Dublin Institute of Technology
Institiúid Teicneolaíochta Átha Cliath

Image Recognition of Plant Leaves for Species Identification

Student Name: Hugh Pearse
Student Number: C08455457
Supervisor: Mr. Paul Bourke

25th November 2011

Acknowledgements

I would like to thank my project supervisor Paul Bourke and all of the staff in the Dublin Institute of Technology for the help and guidance they gave me.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university or institution.

Contents

List of Figures

Abstract

The world contains countless hybrids and unusual combinations of living things. There are interspecific hybrids such as the Liger and even an interkingdom hybrid between the sea slug "Elysia chlorotica" and algae. Hybrids are not under threat from extinction but true species face the damages of environmental impact every day and there is an urgency to document their numbers and locations so that a methodical conservation plan can be devised. We have entered an age where identifying and tracking these endless combinations can be accelerated with software. In this document is described a series of techniques for extracting plant leaf biometrics from a photograph and indexing them in a search engine.

Chapter 1

Introduction

1.1 Problem

The botanic gardens around the world are widely considered the key to conserving rare species. This has recently been demonstrated with the species "Wollemia nobilis" as it was thought to be extinct until recently and can now be found in botanic gardens all over the world. A person doing field research in botany cannot simply find an unusual plant and dig it up to put it on show in the botanic gardens. Restrictions have been put in place in order to protect the ecosystem from environmental damage. This means that the only people who have the authority to submit plant specimens are people educated in the scientific field of botany. The restriction on submission of plant specimens to the botanic gardens means that a large amount of work is now the responsibility of those working in that specialist field. Efforts must be made in order automate their laborious and often tedious work which can be time consuming and reduces productivity.

Botanic gardens and herbariums are quick to share their collections with researchers abroad but this is a slow process. As a result these organizations have started creating high resolution photographs of type specimens and making them publicly available. This facilitates faster research but in addition to images of type specimens, it is extremely useful to obtain images of isolated leaves. Unfortunately most herbariums have not yet completed a full digital catalogue and have only catalogued a small portions of their collections. For example the plant collections of the Smithsonian Institution (one of the largest on earth) amount to 4.8 million historical plant records to date but only 90,000 type specimens with images are currently available in their online catalog[3]. Similarly Kew gardens have 7 million specimens either pressed and dried or preserved in spirit but only 119,386 digital images of type specimens[4].

1.2 Approach

The biometrics can be extracted from leaves using a combination of reliable image manipulation techniques. Segmentation is a difficult step that reliably separates the leaf from the background of the image. In a scientific environment this process is made easier because we can rely on the users to take images in a

way that suits the software by using a uniform background. Cropping the image so that the border fits against the side of the leaf is made easier by a clean segmentation. Any noise that has not been removed from the segmentation could potentially cause cropping to give a false result. After the segmentation and cropping the biometrics can easily be extracted without much additional effort.

There are several identification schemes commonly used by biologists such as a diagnostic key, a multi-access key and the comparison method but there are two principal techniques. The monothetic identification methods are methods where one characteristic is used at a time in a sequence to identify a species. Polythetic methods are ones where several characteristics are used simultaneously. For any polythetic identification method which calculates a measure of similarity, it is possible to use a numerical weighting for each biometric in the hope of improving the results.

1.3 Solution

To start it is necessary to obtain a collection of images of isolated leaves. There are several existing projects that have made parts of their datasets publicly available such as Flavia, Leaf Snap and Leaf-Recognition. These datasets are different in terms of their focus on content. Some datasets have a small collection of species but a large collection of specimens, other datasets have more species but only a single image to represent every possible variety. Both of these variations can be useful for providing content to the database and for calculating the fuzz tolerance for a search. The best approach to take in order to compile an accurate database is to get as many specimens as possible. In order to do this, the community moderated approach used by Wikipedia for providing content is by far the fastest.

Chapter 2

Research

2.1 Methodology

The methodology used in this project is a cross between the agile and the scrum design methodologies. The combination of the methodologies was necessary because the project is being taken through every phase by a single developer and more flexibility is required. It is separated into eight steps. The first step is the feasibility research. This step involves checking if a major change in the application is technically feasibile to implement within a reasonable amount of time. It also aims to establish if the change is needed and if it would be useful. The second step is the core research. This step is a more in depth research period in which the designs of existing systems are analysed. The third step is design which involves documenting individual parts of the system at different levels of detail and the entire system as a whole. The fourth step is the prototype stage. The prototype stage is where the core development is done but has not yet been tested. Stage five is the testing phase. Stages six seven and eight are release, maintain and deprecate.

Different stages are grouped together. Working outwards: design, prototype and test belong to the Rapid Application Development stage which is the only iterative stage and iterates until the feature is ready. The second stage consists of stage one, release and maintain called the Minor Feature stage. This is for adding minor features which do not require a large amount of planning. The third and final stage is called Major Feature. This consists of feasability research, core research, Feature stage and deprecate. This is used for introducing major features into the product and deprecating them after their lifespan.

1. Rapid Application Development
    (a) Design
    (b) Prototype
    (c) Testing
2. Minor Feature
    (a) Design
    (b) Prototype

(c) Testing
(d) ***Release**
(e) ***Maintain**

3. Major Feature
   (a) ***Feasability Research**
   (b) ***Core Research Design**
   (c) Design
   (d) Prototype
   (e) Testing
   (f) Release
   (g) Maintain
   (h) ***Deprecate**

2.2 Version Control

The version control being used for this project is Git. It was initially developed by Linux kernel creator Linus Torvalds. Git offers a different type of version control than CVS because it is a distributed version control system. With a distributed version control system, there isnt one centralized code base to pull the code from. Different branches hold different parts of the code. Other version control systems, such as SVN and CVS, use centralized version control, meaning that only one master copy of the software is used[5].

2.3 Existing Systems

There are a variety of existing projects that attempt to do image recognition of plant leaves. Examples of some of them are Flavia, Leaf Snap, Project Maple, Leaves Recognition, ePlantAnalyzer, leaf-recognition and kfulles. Each of these tools take different approaches to implementing the same concept of leaf recognition for plant leaves.

There are also lots of alternative content based image retrieval systems that do not focus on any specific topic but crawl the internet for more content such as MIFile and the well known Tineye. MIFile is open source and adopts the concept of searching for images using metadata by creating a search key. By doing the metadata extraction as the image is being indexed this allows them to locate similar results for a search request without performing heavy computation while the user waits. Using this technique they give nearly instant results which improves the user experience.

Flavia	`http://flavia.sourceforge.net/`
Leaf Snap	`http://leafsnap.com/`
Project Maple	`https://github.com/johnflan/Maple`
Leaves Recognition	`http://sourceforge.net/projects/lrecog/`
leaf-recognition	`http://code.google.com/p/leaf-recognition/`
kfulles	`http://code.google.com/p/kfulles/`
MIFile	`http://mi-file.isti.cnr.it/CophirSearch/`

2.4 Existing Datasets

Figure 2.1: Malus angustifolia type specimen

The problem with herbariums digitizing their catalogues is that their motive is to facilitate botanists and researchers who would otherwise have to borrow a type specimen which would require waiting a significant amount of time. The herbariums are trying to anticipate what the data is being used for, rather than providing it in a variety of formats.

The format of plant type specimens currently being photographed by herbariums contains a lot of additional data that makes segmentation much more difficult. It contains multiple leaves, branches, flowers a scale and a colour palette. Ideally for creating a database for leaf recognition it would be useful to find a database containing images of isolated leaves. A similar project for image recognition of leaves called Leaf Snap which was developed by the Smithsonian Institution encountered the same problem. Luckily the Leaf Snap project received a grant from the US government so they hired a contractor "Finding Species" to locate, identify and photograph tree leaves, flowers and seeds[3]. These images have a uniform background suitable for segmentation. There are 164 of these images available on the Leaf Snap website at an approximate resolution of 1000 pixels in height and width depending on the dimensions of the leaf.

