CIBIV MFPL
Center for Integrative
Bioinformatics Vienna

University of Vienna
Faculty of Computer Science
Univ.-Prof. Arndt von Haeseler

A Substitution Model for the English Language

Bachelor thesis
by

Wolfgang Fischl
0602106

Advisor: Univ.-Prof. Arndt von Haeseler

Wolfgang Fischl
Setzgasse 32
7072 Mörbisch am See

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Vienna, August, 10th 2010

Wolfgang Fischl

Contents

1 Introduction

The methods and principles of evolution are studied since Charles Darwin published his book "On the Origin of Species" in 1859. From there on evolutionary trees were constructed, which help us to understand the relationship of species. Such trees can be estimated from either morphological data or sequence data. If we use sequence data, we still have the choice to either use DNA, RNA or protein sequences. No matter which type of sequence we use, if we want to analyze such sequences or if we want to infer evolutionary trees by some probabilistic approach, the model by which the sequences evolve plays an important role.

At a molecular level evolution is nothing more than the change of sequences. This means that during the course of time symbols of a sequence are replaced by others or symbols are deleted, which is called deletion or other symbols are added, which is called insertion. In biology those symbols are often abbreviated by a one letter-code, where one letter represents a specific molecule. Models describing such substitutions are dependent on the type of molecule (DNA, proteins, ...), but general models do exist for all kinds of biological sequences. An overview of substitution models will be given in section 2.

Substitution models can also be transferred to other domains, e.g. to linguistics. The change of languages during the course of time can also be seen as evolution. Nowadays, like in biology, we only see the results of this continuous process. All of the current derivations of English evolved from Middle English, which itself has Old English as ancestor. In order to compare different languages, we need to know which substitutions occur more likely than others. The world wide web is a large collection of written text. Its content is indexed by many search engines. Those search engines do not only index correctly spelled words, but also misspelled ones. When we compare the occurrences of these, we might be able to estimate a substitution model for a language. Here we try to do such an estimation for the English language. The collection of the data and the estimation of the substitution model will be explained in the methods section (sec. 3).

Our results and a short analysis will be given in section 4 and at the end of this thesis we will give a short outlook and conclusion in section 5.

2 Substitution Models

In the following we will describe methods and notations used in Tavaré (1986). Let us consider two gene sequences taken one from each of two species. We do know, that these two sequences are homologous, that means they are derived from a common ancestor, but we do not know the sequence of the common ancestor. In order to estimate the sequence of the common ancestor and/or the time of divergence of the two species, we need to know the parameters at which those genes evolved. If we want to model such a process, we need to take into account many forces which change the genes. Such forces are substitution, insertion, deletion, duplication and transition (Tavaré, 1986). For simplicity and the introduction we will only consider DNA sequences, which consist of four bases **A**, **C**, **G**, **T** and we will only consider substitution, which is the replacement of one base by another. Another assumption is that both sequences have the same length $\boldsymbol{n}$. We will write $\boldsymbol{X_i\left(t\right)}$ for the base that occurs in species $\boldsymbol{X}$ at position $\boldsymbol{i}$ after $\boldsymbol{t}$ time units of divergence from the common ancestor. If we now have two species X and Y, we know that $X_i\left(0\right) = Y_i\left(0\right), i = 1, 2, \ldots, n$, therefore both species have the same sequence at the start of the process that we will try to model ($t = 0$), but then evolve independently. During the course of this section we will denote $\boldsymbol{X\left(\cdot\right)}$ and $\boldsymbol{Y\left(\cdot\right)}$ for the nucleotide in homologous positions in sequence $\boldsymbol{X}$ and $\boldsymbol{Y}$.

For the substitution process $\left\{X\left(t\right), t \geq 0\right\}$ we can define a transition function

$$p_{ij}^{X} = P\left[X\left(t\right) = j | X\left(0\right) = i\right] \tag{1}$$

which is the probability of having nucleotide $\boldsymbol{j}$ after time $\boldsymbol{t}$, when there was nucleotide $\boldsymbol{i}$ at time $\boldsymbol{0}$. For the Y species we can describe a corresponding function and define

$$h_{ij}\left(t\right) = P\left[X\left(t\right) = i, Y\left(t\right) = j | X\left(0\right) = Y\left(0\right)\right] \tag{2}$$

which is the probability of having nucleotide $\boldsymbol{i}$ in species X and nucleotide $\boldsymbol{j}$ in species Y, given that both nucleotides have the same common ancestor and under the assumption of a molecular clock, which is that substitutions occur in both species with the same rate. h_{ij} can be calculated with

$$h_{ij}\left(t\right) = \sum_{l \in \{A,C,G,T\}} \pi_l p_{li}^{X}\left(t\right) p_{lj}^{Y}\left(t\right) \tag{3}$$

where π_l is the initial distribution of nucleotides ($\pi_l = P[X(0) = l]$). Under the assumption that both sequences evolve with the same probabilities ($p_{ij}^X(t) = p_{ij}^Y(t) \equiv p_{ij}(t)$), h_{ij} can be written in matrix notation with $H(t) = (h_{ij}(t))$ (Tavaré, 1986):

$$H(t) = P(t)^T H(0) P(t), t \geq 0 \tag{4}$$

where $P(t) = (p_{ij}(t))$ and $H(0) = diag\{\pi_A, \pi_C, \pi_G, \pi_T\}$. $\boldsymbol{H}$ is called divergence matrix and has the frequencies of letter-pairs $(X_i(t), Y_i(t)) \forall i$ as entries. There is still need to specify the transition probabilities $P(t)$. Most frequently $P(t)$ is specified with a **Markov model**, where

$$\boldsymbol{P}(t) = e^{\boldsymbol{Q}t} \tag{5}$$

Markov models use a **matrix** $\boldsymbol{Q} = (q_{ij})$ to model the relative change of each nucleotide. They share following set of assumptions (for nucleotide data) (from Strimmer and von Haeseler, 2009):

1. At any given site in a sequence, the rate of change from base i to base j is independent from the base that occupied that site prior i (*Markov property*).
2. Substitution rates do not change over time (*homogeneity*).
3. The relative frequencies of A, C, G, and T ($\pi_A, \pi_C, \pi_G, \pi_T$) are at equilibrium. (*stationarity*).

