering all the individuals in the preceding generation. For each individual it will then choose a random father. The partner of the father will then be assigned as the mother of the individual.

The final implementation, and the one that the current version of the simulator uses, is the *Hybrid Parent Chooser*. This implementation is a mixture of total promiscuity and total monogamy. A parameter named *Partner Affinity* controls how likely a male is to mate with the same female over and over again. A value of 0 means that a partner will never mate with the same female twice while a value of 1 means that a male will always mate with the same female.

5 RESULTS

5.1 PEDIGREE ISOMORPHISM

One fundamental requirement of the pedigree reconstruction algorithm was that it only return mutually non-isomorphic results. For that we needed an algorithm which could determine whether or not two pedigrees are isomorphic.

Initially we wondered whether there was a polynomial time algorithm for pedigree isomorphism. The in-degree of pedigree graphs is bounded by 2, they are acyclic, discrete-generation pedigree graphs are bipartite and the extant individual nodes are labelled. All these properties restrict pedigree graphs so it doesn't seem too far-fetched that pedigree isomorphism could be a simpler problem than general graph isomorphism. With that being said, our efforts proved fruitless.

Graphs with a bounded degree can be checked for isomorphism in polynomial time (4). Pedigrees however only have a bounded in-degree. The out-degree (number of children) is virtually unbounded. Bipartite graph isomorphism is just as hard as general graph isomorphism (5) so that doesn't help us either. Planar graphs can be checked for isomorphism in polynomial time (6). Constructing a pedigree graph that contains a subdivision of $K_{3,3}$ can be done by adding three male and three female nodes to the pedigree and then let each male have a child with each of the females (nine children in total) so it's apparent that pedigrees are not guaranteed to be planar.

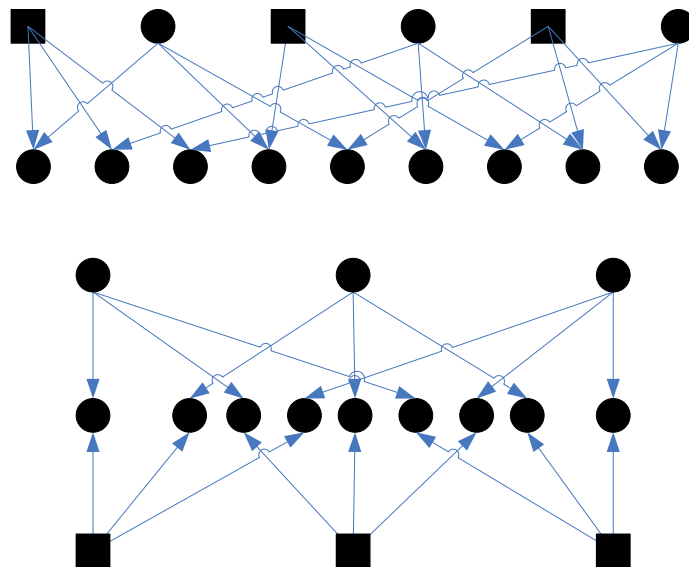


FIGURE 16 - A PEDIGREE GRAPH THAT CONTAINS A SUBDIVISION OF $K_{3,3}$

We did dabble a bit with a polynomial time algorithm which (supposedly) checked for graph isomorphism by ordering the adjacency matrixes of the two graphs. The algorithm returned correct results in most cases but not always so we ultimately decided to abandon it.

In the end we decided to use a general graph isomorphism algorithm. The algorithm we chose is named VF. We found a nice C# implementation of it on the web site

www.codeproject.com (7). The C# implementation is based on an optimized version of VF, sometimes referred to as VF2.

The reason we chose VF (or in our case the optimized version VF2) was that according to (8) VF and SD (Schmidt and Druffel) were the only algorithms to always find a solution and VF2 performed very well in the tests. A modified version of it is being used for graphs with up to 10.000 nodes (9) so we did not expect performance to be an issue considering the relatively puny size of the pedigrees we simulate in this project.

The VF algorithm has spatial complexity of $O(N^2)$ for best and worst case and has temporal complexity of $O(N^2)$ for best case and $O(N!N)$ for the worst case (9).

5.2 OPTIMIZATIONS

After having implemented the initial version of the pedigree reconstruction algorithm, we explored several ways to improve its performance. In this section we'll explain the optimizations that we employed along with test results that show how much an improvement each optimization made.

We created three test files to use for benchmarking our improvements. One test file generated only a handful of solutions; one generated an average number of solutions while the third one generated a large amount of solutions. This allowed us to better gauge how effective our improvements were in relation to the number of solutions found by the reconstruction algorithm.

TABLE 2 - TEST FILE USED FOR BENCHMARKING

File name	Total number of results	Mutually non-isomorphic results
3results.rln	3	2
68results.rln	68	16
267results.rln	267	89

We created a unit test that ran the reconstruction algorithm three times using each test file and returned the average execution time. When we performed the benchmarks we ran the unit test three times or while the execution times kept improving, first without the improvement in place and then with the improvement in place. We then compared the best (lowest) execution times.

5.2.1 TS-GRAPHS

While working on this project we found that there was a certain type of graph which proved to be very useful when dealing with pedigree reconstruction. We drew lots of them by hand. Sheets of paper filled with these graphs would invariably be scattered around our office. We dubbed them TS-graphs simply because we couldn't come up with a descriptive name for them. Ideas for some of the optimizations we did came from looking at TS- graphs so we feel it's important that we explain them.

A TS-graph focuses on a single generation in a reconstructed pedigree. Every individual in the generation is represented by a node. Labels for the nodes are constructed by concatenating the labels of each individual's extant descendants. If an individual is the ancestor of the extant descendants A and C then the node is labelled AC.

The input data is then examined to find all pairs of extant individuals that share a common ancestor at a distance which is one greater than the distance to the generation in the TS-graph. We do this because we're interested in knowing which individuals in the TS-graph might be siblings. If a pair of extant individuals (A, B) has a common ancestor at generation $k+1$ then we put edges from all ancestral nodes of A to all ancestral nodes of B in the TS-graph.

In figure 17 we see a pedigree on the left that has been reconstructed to generation 1. On the right we see the corresponding TS-graph for generation 1.

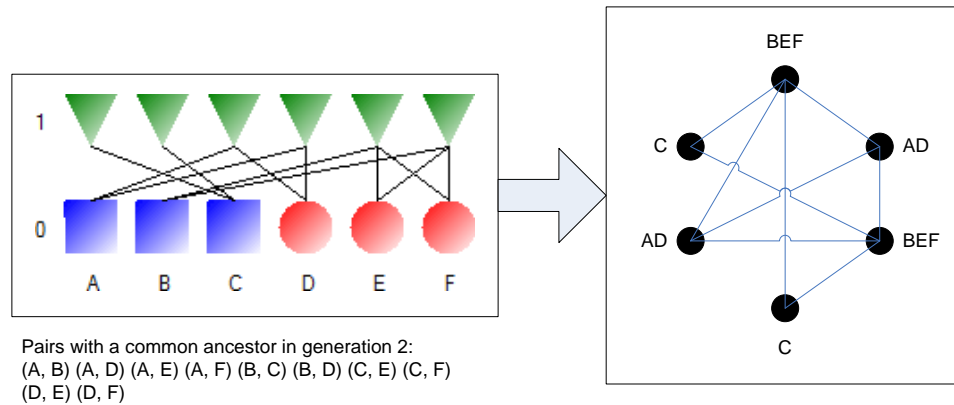


FIGURE 17 - TS-GRAPH EXAMPLE

By looking at a TS-graph we can quickly see that individuals connected by an edge might be siblings whereas individuals not connected by an edge cannot be siblings. This is useful information because it allows us to rule out certain sibling group configuration (or set-partitions) that we know beforehand will not be compatible with the input data. We discuss how we used this fact to optimize the reconstruction algorithm in subsection Allowed pairings 5.2.3.

TS-graphs also have a couple of other properties which inspired further optimizations. Subsection 5.2.5 covers an optimization which involves solving each connected component in a TS-graph separately. Subsection 5.2.7 covers another optimization which involves finding out which individuals must be in the same sibling group.