Datasets used in similar projects:

Flavia Dataset (33 species, 1GB): http://sourceforge.net/projects/flavia/files/
Leaf-Recognition Dataset (92 species[silhouettes], 4MB): http://leaf-recognition.googlecode.com/files/dataset.zip
Leaf Snap (164 species, 13MB) http://leafsnap.com/species/

2.5 Heuristics

In order to be able to accurately review different software applications a set of important attributes must be taken from the software requirements and attributes common to each software package.

1. Requirements
   (a) Compile custom database
   If this product it being aimed at researchers they will want the flexibility of adding new specimens to the database while they're in the field.
   (b) Perform search
   This is the core feature of the product.

(c) Open Source
This will allow the internals of existing systems to be examined for inspiration and research.

2. Common attributes

(a) Learning requirement
A small amount of time to learn how to use a system is preferable to a large amount of time. If a product is difficult to use then nobody will use it.

(b) Time to install
The amount of time taken to set up a system could be a deciding factor.

(c) Search speed
The performance of the system is a core feature of the product and must be carefully assessed.

Name	Custom	Search	Source	Learning	InstallT	SearchT
*Flavia	y	y	y	hi	hi	?
Leaf Snap	n	y	n	lo	lo	15s
Project Maple	y	n	y	na	na	na
*Leaves Recognition	?	?	?	?	hi	?
leaf-recognition	y	y	y	lo	lo	variable
kfulles	y	n	y	lo	lo	na
MIFile	y	y	y	lo	?	4s

2.6 Biological classification

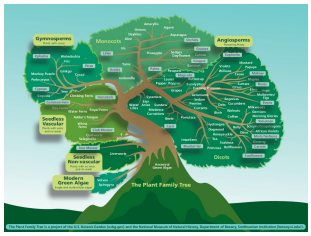

Figure 2.2: Plant Family Tree[1]

There are several different systems that have been used historically for representing relationships between different plants and animals. The concept of living organisms sharing a common ancestor is not a new idea but it is widely accepted that Charles Darwin was the first person to depict the tree of life when he published "The Origin of Species" in the year 1859. It must be noted that any hierarchical system can be represented in a tree structure in order to act as a visual aid. Darwins representation differs from others in that it was a single unifying tree structure where all living organisms shared a root node which represented a historic species and in addition to that the tree had leaf nodes that represented an extinct or living species depending on its location. Darwins tree also contained branches which was common among older classification systems.

The classification kingdom that concerns an image recognition system for species identification is the entire plant kingdom except for leafless plants such as Mosses, Algae and Liverworts. These must be exempt from the system because they are small and have erratic growth patterns. It must also be noted that plants are normally classified by their reproductive systems and accuracy of results may potentially be improved by using flowers as a metric in addition to leaves[6].

2.7 Biometrics

A metric is simply a form of measurement. Some metrics used by botanists would be embryonic leaf count, leaf venation[7], presence of flowers, plants with cones and finally plants with veins but no seeds. Some metrics used by existing leaf recognition systems and content based image retrieval systems are colour histograms, nearest neighbour classifier, moving median centre hyper-sphere, neural networks, leaf shape ratio. A combination of as many reliable biometrics as possible should produce the most accurate results but will affect performance.

2.8 Plant Leaf Variation

Figure 2.3: Leaf shape variation between wild and cultivated form (Solanum lycopersicum)

Non-uniformity is a factor that must be taken into account when designing a database to catalogue living organisms. Nature is ultimately a chaotic system like weather patterns and some people speculate that DNA flows much more freely between kingdoms than most people realize[8]. A virus can simply take the DNA from one organism and use it to infect another. This variation is the backbone of the cold virus and HIV virus.

Plants that are found in urban environments are much more likely to be a selectively bred and cultivated form. These plants are generally a very poor representation of a species because they have been selected for cultivation for the reason that they are exceptional. Cultivated plants are usually selected for reasons such as high yield, disease resistance and wind resistance but also they can be selected for purely cosmetic features such as variegation or scent.

Some content based search engines create a single key structure composed of sub-keys, each of which are taken from the metadata. This concept is known in the field of Information Retrieval systems as an inverted file. This provides exceptional performance in terms of speed. The inverted file key structure is inappropriate for erratic data because the most significant bit is supposed to be the least erratic metric. Finding these metrics is essentially a research field called taxonomy.

Figure 2.4: Leaf shape and colour variation between cultivars (Acer palmatum)

Figure 2.5: Leaf colour variation over time (Photinia glabra)

Variation has obviously been documented between species and cultivars but it can also happen between two clones from the same cultivar (ie, different branches on the same plant). As plants grow over time, the shape ratio can alter so that the length of a leaf in the autumn will greater than it was in the spring. Certain plants occasionally put out pre-leaves and pre-flowers which have no similarity to the shape of the true leaves; particularly with annual plants which will often retain their cotyledons until the end of the season. Cotyledons do not need to be taken into account as there only normally one or two per plant.

Additionally leaf colour variation is another problem that will be encountered when designing the system. Colour variation is exceptionally common in selectively bred cultivars which can be found in urban environments. Plants such as variegated hollies and Japanese maples are a good example of selectivly bred colour variations.

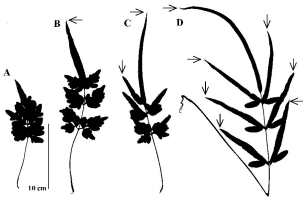

Figure 2.6: Leaf shape variation over time (Pteris mutilata)[2]

Chapter 3

Detailed Design

3.1 Segmentation

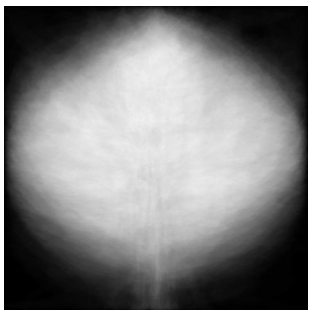

Figure 3.1: Average background of 164 species

Image segmentation is the act of isolating the subject of the photograph from the background or foreground. This is context dependant and difficult to do reliably. The simplest solution is to ensure that all photographs are taken against a uniform background to minimise the amount of noise in the image.

The background colour can be either hard coded into the segmentation tool but it is good design to write reusable code[9]. For this reason the segmentation program should accept a background colour as a parameter. The subject of the image is usually located in the centre area and the background is usually located in the corners. In order to generate an activity map to locate the most used and least used parts of the images, the images must all be resized into the same extent (width and height). Using this averaged data we can see which areas of the images can be reliably sampled. The colours can be averaged from the four corners of the image and removed from the image.

```
bash-4.1# convert *.png -average avg.jpg
```

Figure 3.2: Image Average in ImageMagick

```
bash-4.1#
convert -transparent '#000000' -fuzz 10% infile.png outfile.png
```