Markov processes are discrete stochastic processes that model a sequence of finite states, where future states are independent of the past states given the present state (Wilkinson, 2006). For DNA sequences each site is treated as a random variable with $m = 4$ possible states. The $\boldsymbol{Q}$ matrix for nucleotide sequences has 4 rows and 4 columns. An example of such a matrix is shown in Equation 6, which is known as rate matrix $\boldsymbol{Q}$ for the Jukes and Cantor model (JC69) (Jukes and Cantor, 1969).

$$\begin{array}{c} \\ A \\ C \\ G \\ T \end{array} \begin{array}{c} \begin{array}{cccc} A & C & G & T \end{array} \\ \begin{pmatrix} -\frac{3}{4}\mu & \frac{1}{4}\mu & \frac{1}{4}\mu & \frac{1}{4}\mu \\ \frac{1}{4}\mu & -\frac{3}{4}\mu & \frac{1}{4}\mu & \frac{1}{4}\mu \\ \frac{1}{4}\mu & \frac{1}{4}\mu & -\frac{3}{4}\mu & \frac{1}{4}\mu \\ \frac{1}{4}\mu & \frac{1}{4}\mu & \frac{1}{4}\mu & -\frac{3}{4}\mu \end{pmatrix} \end{array} \tag{6}$$

The JC69 is the simplest model for nucleotide substitution and has only one parameter, the substitution rate μ. The rows specify the relative change from one nucleotide to another, therefore the element q_{13} specifies the relative change from A $\rightarrow$ G. In more general terms, every non-diagonal entry specifies the change from nucleotide i to j. The diagonal entries are chosen, such that the sum of each row equals zero. They represent the total flow that leaves nucleotide i. Note that in the Jukes and Cantor model, all substitutions are equally likely, with a substitution rate of μ. Additionally, every of the four nucleotides occurs with equal frequency ($\pi_A = \pi_C = \pi_G = \pi_T = \frac{1}{4}$). (Strimmer and von Haeseler, 2009)

We also assume a *time-reversible* model, that is that the rate of change from i to j is the same as from j to i. This induces Eqn. 7.

$$\pi_i q_{ij} = \pi_j q_{ji} \quad \forall i, j \tag{7}$$

Once we have such a matrix $\boldsymbol{Q}$ it is possible to calculate the probability of change from any base to any other during evolutionary time t using Equation 5. This exponential is often calculated by decomposing the rate matrix $\boldsymbol{Q}$ into its (right) eigenvalues and eigenvectors,

$$\boldsymbol{P}(t) = \boldsymbol{U} \cdot diag\{e^{\lambda_1 t}, \ldots, e^{\lambda_4 t}\} \cdot \boldsymbol{U}^{-1} \tag{8}$$

where λ_i are the eigenvalues of $\boldsymbol{Q}$ and $\boldsymbol{U}$ is the matrix with the corresponding eigenvectors. We can also reverse this decomposition and calculate the rate matrix $\boldsymbol{Q}$ from the transition probability matrix $\boldsymbol{P}(t)$. Weiss and von Haeseler (2003) shows that a transition probability matrix $\boldsymbol{P}(t)$ can be estimated from relative frequencies of letter pairs $(X_i(t), Y_i(t)), i = 1, \ldots, n$.

Our goal is to find such letter pairs frequencies for the English language and to estimate a transition probability matrix from which we can infer a 26×26 rate matrix.

3 Methods

In this section we will explain the collection of data from the Google search engine (3.1) by an automated polling process (section 3.3). This data is stored in a small database (section 3.2). And from the stored data we estimate a rate matrix $\boldsymbol{Q}$ with the method described in section 3.4.

3.1 Google

The Google search engine provides a portal to the World Wide Web. We want to use Google to extract data with which we are able to build a substitution model.

Google returns the number of times the word has been found. For example the word **google** has been found 1,660,000,000 times (see Figure 1).

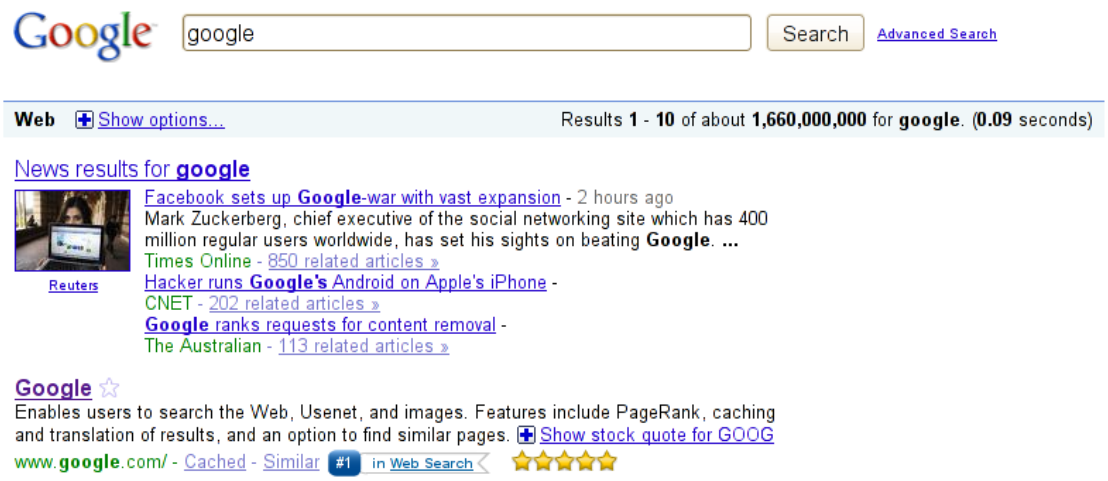

Figure 1: Google result for the word **google**

If a word is misspelled it also returns the number of times the word has been found, but Google also suggests a correction for the word by displaying *Did you mean ...* on top of the web page. For example if we write **gaogle** instead of **google**, google correctly finds out that we might have misspelled the word and suggests **google** as a correction. Google also displays the number of results for the misspelled word (see Figure 2).