5.2.2 COMPARE ON THE GO

Initially the reconstruction algorithm would iterate through the entire generation being reconstructed and calculate and accumulate all the ratios before comparing them with the input data. We changed it so that:

- a) Every time the algorithm finds a common ancestor that links a new pair of extant individuals, the input data is examined to verify that the pair is expected to have a common ancestor at that distance.
- b) Every time the algorithm increments the ratio for a given pair of extant individuals, the input data is checked to see whether the ratio has exceeded that of the input data.

If either of those checks determines that the working copy does not conform to the input data then the algorithm immediately moves on to the next sibling group configuration.

TABLE 3 - COMPARE ON THE GO: BENCHMARK RESULTS

Test file	Before (ms)	After (ms)	Speed increase
3results.rln	936.0	509.6	84%
68results.rln	6084.0	2548.0	139%
267results.rln	3359.2	2126.8	58%

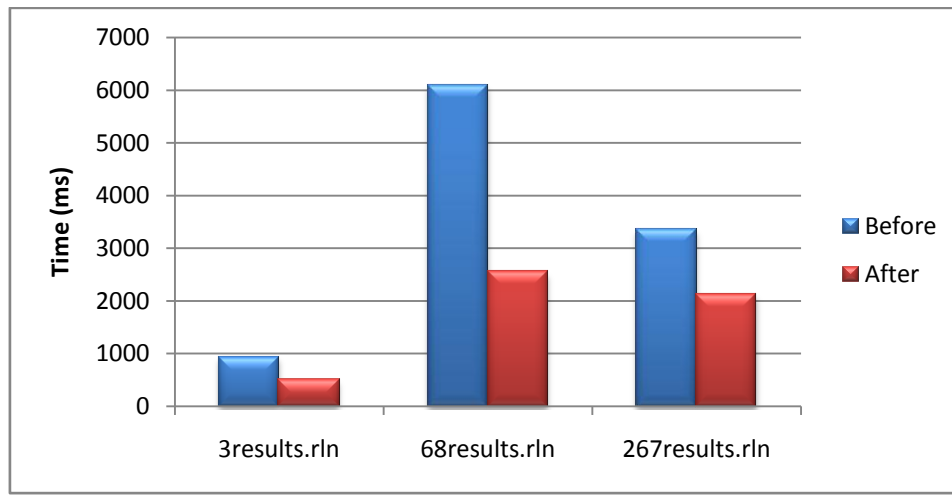


CHART 1 - COMPARE ON THE GO: EXECUTION TIMES

5.2.3 ALLOWED PAIRINGS

When the algorithm is about to reconstruct a new generation, it will generate all partitions of the set of individuals that are the children of the founder generation. The set-partitions are then used to merge the individuals in the founder generation back and forth until every possible way of grouping their children into sibling groups has been tried.

The Allowed pairings optimization attempts to limit the number of set-partitions used by eliminating those set-partitions that would result in a pair of extant individuals having a common ancestor in the founder generation when the input data tells us that no such common ancestor exists.

To accomplish this, the reconstruction algorithm starts by generating a list of allowed pairings. This list contains all pairs of children of the founder generation that can be made siblings without contradicting the input data. When the partitions of the set are being generated, each set-partition is inspected to verify that no subset contains a pair of individuals that's not in the list of allowed pairings. Otherwise the set-partition is discarded.

We should point out that when the first optimization was made (Compare on the go) we had not yet implemented isomorphism checks in the algorithm. The execution times in the

following table are therefore drastically different from those reported in the last benchmark results.

TABLE 4 - ALLOWED PAIRINGS: BENCHMARK RESULTS

Test file	Before (ms)	After (ms)	Speed increase
3results.rln	556.4	166.4	234%
68results.rln	2251.6	2080.0	8%
267results.rln	4643.6	4383.6	6%

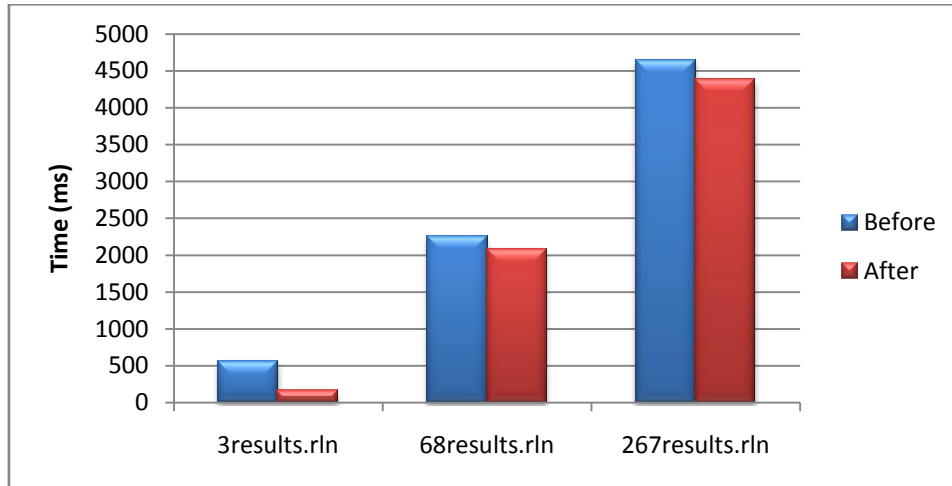


CHART 2 - ALLOWED PAIRINGS: EXECUTION TIMES

5.2.4 PATH CACHE

When the algorithm calculates the ratios at a certain distance in the working copy pedigree, it does so by accumulating the ratios for all pairs of extant descendants of each founder. If a founder has three extant descendants, A, B and C, then the ratios for the pairs (A, B), (A, C) and (B, C) are calculated. During the calculations all paths from the founder to each extant descendant are found. In the initial version of the algorithm the paths were discarded once the ratio for a single pair had been calculated resulting in the same set of paths having to be found over and over again. To correct this inefficiency we added a path cache so that the paths would only have to be found once for each extant descendant.

TABLE 5 - PATH CACHE: BENCHMARK RESULTS

Test file	Before (ms)	After (ms)	Speed increase
3results.rln	119.0	109.2	9%
68results.rln	2232.3	1705.6	31%
267results.rln	4399.2	4123.6	7%

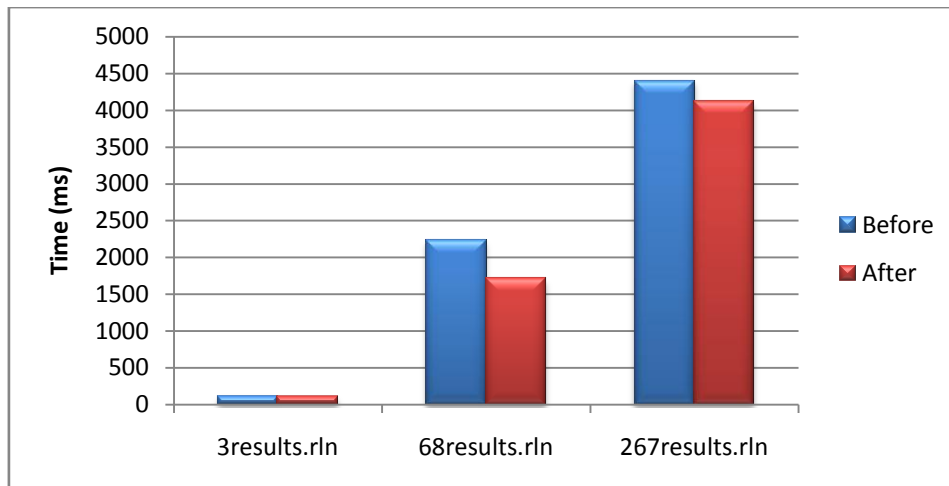


CHART 3 - PATH CACHE: EXECUTION TIMES

5.2.5 CONNECTED COMPONENTS

When drawing TS-graphs we often came across scenarios where the TS-graph had more than a single connected component. In such scenarios the reconstruction of a single generation can be divided into sub-problems since the individuals in a connected component can only be siblings of each other.