Figure 3.3: Segmentation in ImageMagick

3.2 Colour Information

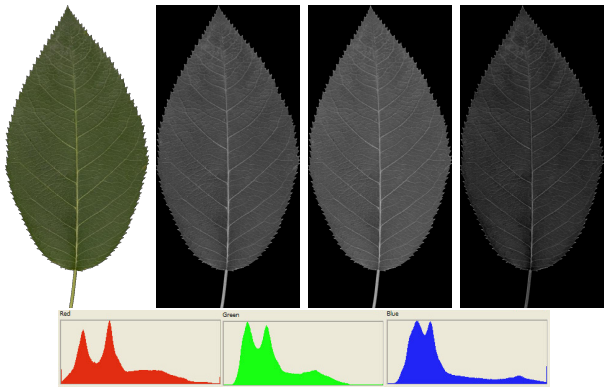

Figure 3.4: Image is split into red, green and blue colour channels

The colour information can be easily extracted from a leaf as a colour image is represented on the screen as red, green and blue. The image is first segmented from its background colour and then each channel from the image is extracted to a separate file. Each channel is now represented as a grayscale image and a histogram is created from the colour data.

```
bash-4.1# convert infile.png -separate -channel RGB outfile.png
```

Figure 3.5: Colour Channel Splitting in ImageMagick

Figure 3.6: Patented ColorChecker Color Rendition Chart

Be aware that the colour information is simply image brightness in different colour channels. This means that the ambient luminosity of the location in which the photo is taken can drastically alter the leaf colour information. As a precaution leaf colour information should be weighted less favourably than shape information, unless the ambient lighting conditions are strictly controlled by using a lighting rig or a ColorChecker Color Rendition Chart.

```
bash-4.1# identify -verbose outfile-0.png
bash-4.1# identify -verbose outfile-1.png
bash-4.1# identify -verbose outfile-2.png
```

Figure 3.7: Colour Histogram in ImageMagick

Name	Red Deviation	Green Deviation	Blue Deviation
Acer platanoides	61.5105	52.2323	88.9521
Halesia tetraptera	38.2732	43.0351	20.6656

Figure 3.8: Acer platanoides var. drummondii and Halesia tetraptera

3.3 Colour Deviation

This can be calculated by taking the colour values of all the pixels within an image, then for each colour value subtract the average and square the result. This gives us the squared difference for each colour value which produces an output that is more sensitive to larger values. Finally work out the average of the squared differences which gives us the standard deviation. The standard deviation is larger if a leaf has a high contrast between its colours. An example of a leaf with a high standard deviation would be a variegated variety.

```
bash-4.1# identify -verbose input.png
```

Figure 3.9: Colour Deviation in ImageMagick

Algorithm 1 (Average)

$\frac{600+470+170+430+300}{5} = \frac{1970}{5} = 394$

Algorithm 2 (Deviation)

$|(600-394)| + |(470-394)| + |(170-394)| + \ldots = 206 + 76 + 224 + 36 + 94$

Algorithm 3 (Absolute Deviation)

$\frac{206+76+224+36+94}{5} = \frac{636}{5} = 127.2$

Algorithm 4 (Standard Deviation)

$\sqrt{\frac{206^2+76^2+224^2+36^2+94^2}{5}}$

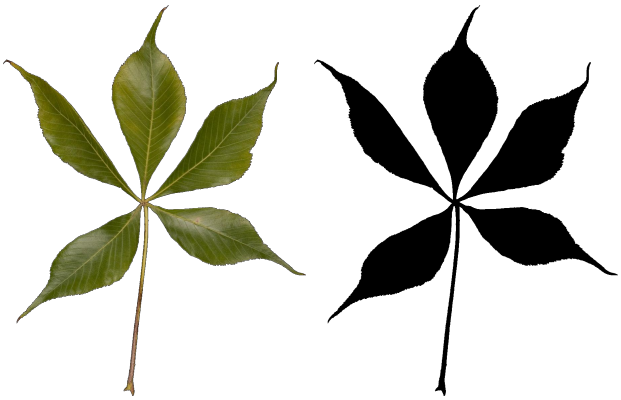

Figure 3.10: Aesculus glabra Conversion to Silhouette

```
bash-4.1# convert input1.png -channel 'RGB'
 -black-threshold 96% -colorspace "Gray" output1.png
bash-4.1# convert output1.png -background
 "#FFFFFF" -alpha Background -alpha Off output2.png
```

Figure 3.11: Extract deviation information

3.4 Surface Area

Size of a leaf cannot be extracted from an image without a reference scale being present in the image as well. However adding a reference scale would affect leaf segmentation. The simplest solution is to take the leaf size as a ratio of the total cropped image size. This can be calculated by first segmenting the leaf from the image background and cropping the image so that the leaf is touching the edge of the image. Then a silhouette is created from the image so that the uniform background colour contrasts with the leaf silhouette. If the leaf solhouette is black and the background colour is white then a simple count of the black and white pixels can represent the ratio of the leaf to image.

3.5 Leaf Perimeter

After the leaf has been segmented from its background, edge detection can be performed on the image. The edge detection algorithm begins by taking the colour at the side of the image (which is an alpha channel) as the background; the image is then traversed horizontally and when an non transparent pixel is encountered the previous pixel is marked and the algorithm continues to the opposite side of the image and continues to mark contrasting areas. This process is repeated for every row in the image.

The metric for calculating the serration is simple because the pixel width used for the edge marking is one pixel. This allows us to simply count the number of white pixels in the leaf edge image. If the leaf is serrated along the edges or highly divided like a fern, then the number of pixels representing the edges will be at a higher ratio to the number of pixels in the surface area of

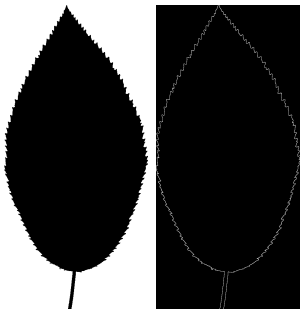

Figure 3.12: Aesculus glabra Conversion to Silhouette

the leaf. A leaf with a perfectly effecient use of surface area with be perfectly smooth around the edges and will be perfectly round.

```
bash-4.1# convert input.png -edge 1 output.png
```

Figure 3.13: Convert image silhouette for perimeter calculation

3.6 Leaf Shape

The shape of the leaf can be measured in a similar way that edges are detected. The algorithm begins on the top left hand side of the image and the program checks if the pixel is empty or is part of the leaf. The program iterates across the image from left to right and when a pixel that is part of the leaf is encountered then the location is marked and the program performs the same operation on the adjacent row beneath. The resultant distance from the edge of the image to the leaf is the larger measurement of the two rows. This means that the calculation can tolerate some errors introduced to noise in the image.

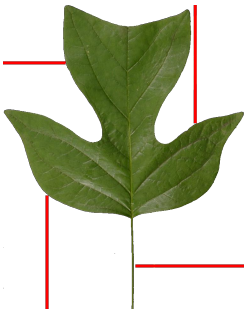

Figure 3.14: Shape extraction example

This can be done for every pixel along the border of the image except it would require a large amount of computation. The simplest solution to this problem is to perform one measurement along each of the four edges of the image. There is a problem with doing this, if the measurements are taken from

the middle of each edge where the image extent is likely to be cropped to the edge of the leaf this will produce a reading of zero for each side because of the leaf symmetry. The solution to this is to take each reading uniformly off centre from the middle of the image border.

3.7 Width and Height

There are a few different approaches to using the width and height of the leaf as a metric for searching in the database. For example if the image is taken at an angle, parallax will be introduced which will distort the reading. This project assumes that all photos are taken correctly and that the images are cropped to the edges of the leaves. This means that we can use width and height ratio as a metric.

```
bash-4.1# identify -verbose input.png
```

Figure 3.15: Extract image width and height information

3.8 Inverted Files

Excerpt from The Art of Computer Programming, Vol. 3 (Donald Knuth)[10]:
An inverted file does not mean that the file is turned upside down; it means that attributes of records can be summarized into manageable data. For example, the inverted file corresponding to a Russian-English dictionary is an English-Russian dictionary. In general, an inverted file usually doesn't stand by itself; it is to be used together with the original non-inverted file. It provides duplicate, redundant information in order to speed up secondary key retrieval. The components of an inverted file are called inverted lists, namely the lists of all records having a given value of some attribute.