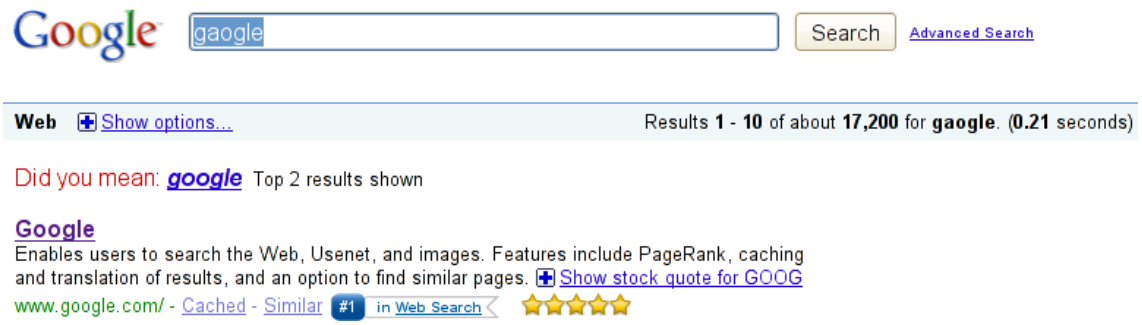

Figure 2: Google result for the word **gaogle**

The idea is to google many words and also the whole neighborhood in respect to substitutions of every word. This neighborhood can be found when we change every letter of every word to the other 25 possibilities. Therefore if we have a word with n letters, we search $1 + n * 25$ different words and remember the number of search hits. With this information it is possible to count the letter pairs frequencies from where a substitution model can be estimated (see section 3.4).

3.2 Database

The words, their mutated derivatives and their number of hits are stored in an Oracle database. Figure 3 displays the model of the database.

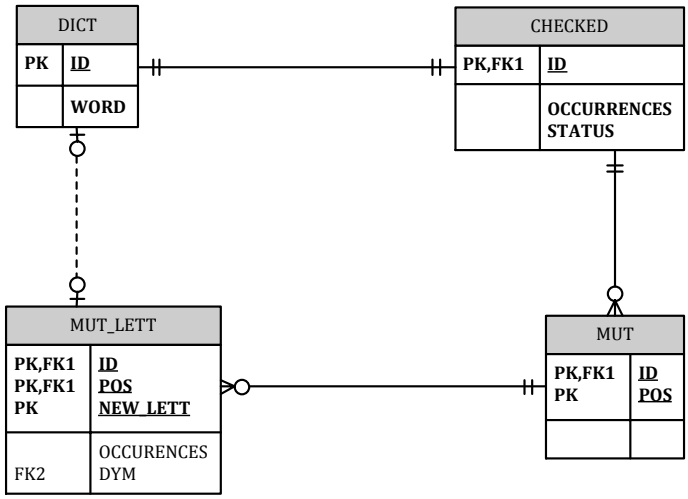

Figure 3: ER model of database

The table DICT saves words of a dictionary we want to google. CHECKED is used to remember the status for the current word and the number of times the word has been found. Then every position of the word is mutated, which is saved in MUT. The mutated letter and the occurrences of the mutated word is stored in MUT_LETT. The DYM attribute stands for "Did you mean" and saves the ID of the word Google suggested in the *Did you mean?* field or -1 if the corrected word is not in the database.

3.3 The Google poller

The Google poller connects to Google and places a search query. The resulting webpage is parsed and the relevant information extracted. This is done with a Perl script, which proceeds as follows:

Step 0 Initialize an object of class LWP::UserAgent, which is able to send requests to a web server. Also initialize a DBI object to access the Oracle database.

Step 1 The next candidate word is fetched from the database and it is status in the table CHECKED is set to *IN PROGRESS*.

Step 2 The word is googled with the LWP::UserAgent object. The response is a single HTML string which is parsed to find the estimated number of results. The parsing is done with simple substitution commands, deleting those characters that don't belong to the information, we try to extract.

Step 3 For every position of the word, change the letter to the other 25 possibilities and google the mutated word. The number of results and the "*Did you mean ...*" field is extracted from the response and written to the database.

Step 4 After every possible mutation of the word has been googled the status of the word in the database is set to *DONE*.

The script is executed for every word in the database.

3.3.1 Detailed description of the Google poller

Step 0 The LWP::UserAgent object implements a local web user agent, which can be used to connect to a webserver and request webpages. This happens in an HTTP style manner, which is a request-response standard for client-server computing (see Fielding *et al.*, 1999). The client is usually called web browser, spider or, more generally, user

agent. The responding server stores resources such as HTML files or images, which are identified by an *Uniform Resource Locator* (URL). An URL usually consists of 3 parts: `protocol://server/resource`. The **protocol** is in most cases `http`, but URLs can also be used to identify resources from other protocols (such as `ftp`, `irc`, etc.). In the `http` specification eight methods exist, which can be performed on resources:

- `GET`: Header of the message (meta-data) and a representation of the resource.
- `HEAD`: Same as `GET` but without message body.
- `POST`: Submits data to a resource.
- `PUT`: Uploads a representation of a resource.
- `DELETE`: Deletes a resource.
- `TRACE`: Returns the same request. This can help the client to see if intermediate servers change the request.
- `OPTIONS`: Returns the HTTP methods that are supported by the server.
- `CONNECT`: Converts the connection to a TCP/IP tunnel.
- `PATCH`: Apply partial modification to a resource.