The major benefit of dividing the generation reconstruction problem into sub-problems is that it reduces the number of set-partitions generated. A set with 8 members has 4140 set-partitions while two sets with 4 members have a total of $15 + 15 = 30$ set-partitions.

We modified the reconstruction algorithm to generate a TS-graph, search for connected components and solve each one independently and finally merge the solutions.

TABLE 6 - CONNECTED COMPONENTS: BENCHMARK RESULTS

Test file	Before (ms)	After (ms)	Speed increase
3results.rln	119.6	114.4	5%
68results.rln	1716.0	1128.4	52%
267results.rln	4123.6	4342.0	-5%

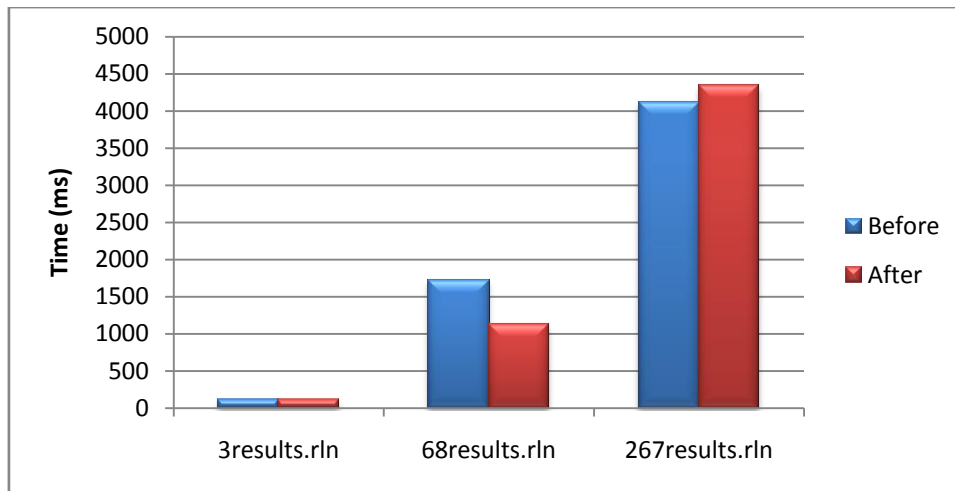


CHART 4 - CONNECTED COMPONENTS: EXECUTION TIMES

5.2.6 REVERSE ISOMORPHISM CHECKS

When the algorithm finds a new possible solution, it iterates through all solutions that have already been found and compares them to the current solution to find out whether any of them is isomorphic to the current one. The current solution is only added to the list if there's no previous solution that's isomorphic to it. If an isomorphic solution is however found the algorithm discards the current solution and moves on to the next sibling group configuration.

Initially the algorithm would iterate through the list of previously found solutions from the first solution found to the last one added to the list. By reversing the order we found that the algorithm performed significantly better for the test file that generated most solutions. This suggests that a possible solution is more likely to be isomorphic to a solution that was added recently to the list.

TABLE 7 - REVERSE ISOMORPHISM CHECKS: BENCHMARK RESULTS

Test file	Before (ms)	After (ms)	Speed increase
3results.rln	93.6	98.8	-5%
68results.rln	889.2	873.6	2%
267results.rln	4050.8	2808.0	44%

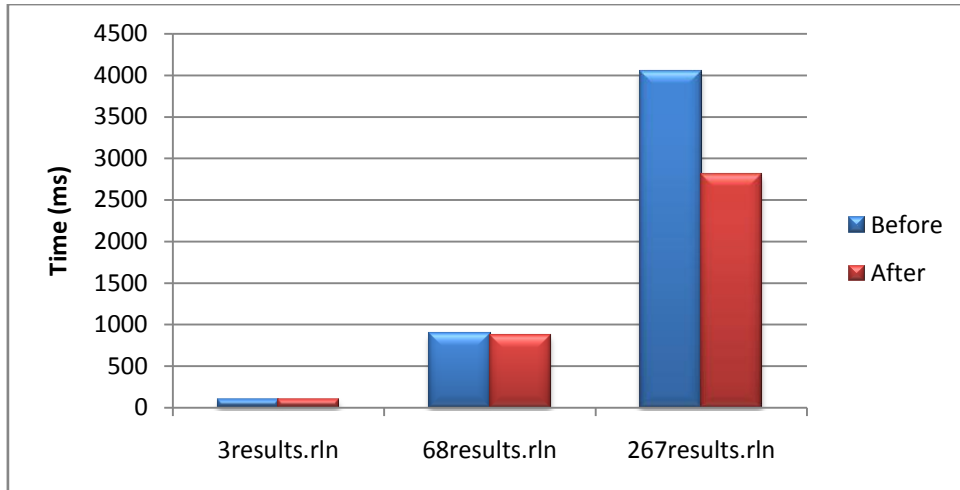


CHART 5 - REVERSE ISOMORPHISM CHECKS: EXECUTION TIMES

5.2.7 REQUIRED PAIRINGS

Just like the allowed-pairings optimization, the required-pairings optimization aims to reduce the number of set-partitions used. This is yet another idea that came to us when drawing TS-graphs. Remember that edges are added to a TS-graph between all ancestors of a pair of extant individuals that have a common ancestor at generation $k+1$ where k is the distance to the generation on the TS-graph. If there's only one edge added for a certain pair of extant individuals then we know that the individuals that are linked by the edge must be siblings. No sibling group configuration will conform to the input data unless it contains these two in a sibling group.

We modified the algorithm to compile a list of required pairings. When the set-partitions are being generated each one is checked to see whether it places those individuals that constitute a required pairing within the same subset. If not then the set-partition is discarded.

TABLE 8 - REQUIRED PAIRINGS: BENCHMARK RESULTS

Test file	Before (ms)	After (ms)	Speed increase
3results.rln	88.4	67.6	31%
68results.rln	878.8	878.8	0%
267results.rln	2818.4	2766.4	2%

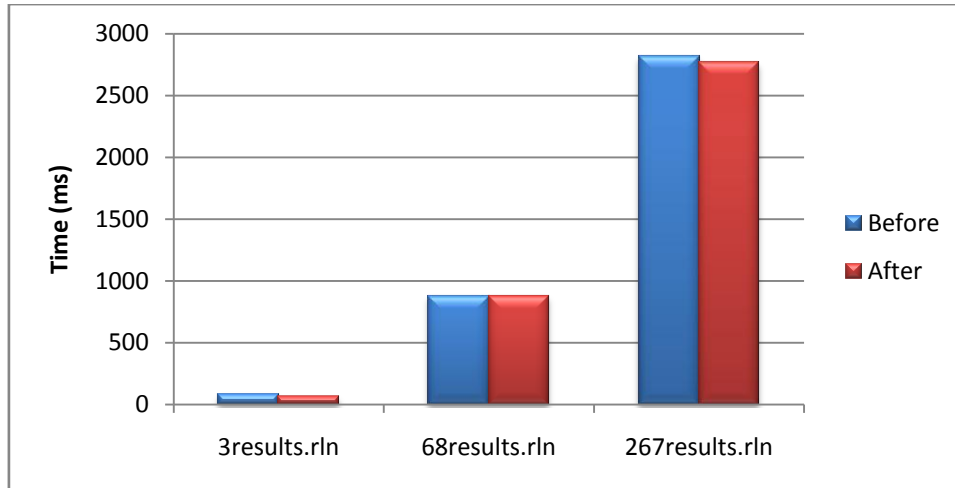


CHART 6 - REQUIRED PAIRINGS: EXECUTION TIMES

5.2.8 LABELS IN ISOMORPHISM CHECKS

We use a general graph isomorphism algorithm to check whether two pedigrees are isomorphic. Since pedigrees have labels on the extant individuals, some trickery is required to use a general graph isomorphism algorithm to check for pedigree isomorphism. Initially we did this by adding nodes to the graph and adding arcs from the extant individual nodes to the added nodes so that each extant individual node had a unique out-degree.