Excerpt from Practical Taxonomic Computing (Richard J. Pankhurst)[11]:
The method of comparison, or matching, is straightforward in principle. The unknown specimen is first examined and its characters are observed or investigated by experiment. It is then compared with all of the taxa of an appropriate group and identified with whichever of these agrees exactly, or is the most similar. The comparison of the specimen with the reference taxa may involve counting the number of characters which agree, or calculating some measure of the agreement (a similarity coefficient) based on the character matches and mismatches. The result of the comparison may also be a short list of taxa rather than just one, from which a subjective choice must be made.

Comparison methods have the advantage that they do not demand any particular characters from the specimen and can be used successfully with incomplete material. The characters are used all together, polythetically and as the outcome does not depend on the correct observation of every character, some errors can be tolerated. However, the results become more reliable as more characters are used. If there are many taxa to consider, there may be much labour involved in comparing the specimen with every taxon.

Search Request: 4, 2.5, 9

Colour	Serration	Surface Area	Total Deviation
5	**3**	**8**	2.5
7	4	7	6.5
2	6	6	8.5

Figure 3.16: Example of a simple search request of an inverted file database

3.9 Fuzz Levels

The fuzz level is the amount of error tolerance that must be set when selecting rows from the database. For example if a user is searching for a leaf with a height to width ratio of 3.4 and the samples that have been collected on average have a ratio of 3 then the search request has a deviation of 0.4. If the samples are aggregated into an average, max, min and deviation and the species are averaged into a universal max, min and deviation then fuzz levels can be realistically set without the need to guess.

Chapter 4

Higher Level Design

After deciding the details of the systems implementation, a layer of abstraction can now be added to the overall architecture and the structure can be decided.

4.1 Software Stack

The core utilities in this system will consist of a backend database, a webserver to act as a middleman between the clients and the database, a web interface for performing search requests, a web interface for performing relevant administrative database tasks and finally computer vision scripts for creating the inverted file data.

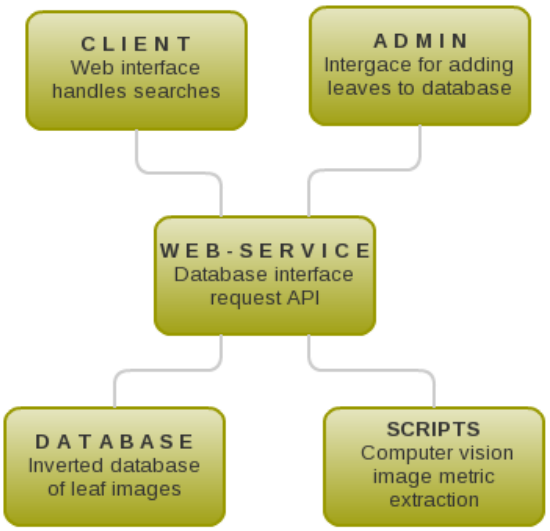

Figure 4.1: Software stack

4.2 Database Design

The database must be able to hold information about plant species and plant specimens within a species. The species table must also maintain averaged data about each of the specimens within the species so that when it is scaled up with thousands of specimens it will not need to retrieve the entire dataset.

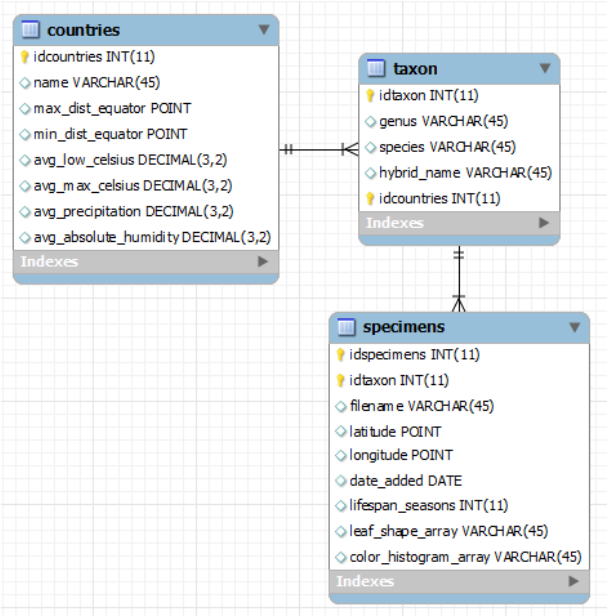

Figure 4.2: First attempt at an ERD

Using identifying foreign key relationships for the tables means that in order to add a leaf to the database the parent tables must be checked for consistency before an insert can be made. This will reduce redundant information but may imply that changes to the data in the leaf table creates changes in the parent tables. This is not the case because the parent tables contain temporal data which is generated on a weekly basis.

Figure 4.3: Second attempt at an ERD

A method of normalizing the tables and removing the possibility of people getting confused is to use a foreign key relationship and use a referencing table between the real data and the temporal data. This however does not remove the obligation to check for consistency before performing an insert onto a child table.

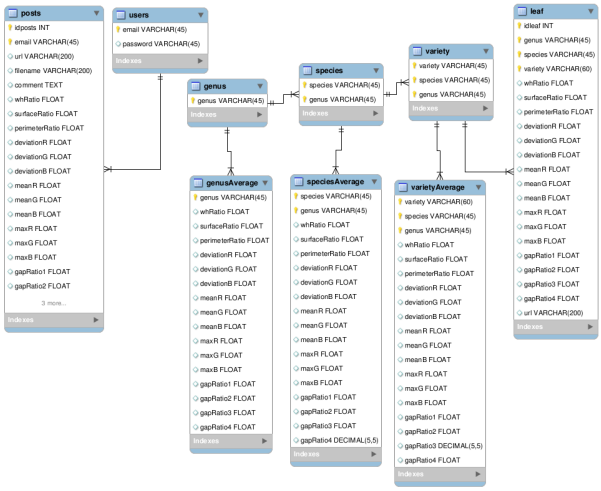

Figure 4.4: Third attempt at an ERD

Using non-identifying relationships and allowing null values for keys in the parent tables will not remove the need to check for consistency.

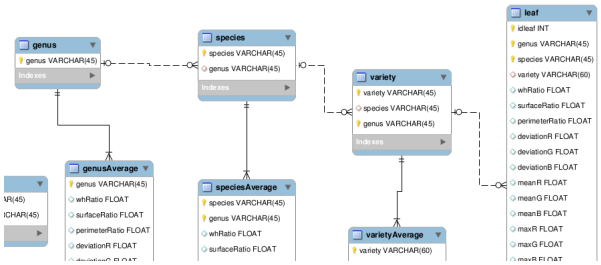

Figure 4.5: Forth attempt at an ERD

Chapter 5

Implementation

5.1 Setting up the Environment

If this software is providing a service to customers the operating system does not need to have bleeding edge packages but needs to have been heavily tested for reliability. A red hat family distribution such as Red Hat or CentOS is the equivalent of Windows Server edition in the Linux world.

This application uses a http server with a PHP module to provide the user with a web interface. The addition of PHP modules allows us to produce dynamic web pages using a model view controller architecture.

The interface allows users to submit images to be processed through an image recognition module. The image recognition module is a bash script to perform the biometric extraction using the ImageMagick command line tools. The results from the extraction script are taken and inserted into the database.

MySQL is a suitable database to use as a backend because it supports multiple users simultaneously. The search engine requires a database backend in order to provide locking for a multiuser system and to keep the data in memory for high speed access.