These eight methods are specified in Fielding *et al.* (1999), but need not implemented in every web server. However the specification requires to have at least `GET` and `POST` method implemented. The header of an HTTP message adds meta-data to the actual request. One can specify accepted encodings (eg. UTF-8 or ASCII), the type of the message body (eg. text, image, video, etc.) and also what type of User-Agent.

We can specify the header used by the `LWP::UserAgent` object. Following options have been explicitly defined:

- *User-Agent = Mozilla/4.76 [en] (Win98; U)*: We emulate a Mozilla web browser running on Windows 98. The language of the browser is English.
- *Accept = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */**: Several image formats are accepted. This is set to make the Google web server believe that we are a real web browser.
- *Accept-Charset = iso-8859-1,*,utf-8*: The agent accepts ISO encoding (US ASCII).
- *Accept-Language = en-US*: Our primary language is US English.

Our client (the `LWP::UserAgent` object) submits an HTTP request by the `GET` method. Originally the `GET` method only requests resources and doesn't allow submission of data to the resource (for this the `POST` method was introduced). This has changed and one can also submit data by attaching arguments to an **URL**, according to following syntax:

```
protocol://server/resource?arg1=value&arg2=value
```

. `arg1` is the name of the variable, which can be processed by the resource on the webserver. For larger amounts of data, one should still use the `SUBMIT` method or, if the data are large binary files, the `PUT` method.

DBI is a database access module for PERL, which defines a set of methods, variables and conventions independent from the actual database used. **DBI** is just an interface, a database driver module is still needed, which implements above mentioned methods, etc. (Bunce, 2010) Since our data is stored in an Oracle database, we use the *DBI:Oracle* module.

The general usage of the **DBI** module is as follows:

1. **connect**: A database handle object is created by connecting with `DBI->connect` to a database.

2. **prepare**: An SQL statement is prepared by calling the `prepare` function of the database handle object. This creates a statement handle, which can be used for later execution.

3. **execute**: The statement is executed with the `execute` function of the statement handle object.

4. **fetch**: If the statement is a `SELECT` statement, table rows can be fetched with the `fetch` command of the statement handle.

5. **disconnect**: Once all statements are executed and the database connection is no longer used, the `disconnect` function has to be called to properly close the database connection.

SQL statements are structured database queries which are used to retrieve data sets from relational databases. In our script we use `SELECT`, `UPDATE` and `INSERT` statements. There also exist other statements like `DELETE`, `CREATE` or `ALTER` (For a detailed description of the SQL standard see International Organization for Standardization (ISO) (1992)).

Step 1 consists of two parts. The first part is to select one word randomly from the dictionary for processing. In the second part we have to update its status in the `CHECKED` table to 1. Since those 2 steps are not atomic, it could happen that another polling process fetches the same random word and two pollers process the same word.

In order to tackle this problem, we created a view. A view is a stored data query in the database, which returns a data set, like every other `SELECT` statement. This view assigns a random number to every so far unprocessed word. The word with the lowest random number is then selected by the poller. The poller script selects the word and inserts the entry for the word in the `CHECKED` table in one atomic step (`INSERT INTO checked SELECT id, 0, ? FROM random_word WHERE rownum = 1`). The field denoted with the `?` in the previous `INSERT` statement, will be substituted by the `execute` command with a parameter. This parameter is another *random number* which is generated by the PERL script and inserted into the `STATUS` field of the `CHECKED` table. This random number is the key to find the word, that was selected for processing. This word is then googled (as explained in Step 2) and it's `STATUS` field set to 1 and the `OCCURENCES` field is updated.

Step 2 This step is executed by the `google` function. It accepts as parameter a word and returns the number of **occurrences** and the **corrected** word, if there is any.

A `HTTP::Request` object is created, which specifies the method (`GET`) and the URL of the resource (`http://www.google.com/search?q=word`). This is then sent with the `request` function of the `LWP::UserAgent` object. The `request` function returns an `HTTP::Response` object, which has an attribute `is_success` that is `true` if the request method completed successfully. From the `decoded_content` attribute of the response object the HTML file can be retrieved as string.

This string is stored in a PERL variable and parsed using the substitution commands of PERL. For example after the string *of about* <*b*> the number of occurrences is listed in the HTML source. In order to extract the number, everything before and after the number is deleted. The same happens with the *Did you mean* field. If the field is present the string *Did you mean* <*b*><*i*> appears immediately before the corrected word. This word is filtered, by deleting the content before and after the word.

Step 3 Now the word is split into its letters and every position is exchanged with any of the 25 possibilities. Then the `google` function is called, which was already explained

in step 2. The position and the changed letter is inserted into the `MUT_LETT` table.

3.4 Estimating a rate matrix

Before constructing a substitution model we have to define and count letter pairs. The number of letter pairs, which represent mutations, are collected in a 26×26 divergence matrix $\boldsymbol{H}$. Where the element h_{ij} is the number of substitutions of letter i by letter j. Apparently the diagonal entries $i = j$ are those cases were i is not substituted.

We do have to fill the divergence matrix $\boldsymbol{H}$ with the number of letter pairs corresponding to each word and each of its mutated derivatives. We fetch from the database the number of times the correctly spelled word w has been found in the world wide web. Let us denote this number with c. For every letter $i = w_k$ in the word we add c to the diagonal entry h_{ii} of the divergence matrix $\boldsymbol{H}$.