We later discovered that the implementation of the VF isomorphism algorithm that we used had built-in properties and interfaces which allowed us to restrict the isomorphism using unique labels of the extant nodes without adding any extra nodes or arcs. This resulted in the graphs being smaller which increased the performance of the isomorphism tests.

TABLE 9 - LABELS IN ISOMORPHISM CHECKS: BENCHMARK RESULTS

Test file	Before (ms)	After (ms)	Speed increase
3results.rln	62.4	62.4	0%
68results.rln	868.4	837.2	4%
267results.rln	2626.0	2199.6	19%

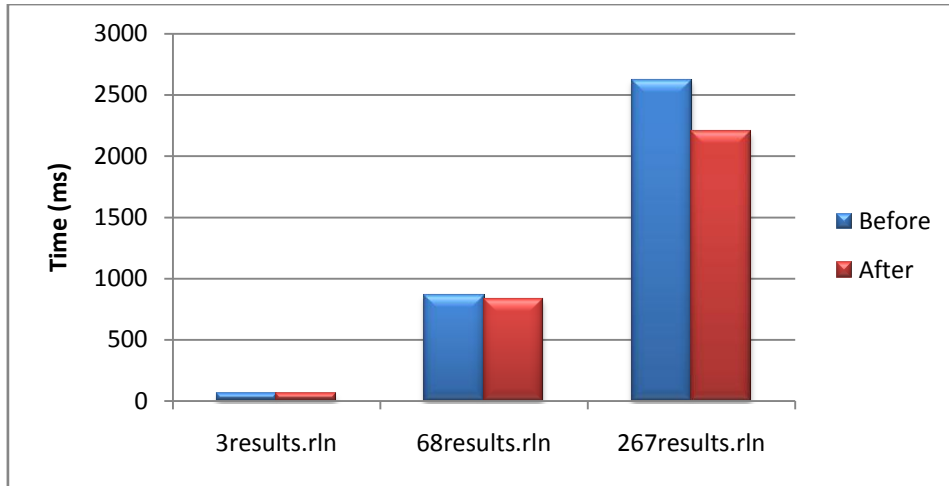


CHART 7 - LABELS IN ISOMORPHISM CHECKS: EXECUTION TIMES

5.2.9 SUMMARY

The overall effect of the optimizations made was quite significant. If we just look at the improvements made after the isomorphism checks were added we managed to more than double the speed for the toughest test file (267results.rln) while increasing the speed for the simplest test file (3results.rln) by a factor of almost 8!

TABLE 10 - OVERALL BENCHMARK RESULTS

Test file	First results (ms)	Non-isomorphic (ms)	Final (ms)	Increase from initial speed	Increase from non-isomorphic speed
3results.rln	936.0	556.4	62.4	1400%	792%
68results.rln	6084.0	2251.6	837.2	627%	169%
267results.rln	3359.2	4643.6	2199.6	53%	111%

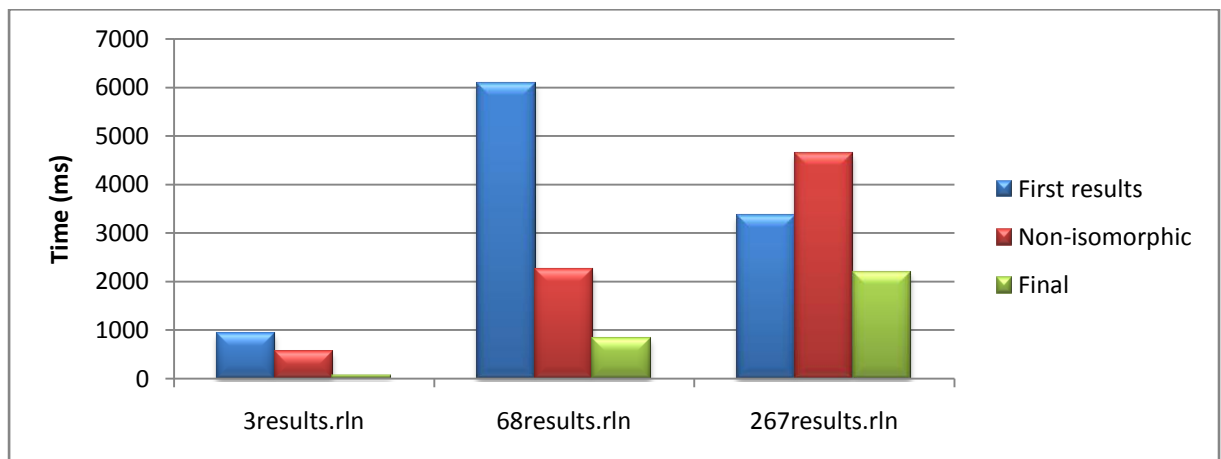


CHART 8 - OVERALL EXECUTION TIMES

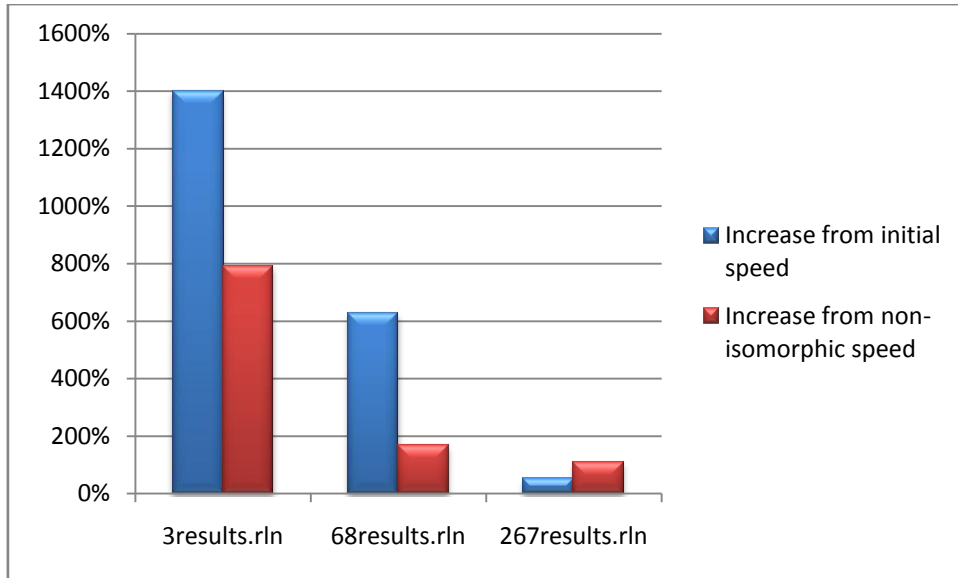


CHART 9 - OVERALL SPEED INCREASE

5.3 NUMBER OF SOLUTIONS

We know that we can reconstruct pedigrees with absolute certainty going one generation back in time. As soon as we go further we might get multiple solutions. We ran several tests where we simulated pedigrees of varying sizes and recorded the number of solutions that the reconstruction algorithm returned when trying to reconstruct the simulated pedigrees. Population size was kept constant while the partner affinity parameter was set to 0.8. The results of those tests can be found in appendix 8.1.

We calculated the mean and standard deviation for each data set returned by the tests. On chart 16 we see how these two values grow as the depth of simulated pedigrees is increased.