5.2 SQL Search Algorithm 1

The first search algorithm titled "Skim" accepts sixteen metrics as parameters to the query. In addition to the sixteen metrics it accepts two tolerance variables for setting the fuzz level of the search.

Take two arbitrary metrics "5" and "25". If a single search is performed using separate statements with a lower threshold of the metric value minus one, and with an upper threshold of the metric value of plus one the output would contain duplicates from merging the separate search results. The number of occurrences that a search request triggers should have a direct correlation with the similarity between the search request and search result.

Lower tolerance = -1		Upper tolerance = +1
A4	E4	Z4

Figure 5.1: Result for first metric using Skim

From these results A, Z, B and J occur only one time each but E occurs twice. This means that E is more likely to be the result that we are looking for.

Lower tolerance = -1		Upper tolerance = +1
B24	E25	J26

Figure 5.2: Result for second metric using Skim

The tolerance levels should be relative to the column they are restricting. If a column has a value which is set at 100 and you want to search for this value within a 10% range then you will need to set the fuzz level for this column to 90 and 110. However if a column has a value of 50 then you will want to set the tolerance to 45 and 55. This is tedious if it needs to be calculated for every value. It is easier to set the tolerance to .9 and 1.1 and multiply it by the value so that 32 additional variables are not needed for 16 metrics.

The following search query performs a variable amount of read operations depending on the fuzz levels that have been set. If the tolerance is low then less data is processed. If the tolerance is high then more data is processed. This provides flexibility to the system.

```
SELECT results.genus,results.species,COUNT(*) as occurrences
from(
  select genus,species from speciesAverage
  where whRatio BETWEEN ($1*$lfuzz) and ($1*$ufuzz)

  union all select genus,species from speciesAverage
  where surfaceRatio BETWEEN ($2*$lfuzz) and ($2*$ufuzz)

  union all select genus,species from speciesAverage
  where perimeterRatio BETWEEN ($3*$lfuzz) and ($3*$ufuzz)

) results
group by genus,species ORDER BY occurrences DESC;
```

Figure 5.3: SQL for Crunch Algorithm

This algorithm returns a single set of results for every metric and combines them. These results are then summarized by their primary key and the number of occurrences of each unique key are outputted. The select statement then returns derived tabular data which consists of the combined primary key (genus and species) and the number of occurrences that have been counted.

Genus	Species	occurrences
Amelanchier	canadensis	9
Prunus	sargentii	7
Quercus	shumardii	7
Crataegus	viridis	7

Figure 5.4: Example of tabular output from Skim

5.3 SQL Search Algorithm 2

The second search algorithm titled "Crunch" accepts sixteen metrics as parameters to the query. This query returns the entire database in a single set of tabular data. This algorithm performs a single read operation on the entire dataset. This means that the time taken to perform a search request can be predicted before the request is executed. The results are also ranked differently. The ratio of difference between a search request and a row in the database is summed up across the sixteen metrics. This means that if a request has fifteen exact matches and a difference of zero but the sixteenth metric differs by a factor of fifty, then the result will be ranked at fifty. The lower the rank the more accurate the result.

```
#Crunch Algorithm
select genus,species,
(
  ABS(greatest ($1, whRatio)/least ($1, whRatio)) +
  ABS(greatest ($2, surfaceRatio)/least ($2, surfaceRatio) ) +
  ABS(greatest ($3, perimeterRatio)/least ($3, perimeterRatio) )
)
as difference
from speciesAverage
ORDER BY difference ASC;
```

Figure 5.5: SQL for Skim Algorithm

This algorithm takes an input value and compares it to its matching column in every row in the database. A temporary column is outputted which is the number of times that the larger value can be divided by the smaller value. Whether the search input is a larger value is ignored. The number of times a division can be made is converted into an absolute value using the ABS function in SQL.

Genus	Species	Difference
Prunus	serrulata	52860.903178236
Pseudolarix	amabilis	57122.8957737989
Amelanchier	canadensis	62410.1542796309
Quercus	bicolor	63137.3635256156

Figure 5.6: Example of tabular output from Crunch

5.4 SQL Search Algorithm 3

Combining the number of occurrences with the total difference will give an indication of the reliability of a search result. For example if a search is performed using the Skim algorithm and multiple results triggered the same amount of metrics then the user would be unable to distinguish which result is more accurate. Similarly if a search result is performed using the Crunch algorithm and two results had a similar difference but the user would be unable to tell the number of metrics that the difference is spread across. Combining the two algorithms tells the user the reliability of the information and the spread of the total difference across the metrics.

```
SELECT genus,species,occurrences,difference
from (
  #Skim Algorithm
  SELECT results.genus as genus,results.species as species,
  COUNT(*) as occurrences
  from(
    select genus,species from leaf
    where whRatio BETWEEN ($1*$lfuzz) and ($1*$ufuzz)
    union all select genus,species from leaf
    where surfaceRatio BETWEEN ($2*$lfuzz) and ($2*$ufuzz)
    union all select genus,species from leaf
    where perimeterRatio BETWEEN ($3*$lfuzz) and ($3*$ufuzz)
  ) results
  group by genus,species
) tbl1
left join
(
  #Crunch Algorithm
  select genus,species,
  (
    ABS(greatest($1, whRatio)/least($1, whRatio)) +
    ABS(greatest($2, surfaceRatio)/least($2, surfaceRatio) ) +
    ABS(greatest ($3, perimeterRatio)/least($3, perimeterRatio) )
  )
  as difference
  from leaf
  GROUP BY genus,species
) tbl2
USING (genus,species)
ORDER BY occurrences DESC, difference ASC;
```

Figure 5.7: SQL for combined algorithms

As seen in the following example there are several results with equal numbers of occurrences but different differences. This these cases the results are ordered in ascending order secondary to the descending order of the ocurrences.

Genus	Species	Occurrences	Difference
Abies	concolor	549	74711.3280640727
Quercus	shumardii	122	70026.6832482166
Quercus	cerris	122	77677.2512565685
Quercus	rubra	122	98165.6190726692

Figure 5.8: Example of tabular output from Combined

5.5 Computer Vision Script

The following steps are taken to take an input of a file that contains an image of a leaf and output image artefacts. These artefacts are temporary for the duration of the script and should be tidied up after the script has completed. For optimal performance these operations should be performed in RAM and not on disk.

1. Convert File to PNG
2. Segment leaf from uniform background colour
3. Crop image border to leaf edges
4. Create segmented silhouette of leaf
5. Create image outlining perimeter of leaf by marking with 1 pixel at edge of leaf
6. Cut the image up into 2 pixel slices from 4 different sides

After creating the image artefacts the biometrics are extracted and returned to the standard output. The following steps are taken to output the metrics. These steps are performed on the various artefacts that were created in the first phase of the extraction script.