For the words where one letter has been substituted with another we compare each position with the correctly spelled word. This gives us a letter-pair (X_i, Y_i) for every position i where X is the original word and Y the mutated word. The number of times this letter-pair has been observed is the number of times the mutated word has been found in the world wide web. This number c is added to the corresponding entries in the divergence matrix. Since we only look at one point mutation, we add, if the word has length n, $n-1$ times c to a diagonal entry and one time c to a non-diagonal entry.

Example 1: Let us assume we have the correctly spelled word *foo*, which we find 100 times. Then we substitute all positions and find every word the number of times listed in Table 1.

aoo	5
boo	10
coo	5
goo	30
fao	5
fbo	10
fco	15

Table 1: Number of times the by point-mutation derived word of *foo* has been found.

We start by adding $c = 100$ to the diagonal entries of the divergence matrix $\boldsymbol{H}$ at the positions $i = 6$ (letter **f**) and 2 times adding c to $i = 15$ (letter **o**).

	f	**o**
f	100	
o		200

Table 2: $\boldsymbol{H}$ after we added the count of *foo* (zero value columns and lines are omitted).

Then we compare the first point-mutated word *aoo* with the correct word *foo*. This gives us the letter pairs $(f, a), (o, o), (o, o)$ which are all found $c = 5$ times. We add c to the corresponding entries in the divergence matrix $\boldsymbol{H}$. Then we compare the next point-mutated word with the correct word and so on. After iterating through all point-mutated words we will have the divergence matrix $\boldsymbol{H}$ listed in Table 3.

	a	**b**	**c**	**f**	**g**	**o**
f	5	10	5	130	30	
o	5	10	15			330

Table 3: $\boldsymbol{H}$ after we added the count of *foo* and the count of all point-mutated derivatives (zero value columns and lines are omitted).

Once we have obtained the divergence matrix $\boldsymbol{H}$ we can calculate our substitution model. The presented method is slightly modified from Weiss and von Haeseler (2003).

$\boldsymbol{H}$ is a quadratic matrix with $n = 26$ rows and columns. The elements on the diagonal are typically larger than the off-diagonal elements (matches are much more likely than mismatches). As a first step we build a symmetric matrix from $\boldsymbol{H}$ ($\hat{\boldsymbol{H}} = \frac{H+H^T}{2}$), because we assume a reversible model, that is, the rate of change of one letter to another is the same in both directions. The second step is to normalize the matrix $\hat{\boldsymbol{H}}$ that all elements sum up to 1 ($\hat{\boldsymbol{H}}$ fulfills Eqn. 9).

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\hat{\boldsymbol{H}}_{ij} = 1 \tag{9}$$

A transition probability matrix $\boldsymbol{P}(t)$ can be calculated:

$$\boldsymbol{P}(t) = \boldsymbol{\Pi}^{-1}\boldsymbol{H} \tag{10}$$

$\boldsymbol{\Pi}$ is a stationary distribution matrix having just diagonal values $\boldsymbol{\Pi} = diag[\pi_A, \pi_B, \ldots, \pi_Z]$. $\pi_A, \ldots, \pi_Z$ are the row sums of $\hat{\boldsymbol{H}}$. $\boldsymbol{\Pi}$ corresponds to the relative occurrence of every letter.

The transition probability matrix $\boldsymbol{P}(t)$ can also be calculated from the rate matrix $\boldsymbol{Q}$ (see: Equation 5). $\boldsymbol{Q}$ has a spectral decomposition

$$\boldsymbol{Q} = \boldsymbol{U} diag[\lambda_1, \ldots, \lambda_{26}] \boldsymbol{U}^{-1} \tag{11}$$

$\boldsymbol{U}$ are the eigenvectors and $D = diag[\lambda_1, \ldots, \lambda_{26}]$ is a diagonal matrix containing the eigenvalues of matrix $\boldsymbol{Q}$ (see also Eqn. 8). With Equations 10 and 11 $\boldsymbol{P}(t)$ can be rewritten as

$$\boldsymbol{P}(t) = \boldsymbol{U} diag[\eta_1, \ldots, \eta_{26}] \boldsymbol{U}^{-1} \tag{12}$$

with $\eta_s = e^{\lambda_s t}, s = 1, \ldots, 26$. We assume that the observed mutations happened in one time step and therefore we set the unit time to $t = 1$. We can calculate the matrix of eigenvalues $\boldsymbol{U}$ and the diagonal matrix of eigenvectors $\boldsymbol{D}$ from $\boldsymbol{P}(t)$ (Eqn. 10). The eigenvalues of the rate matrix $\boldsymbol{Q}$ are given by:

$$\lambda_s = log(\eta_s), s = 1, \ldots, 26 \tag{13}$$

With the spectral decomposition of $\boldsymbol{P}(t)$ (Eqn. 12) and Equations 11 and 13 we compute the rate matrix $\boldsymbol{Q}$.

Finally we need to scale every element q_{ij} in the matrix $\boldsymbol{Q}$ such that time is measured in expected number of substitutions (Eqn. 14, $\boldsymbol{Q}$ fulfills 15).

$$q_{ij} = \frac{q_{ij}}{-\sum_k \pi_k q_{kk}} \forall i, j \tag{14}$$

$$-\sum_i \pi_i q_{ii} = 1 \tag{15}$$

4 Analysis

For the analysis the English dictionary installed with openSuSE 10.3 has been used. This file has 380,645 entries. As a pre-processing step we removed all words having an apostrophe s ('s), all plural nouns that also exist in singular form and all third person conjugates of verbs. The list of words consists then of 75,694 entries.

We tried to restrict the search to web sites in English language. This can be achieved by adding appropriate arguments to the HTTP header. But Google might mark web sites in other languages as English web sites as well and therefore we can't guarantee that all results are from English web sites only.