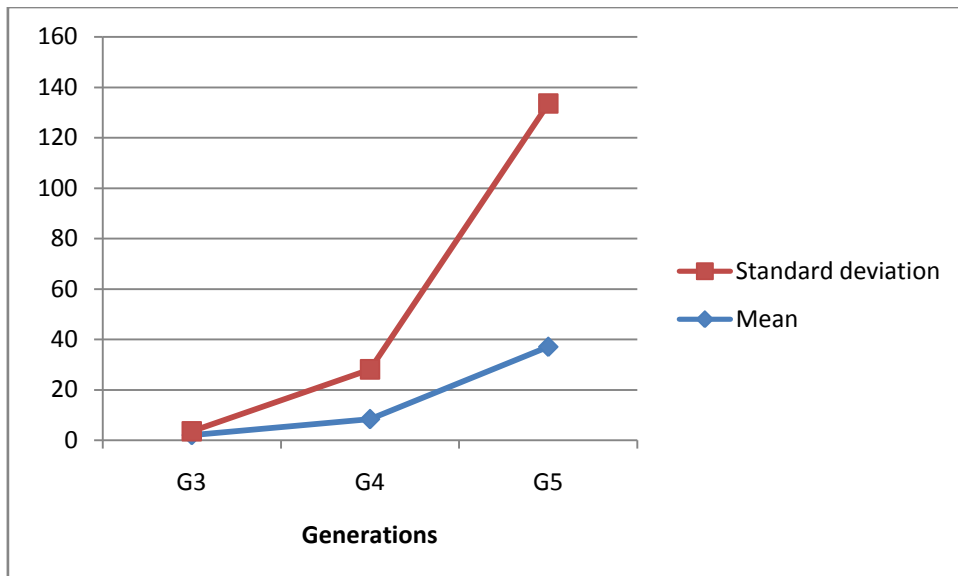


CHART 10 - STANDARD DEVIATION AND MEAN GROWTH (POPULATION SIZE 4)

The mean increases considerably as we add generations and we are also seeing an increase in the standard deviation and max value. This also applies to the case when we increase the population size. Since we are unable to run our program for large pedigrees we cannot say with absolute certainty that this increase in mean and standard deviation will continue but all the evidence we have point to that conclusion. An interesting fact is that in all our runs, 1 solution is the most frequent value. That is a positive sign.

5.4 EXECUTION TIME

Studying the execution time of the algorithm as a function of pedigree depth (number of generations) and width (population size) is of interest. While we do not provide a bound on the algorithm's growth rate, there's some evidence that points to exponential growth for the depth and super-exponential growth rate for the width.

Let's start by considering the width of the pedigree. For each generation the reconstruction algorithm generates all partitions of a set of size n where n equals the number of individuals in the generation². The number of partitions of a set is called a Bell number in honour of Eric Temple Bell (10). Bell numbers can be calculated using the following recursion formula where $B_0 = B_1 = 1$.

$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k$$

On chart 17 we see the first 20 Bell numbers plotted on a logarithmic scale. The line is curved upwards which indicates super-exponential growth.

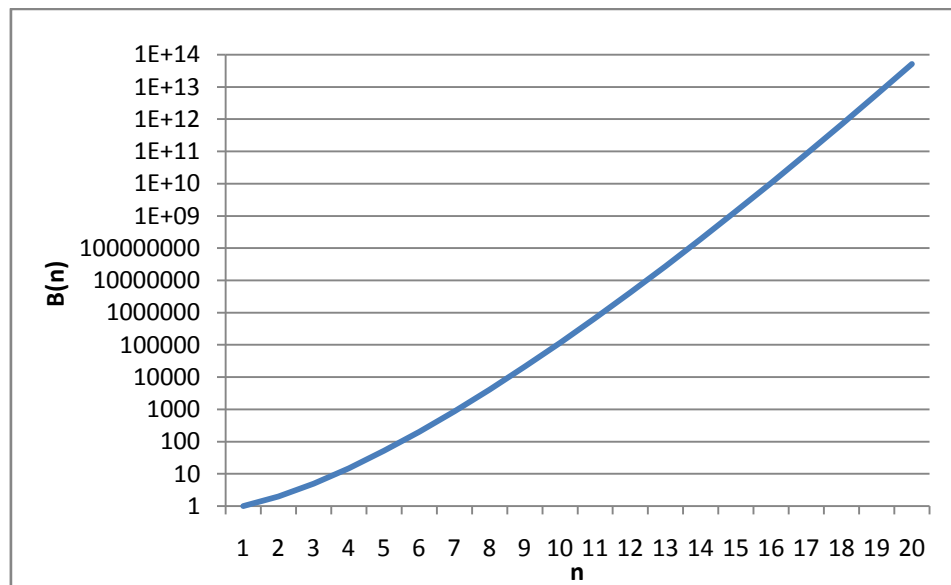


CHART 11 - BELL NUMBER GROWTH

In order to estimate the growth as a function of pedigree depth we look to a bound proposed by Bhalchandra D. Thatte (11) on the number of distinct (mutually non-

² Assuming that the TS-graph of the generation only contains a single connected component which makes it impossible to split the generation up into smaller independent subsets.

isomorphic) pedigree cores³ in a discrete generation pedigree with a constant population size. If n is the width of the pedigree, d is the depth of the pedigree and $g(n, d)$ is the number of distinct pedigree cores then the bound is as follows:

$$g(n, d) \leq \frac{n^{2nd+1}}{n!^d 2^{nd/2}}$$

This bound grows exponentially as d (the depth) increases. It seems logical to assume that the reconstruction algorithm also has an exponential growth rate for d since it's in essence a brute-force algorithm that tries all possible pedigrees in search of matches.

To gather some empirical data we ran a series of tests. In our tests we simulated pedigrees of varying sizes, extracted pair-wise distances from them and fed them to the pedigree reconstruction algorithm while recording the time it took the algorithm to finish execution. We ran the algorithm 100 times for each pedigree size. The following charts display our results.

Note that the best case time for 2 generations is missing from chart 18 since 0 does not fit onto a logarithmic scale.

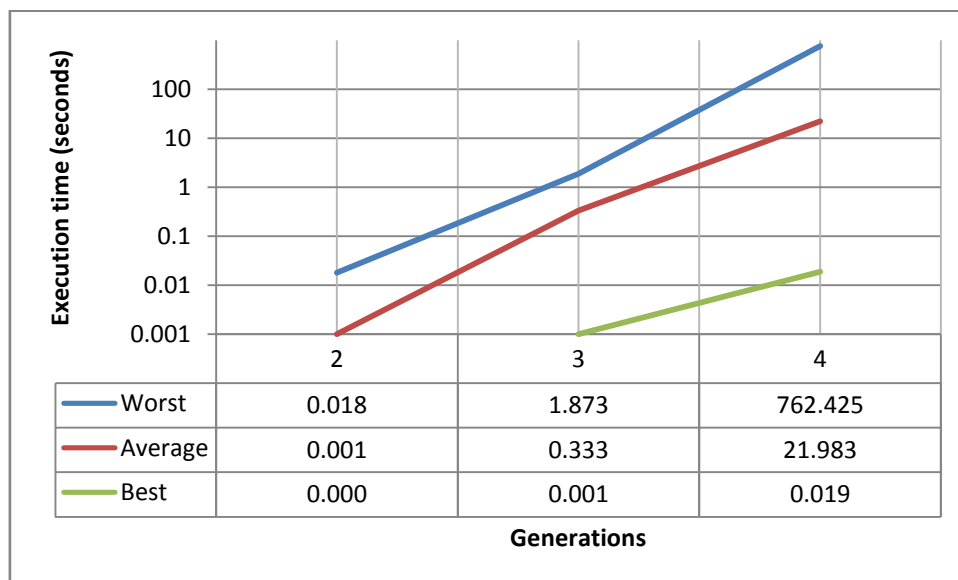


CHART 12 - EXECUTION TIMES FOR POPULATION SIZE 6

On chart 18 we see super-exponential growth for worst-case execution times as the depth of the pedigree increases. The average times exhibit sub-exponential growth. Here we expected to see strict exponential growth.

³ A pedigree core consists of the extant individuals and their ancestors. Individuals that don't have extant descendants are not part of the pedigree core.