1. Calculate Width to Height ratio
2. Calculate information for Red, Green and blue colour channels (mean, deviation, average, maximum)
3. Calculate relative surface area of the leaf by counting pixels in segmented silhouette artefacts. This is a ratio to the total number of pixels in the image
4. Calculate the relative perimeter by counting the number of marked pixels at the edge of the leaf. This is calculated as a ratio to the surface area
5. Calculate gaps between leaf edge and image border

```
#!/bin/bash
#Leaf Biometrics Script
#Author: Hugh Pearse
#----------File Conversion----------

#Take a filename from the standard input
#Shell expansion to replace the file extension with ".png"
pic=${1/\.*/.png}

#Convert file format to PNG
convert $1 png:$pic

#Segment leaf from uniform background colour
convert -transparent '#000000' -fuzz 10% $pic seg_$pic

#Crop image border to leaf edges
./autotrim.sh -c "0,0" seg_$pic seg_cr_$pic
```

Script Extract 1: Creating base image for processing

```
#Create segmented silhouette of leaf for
#measuring shape information
convert seg_cr_$pic -channel 'RGB' -black-threshold 96% \
-colorspace "Gray" seg_cr_sil_$pic

#Create black on white silhouette for
#measuring perimeter information
convert seg_cr_sil_$pic -background "#FFFFFF" -alpha Background \
-alpha Off seg_cr_bwsil_$pic

#Create image outline of leaf perimeter
convert seg_cr_bwsil_$pic -edge 1 seg_cr_bwsil_edge_$pic

#Cut the image up into 2 pixel slices
width=`identify -format "%[width]" seg_cr_$pic`
height=`identify -format "%[height]" seg_cr_$pic`
oneThirdH=`echo "$height/3" | bc`
oneThirdW=`echo "$height/3" | bc`
```

Script Extract 2: Creating artefacts optimised for different biometrics

```
#rotate image
convert seg_cr_bwsil_$pic -rotate 180 \
seg_cr_bwsil_180_$pic
convert seg_cr_bwsil_$pic -rotate 90 \
seg_cr_bwsil_90_$pic
convert seg_cr_bwsil_$pic -rotate 270 \
seg_cr_bwsil_270_$pic

#extract slices 1/3 from top
convert -extract $widthx2+0+$oneThirdH \
seg_cr_bwsil_$pic slice1_$pic
convert -extract $widthx2+0+$oneThirdH \
seg_cr_bwsil_180_$pic slice3_$pic
convert -extract $heightx2+0+$oneThirdW \
seg_cr_bwsil_90_$pic slice2_$pic
convert -extract $heightx2+0+$oneThirdW \
seg_cr_bwsil_270_$pic slice4_$pic

convert -fill red slice1_$pic -floodfill \
+0+0 "#FFFFFF" -fuzz "10%" slice1.0_$pic
convert -fill red slice3_$pic -floodfill \
+0+0 "#FFFFFF" -fuzz "10%" slice3.0_$pic
convert -fill red slice2_$pic -floodfill \
+0+0 "#FFFFFF" -fuzz "10%" slice2.0_$pic
convert -fill red slice4_$pic -floodfill \
+0+0 "#FFFFFF" -fuzz "10%" slice4.0_$pic
```

Script Extract 3: Creating artefacts for shape analysis - part 1

```
convert +transparent red -fuzz 10% \
slice1.0_$pic slice1.1_$pic
convert +transparent red -fuzz 10% \
slice3.0_$pic slice3.1_$pic
convert +transparent red -fuzz 10% \
slice2.0_$pic slice2.1_$pic
convert +transparent red -fuzz 10% \
slice4.0_$pic slice4.1_$pic

convert slice1.1_$pic -bordercolor none \
-border 1x1 -trim +repage slice1.2_$pic
convert slice3.1_$pic -bordercolor none \
-border 1x1 -trim +repage slice3.2_$pic
convert slice2.1_$pic -bordercolor none \
-border 1x1 -trim +repage slice2.2_$pic
convert slice4.1_$pic -bordercolor none \
-border 1x1 -trim +repage slice4.2_$pic
```

Script Extract 4: Creating artefacts for shape analysis - part 2

```
#----------Biometrics----------

#Create Width to Height ratio

#Extract information for Red, Green and blue colour channel
deviationR=‘identify -channel R -format \
"%[standard-deviation]" seg_cr_$pic‘
deviationG=‘identify -channel G -format \
"%[standard-deviation]" seg_cr_$pic‘
deviationB=‘identify -channel B -format \
"%[standard-deviation]" seg_cr_$pic‘

meanR=‘identify -channel R -format "%[mean]" seg_cr_$pic‘
meanG=‘identify -channel G -format "%[mean]" seg_cr_$pic‘
meanB=‘identify -channel B -format "%[mean]" seg_cr_$pic‘

maxR=‘identify -channel R -format "%[max]" seg_cr_$pic‘
maxG=‘identify -channel G -format "%[max]" seg_cr_$pic‘
maxB=‘identify -channel B -format "%[max]" seg_cr_$pic‘
```

Script Extract 5: Extracting simple colour metrics

```
#Calculate relative surface area of the leaf
#Take the total number of black pixels and empty
#pixels to get the total surface area of the image.
#Divide the black pixels by the total to get
#the ratio of leaf to image.
width=‘identify -format "%[width]" seg_cr_$pic‘
height=‘identify -format "%[height]" seg_cr_$pic‘
totalPix=‘echo "$width*$height" | bc‘
blackCount=‘convert seg_cr_sil_$pic -format %c \
histogram:info:- 2>&1 | grep black | awk \
’{printf $1}’ | tr -d ’:’‘
noneCount=‘convert seg_cr_sil_$pic -format %c \
histogram:info:- 2>&1 | grep none | awk \
’{printf $1}’ | tr -d ’:’‘
surfaceRatio=‘echo "scale=5; $blackCount/$totalPix" | bc‘

#Calculate width to height ratio of the leaf
whRatio=‘echo "scale=2; $height/$width" | bc‘

#Calculate the leaf preimeter to surface area ratio
#The edge should be 1px wide
#we can count all of the white pixels to
#sum up the total perimeter
#then divide the perimeter by the surface area
whiteEdge=‘convert seg_cr_bwsil_edge_$pic -format %c \
histogram:info:- 2>&1 | grep white | awk \
’{printf $1}’ | tr -d ’:’‘
perimeterRatio=‘echo "scale=0; $blackCount/$whiteEdge" | bc‘
```

Script Extract 6: Create secondary biometric data based on primary data

```
#Calculate gaps around the leaf as a ratio to the full slice
sliceWidth1=‘identify -format "%[width]" slice1.1_$pic‘
sliceWidth3=‘identify -format "%[width]" slice3.1_$pic‘
sliceWidth2=‘identify -format "%[width]" slice2.1_$pic‘
sliceWidth4=‘identify -format "%[width]" slice4.1_$pic‘

gapWidth1=‘identify -format "%[width]" slice1.2_$pic‘
gapWidth3=‘identify -format "%[width]" slice3.2_$pic‘
gapWidth2=‘identify -format "%[width]" slice2.2_$pic‘
gapWidth4=‘identify -format "%[width]" slice4.2_$pic‘

gapRatio1=‘echo "$sliceWidth1/$gapWidth1" | bc‘
gapRatio3=‘echo "$sliceWidth3/$gapWidth3" | bc‘
gapRatio2=‘echo "$sliceWidth2/$gapWidth2" | bc‘
gapRatio4=‘echo "$sliceWidth4/$gapWidth4" | bc‘

#----------Output----------
echo "$whRatio $surfaceRatio $perimeterRatio $deviationR \
$deviationG $deviationB $meanR $meanG $meanB $maxR $maxG \
$maxB $gapRatio1 $gapRatio3 $gapRatio2 $gapRatio4";
```

Script Extract 7: Calculate leaf shape data and output all metrics

5.6 Web Interface

The web interface uses a model view controller architecture (also known as MVC). The model is a file that contains information about the tables, keys and columns in the database. This allows the web application to use object relational mapping. Object relational mapping means programs can be written in a way that cells in a table can be referenced directly from the application without needing to manually connect to the database, the database connection is automated. The controller file is the business logic of the web application. The controller is where tasks are written into functions and are triggered by actions such as a page request. The views are frontend files that are delivered to the user after the application receives a page request. The views can be either static html pages or dynamically generated.

```
bash-4.1# php ./framework/yiic webapp myapplication
```

Figure 5.9: Creating a MVC skeleton

Before any database related tasks can be performed, the model files need to be modified to act as a representation of the tables, columns and keys in the database. This can either be done manually or alternatively it can be done automatically if the tables have already been created in the database.

```
bash-4.1# php ./framework/yiic shell
model Users users
model Posts posts
model Leaf leaf
model SpeciesAverage speciesAverage
```

Figure 5.10: Creating the database model automatically

Certain MVC frameworks support automatic code generation to perform common menial database tasks such as create, read, update and delete (also known as CRUD). This CRUD generation accelerates the implementation process to a point that the website will be created to provide bare bones functionality for each table. The remaining functionality of the website needs to be written manually.

```
bash-4.1# php ./framework/yiic shell
model Users users
model Posts posts
model Leaf leaf
model SpeciesAverage speciesAverage
```

Figure 5.11: Setting up Object Relational Mapping

The rich graphical environment is implemented by using the jquery library. The results page is displayed with a hidden element for each row returned. A mouseover action is applied to each result that triggers the unhide action to display the rows corresponding hidden element.