So far 21,744 of 75,694 (28.7%) words have been googled. At the start of our analysis we googled the words from the dictionary in ascending order. After about 5000 words we improved the script to google the dictionary in random order. Altogether we have googled 3,741,800 one point mutations. We only counted the letter-pairs of mutated words where the google search engine suggested the correct word in the "*Did you mean:*" field. At the end our analysis includes 14,952 words with 1,662,048 one-point mutated derivatives. From these the divergence matrix $\boldsymbol{H}$ was calculated. Table 4) shows the divergence matrix $\hat{\boldsymbol{H}}$ which is a symmetric and normalized matrix.

From the matrix $\hat{\boldsymbol{H}}$ we can calculate the stationary distribution $\boldsymbol{\Pi}$. It should resemble the relative occurrence of every letter in the English language. A comparison of estimated letter frequencies in the literature (Lewand, 2000) and the relative frequencies estimated by our method is listed in Table 5. This table shows an overrepresentation (frequency estimated by us is more than one percent larger) of the letters **a**, **b**, **l** and **r**, whereas the letters **e**, **f**, **h**, **o**, **t** and **w** are underrepresented (frequency estimated by us is more than one percent smaller) in our data set.

We calculated the eigenvalues and eigenvectors with R (R Development Core Team, 2010). The eigenvalues are listed in Table 6.

Then the rate matrix $\boldsymbol{Q}$ (Table 7) is calculated with Equation 11 and 13.

As a visualization a bubble plot is created, shown in Figure 4. The area of the bubbles correspond to the value in the rate matrix $\boldsymbol{Q}$, while the negative values in the diagonal are omitted for the sake of simplicity. We immediately recognize letters with high rates by looking at the size of the bubbles. Letters that have a higher substitution rate have larger bubbles in their row than others. For example the letter **Z** undergoes a substitution approximately 30 times more often than the letters **T** or **R**.

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	
1.2E-1	3.5E-7	8.2E-7	1.2E-6	8.6E-5	1.6E-7	2.8E-7	3.9E-7	1.4E-5	2.0E-7	1.1E-7	4.3E-7	3.5E-7	**A**
	3.3E-2	5.8E-7	2.8E-6	2.0E-7	2.1E-7	1.1E-6	1.4E-6	5.0E-6	8.8E-8	2.6E-7	2.7E-7	8.4E-7	**B**
		3.6E-2	1.0E-6	2.3E-6	2.4E-7	1.0E-6	8.0E-7	3.7E-7	5.6E-7	2.6E-5	4.2E-7	5.8E-7	**C**
			3.4E-2	7.0E-7	5.2E-7	2.4E-6	1.0E-6	3.7E-7	1.2E-7	2.4E-7	3.5E-7	8.8E-7	**D**
				1.1E-1	3.0E-7	4.1E-7	3.4E-7	3.6E-5	1.5E-7	2.7E-7	4.0E-7	1.4E-7	**E**
					1.1E-2	2.4E-6	2.1E-7	3.3E-7	1.5E-7	1.2E-7	1.9E-7	1.5E-7	**F**
						2.4E-2	6.0E-7	2.3E-7	3.5E-6	1.7E-6	1.1E-7	1.7E-7	**G**
							2.9E-2	2.5E-6	9.3E-7	2.0E-5	7.8E-7	4.3E-7	**H**
								7.0E-2	4.5E-6	5.7E-7	4.7E-6	2.1E-7	**I**
									5.9E-3	2.8E-7	4.7E-7	1.9E-7	**J**
										1.1E-2	2.5E-6	2.8E-7	**K**
											5.5E-2	5.9E-7	**L**
												3.2E-2	**M**