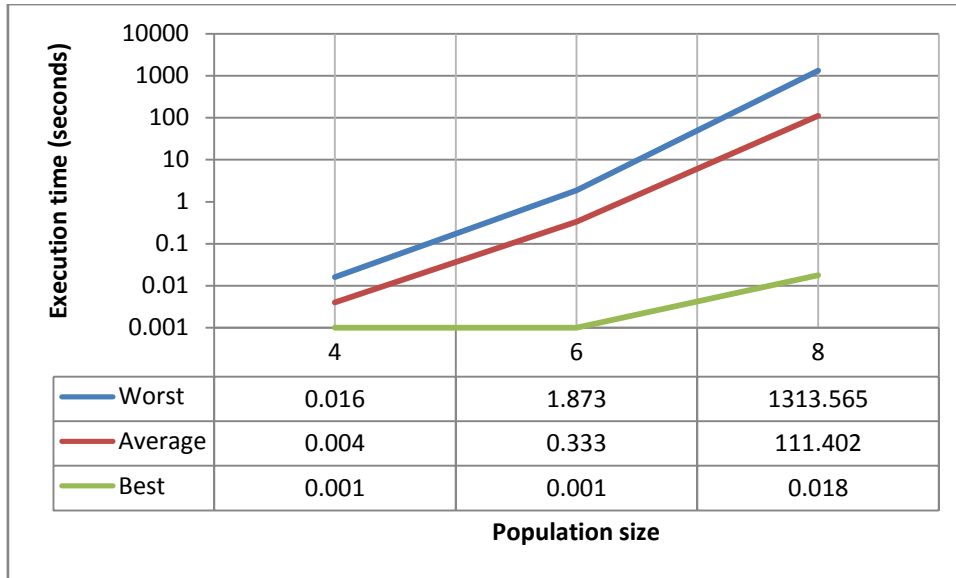


CHART 13 - EXECUTION TIMES FOR 3 GENERATIONS

The test results on chart 19 indicate super-exponential growth for both the worst-case and average times as the width (population size) increases. These results are in line with our expectations.

6 CONCLUSION AND FUTURE WORKS

During our first week in Oxford we focused on creating the pedigree simulator. The week after that we started working on defining the input data; what information could theoretically be extracted from comparing genomic data. Once that was out of the way we sat down with pen and paper and tried to reconstruct tiny pedigrees, hoping that by doing so we would come up with a logical and intuitive method for solving pedigree reconstruction problems. For two or three days we did nothing else yet had nothing to show for it. We gained neither insight nor intuition which suggested that a simple elegant solution method exists. It may be due to our limited capabilities or it may be a sign of pedigree reconstruction being a hard problem. Perhaps both are true.

Eventually we settled on a brute-force method, one that tries generating lots of different pedigrees and then checks which ones are in agreement with the input data. That's the algorithm that we implemented and present in this paper. Coming up with an algorithm that reconstructs pedigrees may be a win in itself but we can't help feeling disappointed by just how small the pedigrees have to be in order for the algorithm to finish execution in a reasonable amount of time. We found that simulated pedigrees that are 6 individuals wide and 4 generations deep are pretty much the biggest pedigrees that can reliably be reconstructed. Going beyond that gave us mixed results; sometimes we would get a result pretty quickly but more often than not we gave up and stopped the program. The connected-components optimization allowed us to reconstruct wider pedigrees (with up to 20 individuals) if we limited the depth to 2 generations.

Bhalchandra D. Thatte suggested a couple of performance improvements that could be made to our reconstruction algorithm but time restrictions kept us from pursuing them. Both are related to set-partitions.

When we implemented the algorithm we created our own method for generating all partitions of a set. It is both crude and inefficient. Thatte pointed out to us that there are known algorithms that solve this exact problem in an efficient manner. We found two articles describing fast set-partition generation algorithms, one by M. C. Er titled 'A Fast Algorithm for Generating Set Partitions' (12) and the other by B. Djokic et al. titled 'A Fast Iterative Algorithm for Generating Set Partitions' (13). Substituting our set-partition method for an algorithm based on either of these articles should increase the performance of the pedigree reconstruction algorithm.

The second improvement suggested by Thatte is to label each individual by concatenating the labels of the individual's extant descendants (just like when labelling nodes in TS-graphs) and then only generate set-partitions for distinct labels. For example if we have six individuals that have been labelled {AD, AD, BEF, BEF, C, C} we would only generate partitions of the set {AD, BEF, C}. This would drastically reduce the number of set-partitions. On the other hand, some "unrolling" of the set-partitions would be required instead. If a subset contains a label shared by multiple individuals then the algorithm must try sibling groups that contain different numbers of individuals with that label. It's still a very promising idea. Another expected benefit is that this improvement should reduce the number of isomorphic pedigrees generated.

During our final status meeting with our instructors at Reykjavík University a couple of other ideas surfaced that might be worth considering.

The first one is to add a parameter to the reconstruction algorithm that specifies a margin for population size growth. Imagine that this parameter were set to 20%. That would instruct the reconstruction algorithm not to consider pedigrees where the increase or decrease in population size from one generation to another exceeds 20%.

The second idea is to add a parameter that specifies the maximum gender ratio. If this ratio parameter were for example set to 2:1 then the algorithm would rule out all pedigrees that contained a generation where members of one gender outnumbered the others by a factor greater than 2:1.

We knew from the get-go that we would not be able to deterministically reconstruct pedigrees using the input data that we chose. The interesting question was always going to be how often would we get multiple pedigrees and how many. Based on our results the uncertainty grows as the pedigree being reconstructed grows larger. Even still, one solution was the most common result in our tests so there's some comfort in that. Whether that's also true for pedigrees approaching real-life population sizes remains unanswered.

It might be of interest to study pedigree reconstruction using input data that contains more information than just pair-wise distances. With pair-wise distances we only know that two extant individuals shared a common ancestor some time ago. It may very well be that the same ancestor had more extant descendants but it's often impossible to tell just by looking at pair-wise distances. If the input data contained n -tuple-wise distances then we would know that n extant individuals shared a common ancestor some time ago. Such information would rule out all pedigrees that do not include an ancestor of exactly those n individuals at the specified distance and therefore reduce the number of possible solutions.

6.1 ACKNOWLEDGEMENTS

We thank Jotun Hein and Rune Lyngsø for guiding us through the project. Bhalchandra D. Thatte provided many insightful comments on our project and this report for which we are indebted. We would also like to thank Bjarni Halldórsson, Marta Lárusdóttir and Yngvi Björnsson at Reykjavík University who collectively steered us through the last few hurdles.

We are especially grateful to Luca Aceto for recommending us for the project and cutting through the red tape.

The six weeks we spent working on the project in Oxford were highly enjoyable. This was largely due to a brilliant group of fellow summer project students. We would like to thank them for making this summer unforgettable.