Figure 5.12: Screenshot of front page before login

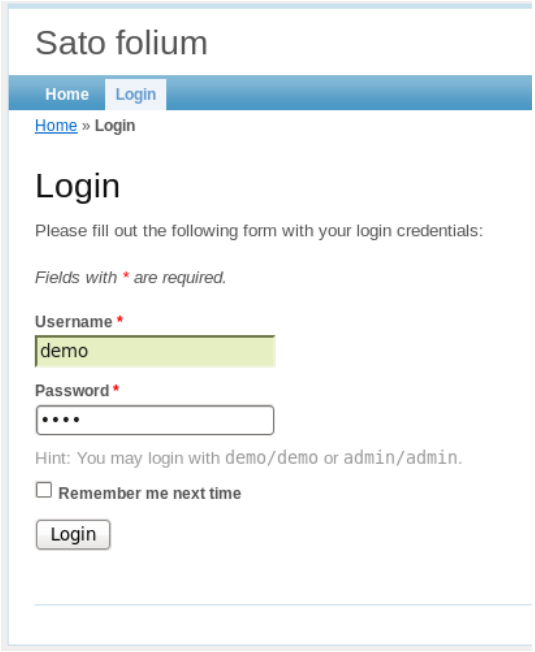

Figure 5.13: Screenshot of login page

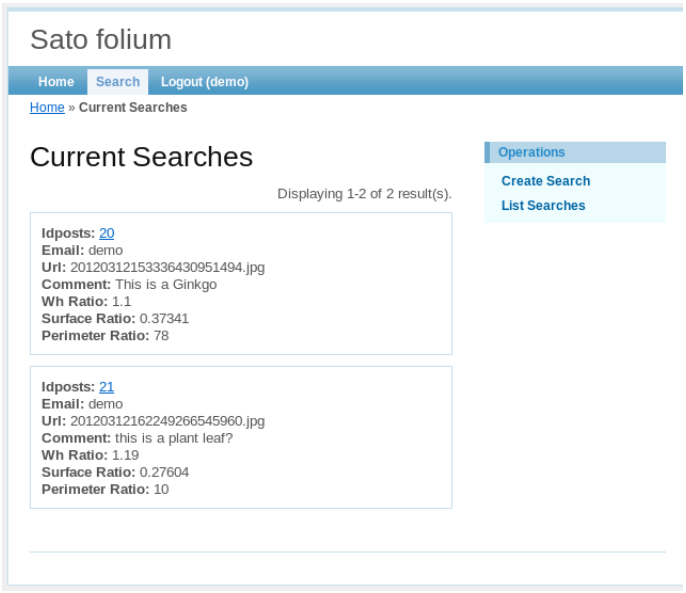

Figure 5.14: Screenshot of searches page

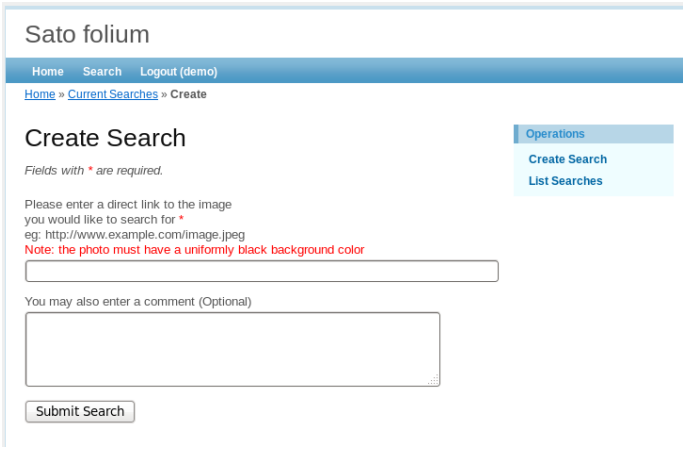

Figure 5.15: Screenshot of create search page

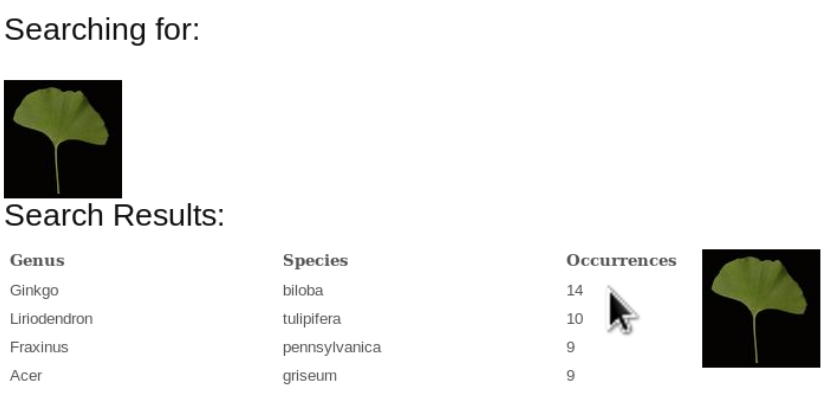

Figure 5.16: Screenshot of jQuery performing mouseover

Chapter 6

Testing

6.1 Performance

The first algorithm was tested by inserting 10,000 records into the leaf table and by running 164 unique requests directly against the leaf table without averaging the leaf data into the variety table or species table or genus table. This ensures that there is a minimum amount of time spent sending socket requests to the kernel and a maximum amount of time spent measuring the performance of the query structure.

Entity	Time
real	0m27.917s
user	0m0.853s
sys	0m1.005s

Figure 6.1: Benchmarking Skim Algorithm

The second algorithm demonstrated a significant increase in performance using the same dataset. The time spent processing application code and libraries is relatively similar to the first algorithm because most of the time is wasted in the first algorithm performing read operations which do not require the cpu to interact.

Entity	Time
real	0m11.023s
user	0m0.867s
sys	0m1.078s

Figure 6.2: Benchmarking Crunch Algorithm

The hybrid algorithm demonstrated a significant decrease in performance using the same dataset as the previous two algorithms. The additional functionality of showing both the number of occurrences and difference comes at a cost.