N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	
1.1E-6	4.2E-5	7.7E-7	4.2E-7	1.4E-6	2.8E-6	1.3E-6	1.0E-5	1.5E-7	3.6E-7	2.0E-7	5.3E-6	6.2E-7	**A**
2.4E-6	1.8E-7	2.0E-6	4.6E-8	3.8E-7	3.6E-7	3.0E-7	8.5E-8	3.5E-6	5.3E-7	1.3E-8	8.7E-8	2.5E-7	**B**
1.8E-6	1.1E-6	3.9E-7	4.8E-6	1.3E-6	1.4E-5	2.8E-6	2.0E-7	2.0E-6	2.2E-7	5.9E-7	1.1E-7	8.9E-7	**C**
1.0E-6	2.7E-7	4.4E-7	4.7E-7	5.8E-7	6.2E-6	4.3E-6	1.6E-7	2.9E-7	8.6E-8	1.0E-7	1.6E-7	3.0E-7	**D**
8.5E-7	1.4E-5	2.6E-6	3.9E-8	8.7E-6	2.5E-6	1.5E-6	9.5E-6	5.3E-8	1.5E-6	4.5E-8	4.3E-6	8.8E-7	**E**
1.2E-7	7.6E-8	4.0E-7	9.9E-9	4.3E-7	2.2E-6	4.2E-7	7.7E-8	3.3E-7	1.0E-7	5.3E-8	4.6E-8	4.1E-8	**F**
4.5E-7	1.2E-7	1.4E-7	1.5E-6	1.9E-7	2.7E-7	4.0E-7	1.2E-7	1.2E-7	1.2E-7	8.1E-8	2.2E-7	2.3E-7	**G**
1.2E-6	1.0E-7	1.1E-7	9.0E-8	4.1E-7	3.9E-7	5.1E-7	5.1E-7	2.2E-7	4.0E-7	6.5E-8	4.2E-7	3.4E-7	**H**
6.2E-7	1.0E-5	1.2E-6	1.2E-8	7.6E-7	6.6E-7	7.1E-7	7.6E-6	1.1E-7	5.7E-8	5.8E-8	3.3E-5	6.1E-8	**I**
5.1E-7	5.7E-8	1.1E-7	2.1E-8	1.3E-6	2.8E-7	1.2E-7	2.2E-7	6.2E-7	1.9E-7	5.3E-7	1.8E-6	2.0E-7	**J**
1.7E-6	3.8E-7	6.6E-8	2.6E-7	2.1E-7	3.0E-7	7.9E-7	5.3E-8	6.9E-8	6.0E-8	2.2E-7	3.0E-7	4.9E-8	**K**
1.4E-6	3.5E-7	4.2E-7	4.0E-8	1.9E-6	4.8E-7	3.4E-6	3.6E-7	3.9E-7	1.5E-7	1.2E-7	2.5E-7	1.4E-6	**L**
1.2E-5	4.8E-7	1.2E-6	4.6E-8	9.7E-7	2.8E-7	3.5E-7	1.2E-7	3.5E-7	3.2E-7	5.7E-8	3.1E-7	5.2E-7	**M**
7.5E-2	3.6E-7	2.5E-7	9.4E-8	2.0E-6	1.5E-6	1.6E-6	6.8E-7	7.3E-7	2.4E-7	1.5E-7	3.0E-7	3.6E-7	**N**
	6.4E-2	1.1E-6	2.0E-7	2.3E-7	3.6E-7	1.8E-7	9.8E-6	8.0E-8	9.5E-8	3.4E-8	3.5E-6	7.7E-7	**O**
		2.4E-2	5.6E-8	2.2E-7	2.0E-7	5.5E-7	9.1E-8	1.1E-7	4.6E-8	1.6E-8	1.2E-7	3.7E-8	**P**
			1.1E-3	2.7E-8	8.8E-8	4.8E-8	3.1E-8	3.0E-8	6.2E-8	1.3E-8	1.8E-8	4.4E-8	**Q**
				7.3E-2	1.6E-6	4.9E-6	4.5E-7	3.7E-7	2.8E-7	1.8E-7	4.4E-7	2.0E-7	**R**
					5.6E-2	7.8E-7	2.2E-7	6.3E-7	2.5E-7	8.1E-7	3.1E-7	1.9E-5	**S**
						6.4E-2	1.0E-7	4.0E-7	1.6E-7	9.7E-8	3.3E-6	4.5E-7	**T**
							2.9E-2	6.8E-7	1.4E-5	2.2E-8	2.3E-6	3.9E-8	**U**
								1.2E-2	7.4E-6	2.7E-8	6.9E-7	1.3E-7	**V**
									1.3E-2	5.2E-8	1.4E-7	3.5E-8	**W**
										1.7E-3	2.4E-8	2.1E-6	**X**
											1.8E-2	3.4E-7	**Y**
												2.5E-3	**Z**

Table 4: Divergence Matrix H

Letter	Freq. est. by Lewand (2000)	Freq. est. by us
a	8.17%	12.08%
b	1.49%	3.30%
c	2.78%	3.66%
d	4.25%	3.38%
e	12.70%	10.68%
f	2.23%	1.08%
g	2.02%	2.39%
h	6.09%	2.93%
i	6.97%	6.98%
j	0.15%	0.59%
k	0.77%	1.07%
l	4.03%	5.46%
m	2.41%	3.16%
n	6.75%	7.46%
o	7.51%	6.37%
p	1.93%	2.36%
q	0.10%	0.12%
r	5.99%	7.28%
s	6.33%	5.62%
t	9.06%	6.41%
u	2.76%	2.90%
v	0.98%	1.17%
w	2.36%	1.33%
x	0.15%	0.18%
y	1.97%	1.81%
z	0.07%	0.25%

Table 5: Relative Frequencies of Letters

1.00000	0.99972	0.99964	0.99958	0.99957	0.99948
0.99947	0.99933	0.99931	0.99923	0.99916	0.99913
0.99901	0.99894	0.99859	0.99843	0.99839	0.99811
0.99764	0.99710	0.99710	0.99676	0.99634	0.99402
0.99256	0.98829				