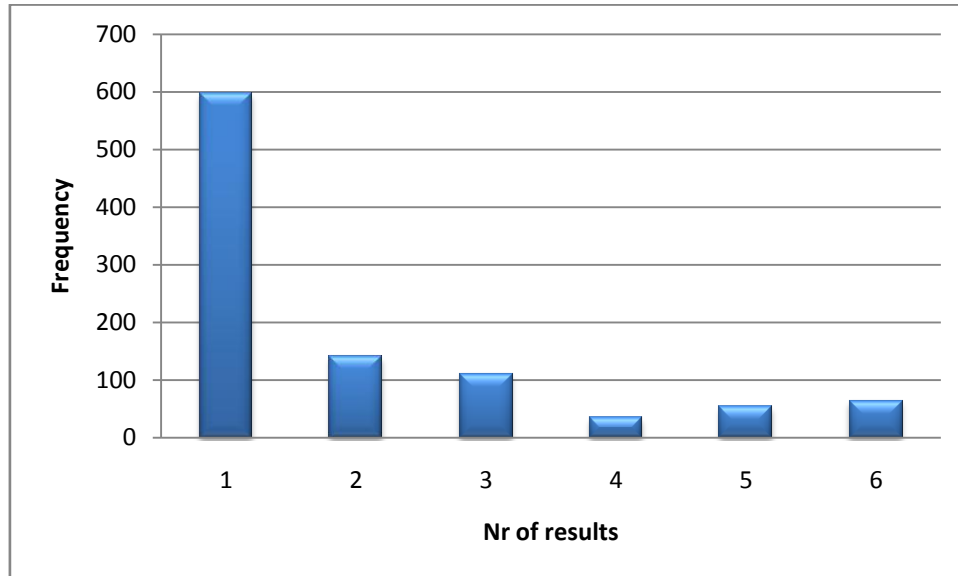
7 REFERENCES

1. *Reconstructing pedigrees: A combinatorial perspective*. **Steel, Mike and Hein, Jotun**. 3, 7 June 2006, Journal of Theoretical Biology, Vol. 240, pp. 360-367.
2. **Hein, Jotun and Lauritzen, Steffen**. *Combinatorial Pedigree Inference from Genomic Data*. Oxford : s.n., 2008.
3. **Wikimedia Foundation, Inc**. Wikipedia. *Partition of a set - Wikipedia, the free encyclopedia*. [Online] [Cited: 4 September 2008.] http://en.wikipedia.org/wiki/Partition_of_a_set.
4. *Isomorphism of graphs of bounded valence can be tested in polynomial time*. **Luks, Eugene M**. 1982, Journal of Computer and System Sciences, Vol. 25, pp. 42-65.
5. **Wikimedia Foundation, Inc**. Wikipedia. *Graph isomorphism - Wikipedia, the free encyclopedia*. [Online] [Cited: 8 September 2008.] http://en.wikipedia.org/wiki/Graph_isomorphism.
6. *Linear time algorithm for isomorphism of planar graphs*. **Hopcroft, John and Wong, J**. 1974, Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, pp. 172-184.
7. **Darrell, P**. Codeproject. [Online] [Cited: 20 August 2008.] <http://www.codeproject.com/KB/recipes/VFIsomorphism2.aspx>.
8. *A Performance Comparison of Five Algorithms for Graph Isomorphism, Proceedings of the 3rd IAPR-TC15 Workshop on Graph based Representation (GbR2001), Italy*. **Foggia, P, Sansone, C and Vento, M**. 2001.
9. *Performance Evaluation of the VF Graph Matching Algorithm*. **Cordella, L.P., et al**. Venice : s.n., 1999. Image Analysis and Processing, 1999. Proceedings. International Conference on. pp. 1172-1177.
10. **Wikimedia Foundation, Inc**. Bell number. *Wikipedia, the free encyclopedia*. [Online] [Cited: 19 09 2008.] http://en.wikipedia.org/wiki/Bell_number.
11. **Thatte, Bhalchandra D**. *Personal communication*. Oxford, 2008.
12. *A Fast Algorithm for Generating Set Partitions*. **Er, M. C**. 3, 1988, The Computer Journal , Vol. 31, pp. 283-284.
13. *A Fast Iterative Algorithm for Generating Set Partitions*. **Djokic, B, et al**. 3, 1989, The Computer Journal, Vol. 32, pp. 281-282.
14. *Combinatorics of Pedigrees I: Counterexamples to a Reconstruction Question*. **Thatte, Bhalchandra D**. 3, 21 May 2008, SIAM Journal on Discrete Mathematics, Vol. 22, pp. 961-970.
15. *Reconstructing pedigrees: A stochastic perspective*. **Steel, Mike and Thatte, Bhalchandra D**. 3, 7 April 2008, Journal of Theoretical Biology, Vol. 251, pp. 440-449.

8 APPENDIX

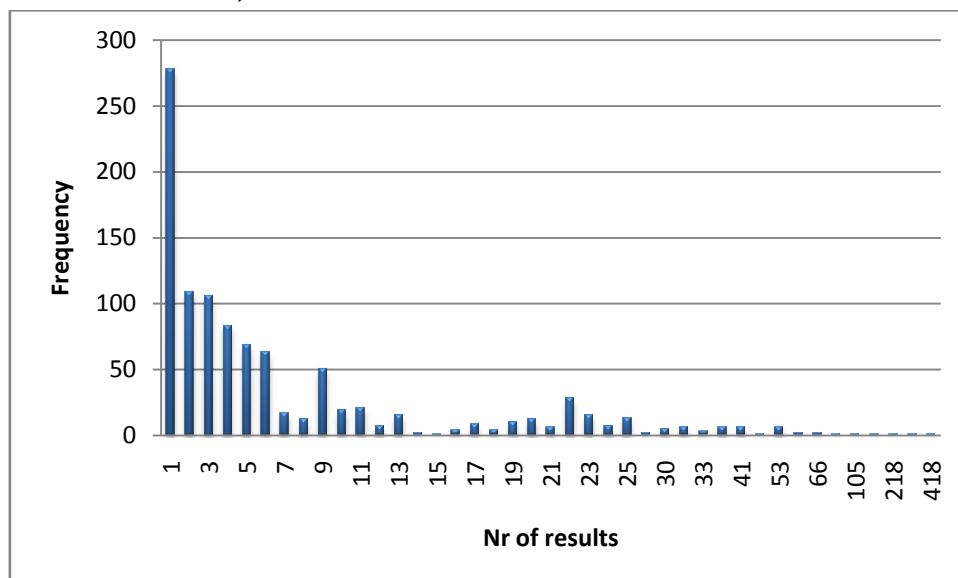
8.1 NUMBER OF SOLUTIONS – TEST RESULTS

8.1.1 3 GENERATIONS, POPULATION SIZE 4



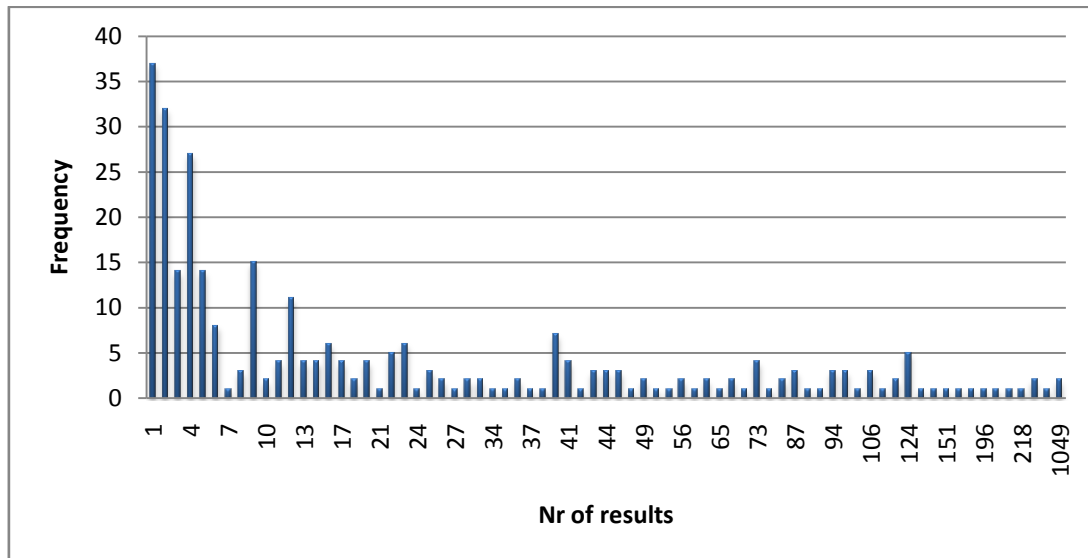
Total runs:	1000
Mean:	1.999
Standard deviation:	1.533
Min:	1
Max:	6
Most frequent value:	1