Entity	Time
real	0m37.725s
user	0m1.071s
sys	0m1.070s

Figure 6.3: Benchmarking Hybrid Algorithm

6.2 Accuracy

Accuracy is tested by performing a search for every element in the database with a predetermined matching request. If the results that are returned contain the data that was searched for every single element then the search query will be reasonably steady. This can easily be automated at the command line level using a combination of bash, grep and the MySQL command line utility.

```
#!/bin/bash
rm results.txt
for FILENAME in `ls -ISEARCH.sh -Iscript.sh -1`
do
        specvar=`echo $FILENAME | tr '-' '.' | sed 's/\.jpg//g' `;
        result=`./SEARCH.sh $FILENAME`;
        exists=`echo $result | egrep -c "$specvar"`;

        #Write to file if result was returned
        echo -n $exists >> results.txt
        #Write to file image name
        echo $specvar >> results.txt
done;
```

Figure 6.4: Accuracy script

The head command will filter the top 10 results in every request. If the result was not found in the top 10 returned results then the request is output as failed. For all three algorithms the results returned a correct match for 99% of the requests. The only two requests that failed were "Crataegus crus galli" and "Robinia pseudo acacia". The reason these failed to be found was that the database design only works for the binomial naming convention which consists of genus and species. The two results that failed use the trinomial naming convention because they have been categorized into sections/subsections/divisions.

6.3 Reliability

Reliability is measured similarly to accuracy; except instead of testing every element in the database once, every element is tested multiple times. This differs from performance in the way that the transaction is executed. In performance testing a single query is executed for thousands of results. In reliability testing a new query is executed for every single result.

```
#!/bin/bash

function query(){
        for LINE in ‘less database.txt‘
        do
                LINE=‘echo $LINE | tr ’,’ ’ ’‘;
                ./query.sh $LINE &> /dev/null
        done;
}

for i in {1..1000}
do
        query;
done
```

Figure 6.5: Reliability script

6.4 Survey

A survey was chosen as a suitable method of data collection. The advantages of a survey over focus groups and interviews is that it does not require meeting the users and multiple surveys can be done simultaneously. This means that time can be used much more effeciently. Choosing which questions to put on the survey is a difficult task. The questions can be contextual requiring the users to submit constructive feedback or the questions can be numerical so that the answers can be averaged and additional data can be derived from the results. For these reasons the following questions were chosen for five different web pages labelled A-E.

1. A1-Clarity of the web pages purpose
2. A2-Clarity of the next step to take
3. A3-Your satisfaction with the page
4. A4-Your frustration with the page

The questions should be labelled with a section number and question number. This means that the answers can be easily filtered using a regular expression and averaged.

```
less SURVEY.txt | grep "A1" | sed -e ’s/.*(//; s/).*//;’ | \
tr -d ’ ’ | stats | egrep ‘‘mean|std dev’’
```

Figure 6.6: Survey script

Out of all the people who were contacted to try the demo kit and fill out the survey seven people responded. Because the questions were numerical their answers were averaged into a single survey result. The answer to each question is a number between 1 and 10 inclusive where 1 is the lowest answer and 10 is the highest.

```
A1 - Welcome Page Clarity of page purpose
mean:  8.142857142857
std dev:  1.345185418269

A2 - Welcome Page Clarity of next step
mean:  6.857142857143
std dev:  3.023715784074

A3 - Welcome Page satisfaction
mean:  7.142857142857
std dev:  1.951800145897

A4 - Welcome Page frustration
mean:  2.571428571429
std dev:  2.070196678027
```

Figure 6.7: Survey Results for Welcome page

```
B1 - Login Page Clarity of page purpose
mean:  9.428571428571
std dev:  0.975900072949

B2 - Login Page Clarity of next step
mean:  9.428571428571
std dev:  1.133893419028

B3 - Login Page satisfaction
mean:  8.571428571429
std dev:  1.272418020561

B4 - Login Page frustration
mean:  1.571428571429
std dev:  0.975900072949
```

Figure 6.8: Survey Results for Login page

```
C1 - Curent Searches Page Clarity of page purpose
mean:  6.714285714286
std dev:  2.563479777847

C2 - Curent Searches Page Clarity of next step
mean:  5.714285714286
std dev:  2.138089935299

C3 - Curent Searches Page satisfaction
mean:  6.400000000000
std dev:  2.509980079602

C4 - Curent Searches Page frustration
mean:  2.857142857143
std dev:  1.864454471472
```

Figure 6.9: Survey Results for Current Searches page

```
D1 - Create Search Page Clarity of page purpose
mean:  9.142857142857
std dev:  1.214985792588

D2 - Create Search Page Clarity of next step
mean:  8.857142857143
std dev:  1.676163419695

D3 - Create Search Page satisfaction
mean:  8.500000000000
std dev:  1.378404875209

D4 - Create Search Page frustration
mean:  1.428571428571
std dev:  0.786795792469
```

Figure 6.10: Survey Results for Create Search page

```
E1 - View Result Page Clarity of page purpose
mean:  8.000000000000
std dev:  1.154700538379

E2 - View Result Page Clarity of next step
mean:  5.666666666667
std dev:  2.943920288776

E3 - View Result Page satisfaction
mean:  7.000000000000
std dev:  1.414213562373

E4 - View Result Page frustration
mean:  2.285714285714
std dev:  1.603567451475
```

Figure 6.11: Survey Results for View Result page

Chapter 7

Conclusion

7.1 Changes

There are countless improvements that could be made to this project to increase accuracy and improve performance. The addition of more metrics would reduce the number of false positives.

In terms of accuracy, flower colour information and seed colour information to the database would be more true to real life as living organisms are classified by their reproductive systems. In terms of reliability, derived deviation information could be added to the parent tables to show the reliability of the averaged information in the temporal tables, this would be a deviation of the deviation data. In terms of performance, the weighting algorithm for calculating the total difference between a search request and a result could be improved by adding a pre-calculated hash table so that search requests do not need to compute the difference at the time of retrieval. This would mean that the database software is performing read operations instead of mathematical operations which means it can be done without using the CPU. Finally the reliability of the system is improved by the quantity of data in it. If the database contains thousands of leaves per variety then this will be averaged up into the parent tables.

The system could also be modified to accommodate different priorities of requests. A request with an accuracy priority could perform computational calculations on the leaf table using the Crunch algorithm. This could take several hours depending on the size of the database. Similarly a request with a performance priority could perform the Skim algorithm on the averaged temporal tables which would minimize the amount of reads that need to be performed and minimizes the amount of computation that needs to be performed.

Bibliography

[1] United States Botanic Garden. Plant family tree - 2011. http://www.usbg.gov/education/schools.cfm.

[2] Shiro Tsuyuzaki. Characteristics of "number of veins" to estimate leaf maturity in pteris mutilata. *Journal of Plant Research*, pages 415–418, 2000.

[3] Smithsonian Institution. Botany department, smithsonian institution. http://botany.si.edu/, 2011.

[4] Kew Gardens London. Herbarium catalogue - kew gardens london. http://apps.kew.org/herbcat/gotoHerbariumGrowthPage.do, 2011.

[5] Glen Stansberry. 7 version control systems reviewed. http://www.smashingmagazine.com/2008/09/18/the-top-7-open-source-version-control-systems/, September 2008.

[6] Maria-Elena Nilsback and Andrew Zisserman. A visual vocabulary for flower classification. In *CVPR (2)'06*, pages 1447–1454, 2006.

[7] Shiro Tsuyuzaki. Characteristics of number of veins to estimate leaf maturity in pteris mutilata (pteridaceae). *Journal of Plant Research*, 113:415–418, 2000. 10.1007/PL00013949.

[8] George F. and Sprague Jr. Genetic exchange between kingdoms. *Current Opinion in Genetics and Development*, 1(4):530 – 533, 1991.

[9] E. Gamma. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley professional computing series. Addison-Wesley, 1995.

[10] D.E. Knuth. *The art of computer programming*. Number v. 3 in The Art of Computer Programming. Addison-Wesley Pub. Co., 1998.

[11] R.J. Pankhurst. *Practical taxonomic computing*. Cambridge University Press, 1991.