8.1.2 4 GENERATIONS, POPULATION SIZE 4



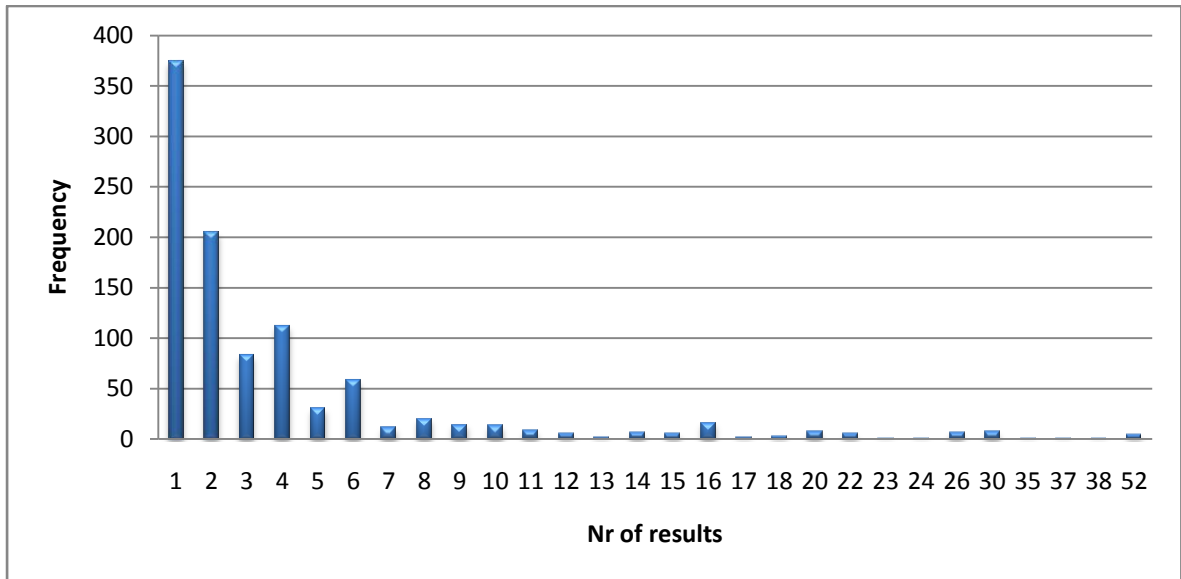
Total runs:	1000
Mean:	8.329
Standard deviation:	19.760
Min:	1
Max:	418
Most frequent value:	1

8.1.3 5 GENERATIONS, POPULATION SIZE 4



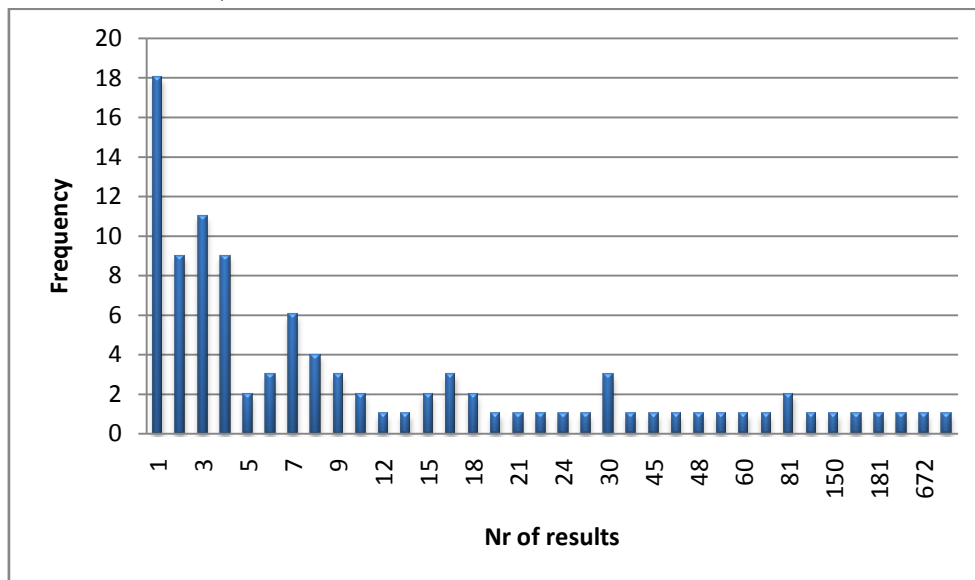
Total runs:	300
Mean:	37.020
Standard deviation:	96.438
Min:	1
Max:	1049
Most frequent value:	1

8.1.4 3 GENERATIONS, POPULATION SIZE 6



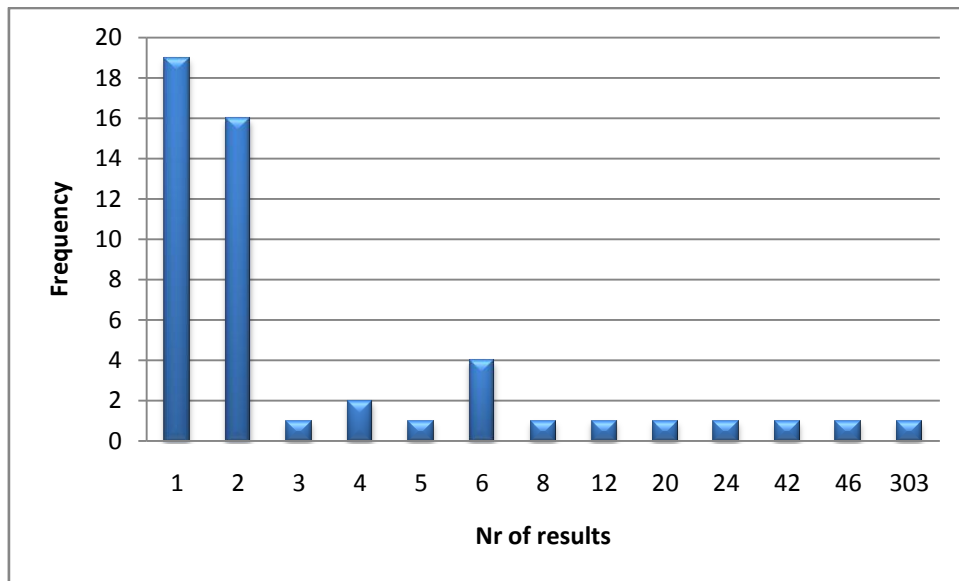
Total runs:	1000
Mean:	4.128
Standard deviation:	5.943
Min:	1
Max:	52
Most frequent value:	1

8.1.5 4 GENERATIONS, POPULATION SIZE 6



Total runs:	100
Mean:	69.560
Standard deviation:	408.934
Min:	1
Max:	4016
Most frequent value:	1

8.1.6 3 GENERATIONS, POPULATION SIZE 8



Total runs:	50
Mean:	10.920
Standard deviation:	43.143
Min:	1
Max:	303
Most frequent value:	1

8.2 TABLE REFERENCE

Table 1 - Example input data (limited to distance ≤ 2) 12

Table 2 - Test file used for benchmarking..... 23

Table 3 - Compare on the go: benchmark results 25

Table 4 - Allowed pairings: benchmark results 26

Table 5 - Path cache: benchmark results 26

Table 6 - Connected components: benchmark results 27

Table 7 - Reverse isomorphism checks: benchmark results 28

Table 8 - Required pairings: benchmark results 29

Table 9 - Labels in isomorphism checks: benchmark results 30

Table 10 - Overall benchmark results 31

8.3 FIGURE REFERENCE

Figure 1 - An ancestral family tree 3

Figure 2 - A descendant family tree 4

Figure 3 - A mock phylogenetic tree. LUCA stands for Last Universal Common Ancestor. 4

Figure 4 - Founders at the top and extant individuals at the bottom 6

Figure 5 - Steel's and Hein's counter-example 7

Figure 6 - Isomorphic pedigrees 8

Figure 7 - An example of two graphs that are digraph isomorphic but not pedigree isomorphic..... 8

Figure 8 - A pedigree graph which cannot be gender labelled 14

Figure 9 - An individual with only three grand-parents.....	14
Figure 10 - One common ancestor, three paths.....	15
Figure 11 - One common ancestor, two node-disjoint paths.....	15
Figure 12 - Pedigree Simulator	18
Figure 13 - Simulated pedigree.....	19
Figure 14 - Reconstruction data extracted from simulated pedigree.....	19
Figure 15 - Reconstructed pedigree	20
Figure 16 - A pedigree graph that contains a subdivision of $K_{3,3}$	22
Figure 17 - TS-graph example	24

8.4 CHART REFERENCE

Chart 1 - Compare on the go: execution times.....	25
Chart 2 - Allowed pairings: execution times	26
Chart 3 - Path cache: execution times	27
Chart 4 - Connected components: execution times	28
Chart 5 - Reverse isomorphism checks: execution times	29
Chart 6 - Required pairings: execution times	30
Chart 7 - Labels in isomorphism checks: execution times.....	31
Chart 8 - Overall execution times.....	31
Chart 9 - Overall speed increase.....	32
Chart 16 - Standard deviation and mean growth (population size 4)	32
Chart 17 - Bell number growth	33
Chart 18 - Execution times for population size 6	34
Chart 19 - Execution times for 3 generations	35