Table 6: Eigenvalues of $P(t)$

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	
-1.1704	.0024	.0056	.0082	.5861	.0011	.0019	.0026	.0988	.0014	.0007	.0029	.0024	**A**
.0087	-.5783	.0146	.0692	.0051	.0052	.0267	.0359	.1241	.0022	.0065	.0067	.0210	**B**
.0183	.0132	-1.4619	.0229	.0522	.0053	.0234	.0175	.0084	.0126	.5886	.0095	.0131	**C**
.0293	.0675	.0248	-.6343	.0172	.0126	.0598	.0248	.0090	.0028	.0059	.0086	.0215	**D**
.6628	.0016	.0179	.0054	-1.3360	.0023	.0031	.0026	.2756	.0012	.0021	.0030	.0011	**E**
.0122	.0160	.0180	.0394	.0227	-.7063	.1807	.0162	.0250	.0115	.0094	.0148	.0112	**F**
.0095	.0369	.0359	.0846	.0141	.0817	-.6174	.0206	.0078	.1201	.0584	.0039	.0058	**G**
.0109	.0403	.0218	.0286	.0095	.0059	.0168	-.9574	.0698	.0261	.5609	.0218	.0121	**H**
.1711	.0586	.0044	.0044	.4217	.0039	.0027	.0293	-1.4623	.0534	.0068	.0555	.0024	**I**
.0286	.0124	.0783	.0162	.0211	.0210	.4879	.1303	.6343	-2.3991	.0395	.0664	.0265	**J**
.0082	.0201	2.0158	.0187	.0209	.0095	.1306	1.5420	.0443	.0217	-4.3947	.1968	.0213	**K**
.0064	.0040	.0063	.0053	.0060	.0029	.0017	.0117	.0710	.0071	.0385	-.3299	.0089	**L**
.0092	.0219	.0151	.0231	.0036	.0038	.0044	.0112	.0054	.0049	.0072	.0154	-.5618	**M**
.0121	.0267	.0194	.0111	.0094	.0013	.0049	.0128	.0068	.0056	.0188	.0154	.1289	**N**
.5474	.0023	.0139	.0035	.1804	.0010	.0016	.0013	.1299	.0007	.0049	.0046	.0062	**O**
.0269	.0693	.0137	.0155	.0893	.0139	.0048	.0038	.0404	.0039	.0023	.0147	.0436	**P**
.2970	.0326	3.4262	.3341	.0279	.0070	1.0948	.0644	.0086	.0151	.1839	.0285	.0331	**Q**
.0163	.0043	.0148	.0065	.0984	.0049	.0022	.0046	.0086	.0147	.0024	.0210	.0109	**R**
.0415	.0052	.2038	.0906	.0364	.0324	.0040	.0057	.0096	.0041	.0044	.0070	.0040	**S**
.0169	.0039	.0362	.0555	.0190	.0054	.0051	.0066	.0091	.0016	.0102	.0439	.0045	**T**
.2873	.0024	.0057	.0047	.2699	.0022	.0035	.0145	.2169	.0061	.0015	.0101	.0033	**U**
.0103	.2437	.1374	.0201	.0037	.0234	.0082	.0156	.0075	.0437	.0048	.0272	.0248	**V**
.0223	.0329	.0134	.0053	.0914	.0063	.0073	.0246	.0034	.0119	.0037	.0094	.0198	**W**
.0956	.0060	.2758	.0492	.0208	.0250	.0382	.0305	.0274	.2502	.1057	.0579	.0269	**X**
.2410	.0039	.0052	.0075	.1967	.0021	.0102	.0193	1.5050	.0821	.0138	.0112	.0140	**Y**
.2031	.0827	.2886	.0986	.2867	.0132	.0743	.1122	.0197	.0653	.0159	.4438	.1702	**Z**

N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	
.0074	.2886	.0053	.0028	.0098	.0193	.0090	.0689	.0010	.0025	.0014	.0361	.0043	**A**
.0605	.0044	.0497	.0011	.0096	.0089	.0075	.0021	.0867	.0133	.0003	.0022	.0064	**B**
.0395	.0243	.0089	.1086	.0294	.3133	.0635	.0046	.0441	.0049	.0132	.0026	.0201	**C**
.0245	.0066	.0108	.0115	.0140	.1508	.1052	.0040	.0070	.0021	.0026	.0040	.0074	**D**
.0066	.1076	.0197	.0003	.0671	.0191	.0114	.0732	.0004	.0114	.0003	.0333	.0068	**E**
.0093	.0058	.0304	.0008	.0330	.1684	.0319	.0059	.0254	.0078	.0041	.0035	.0031	**F**
.0154	.0042	.0047	.0531	.0067	.0094	.0137	.0043	.0040	.0041	.0028	.0077	.0079	**G**
.0325	.0029	.0031	.0025	.0115	.0109	.0144	.0143	.0062	.0112	.0018	.0119	.0097	**H**
.0073	.1185	.0137	.0001	.0090	.0078	.0083	.0900	.0013	.0006	.0007	.3903	.0007	**I**
.0715	.0079	.0156	.0030	.1820	.0395	.0172	.0302	.0872	.0269	.0746	.2529	.0282	**J**
.1317	.0292	.0051	.0200	.0163	.0230	.0611	.0041	.0053	.0046	.0174	.0234	.0038	**K**
.0211	.0053	.0064	.0006	.0281	.0072	.0515	.0054	.0058	.0023	.0019	.0037	.0207	**L**
.3048	.0125	.0326	.0012	.0252	.0071	.0092	.0031	.0092	.0084	.0015	.0080	.0137	**M**
-.3646	.0039	.0027	.0010	.0218	.0166	.0180	.0075	.0081	.0026	.0017	.0034	.0040	**N**
.0046	-1.1153	.0148	.0026	.0029	.0046	.0024	.1276	.0010	.0012	.0004	.0453	.0100	**O**
.0086	.0400	-.4410	.0020	.0077	.0068	.0192	.0032	.0038	.0016	.0006	.0043	.0013	**P**
.0672	.1446	.0400	-6.0620	.0189	.0622	.0343	.0219	.0213	.0442	.0096	.0128	.0316	**Q**
.0223	.0025	.0025	.0003	-.3320	.0179	.0551	.0051	.0042	.0032	.0020	.0049	.0023	**R**
.0220	.0053	.0029	.0013	.0232	-.8222	.0114	.0032	.0092	.0037	.0118	.0046	.2751	**S**
.0210	.0024	.0071	.0006	.0627	.0100	-.3789	.0013	.0051	.0020	.0012	.0419	.0058	**T**
.0194	.2806	.0026	.0009	.0127	.0063	.0029	-1.6495	.0192	.4085	.0006	.0664	.0011	**U**
.0515	.0056	.0076	.0021	.0262	.0442	.0280	.0475	-1.3606	.5177	.0019	.0486	.0090	**V**
.0148	.0057	.0028	.0038	.0174	.0155	.0098	.8862	.4548	-1.6763	.0032	.0085	.0021	**W**
.0706	.0160	.0076	.0063	.0826	.3770	.0455	.0103	.0129	.0245	-2.6643	.0113	.9906	**X**
.0138	.1595	.0056	.0008	.0199	.0143	.1484	.1062	.0315	.0062	.0011	-2.6345	.0154	**Y**
.1177	.2510	.0120	.0144	.0655	6.0823	.1472	.0126	.0414	.0112	.6828	.1095	-9.4219	**Z**

Table 7: Rate Matrix Q

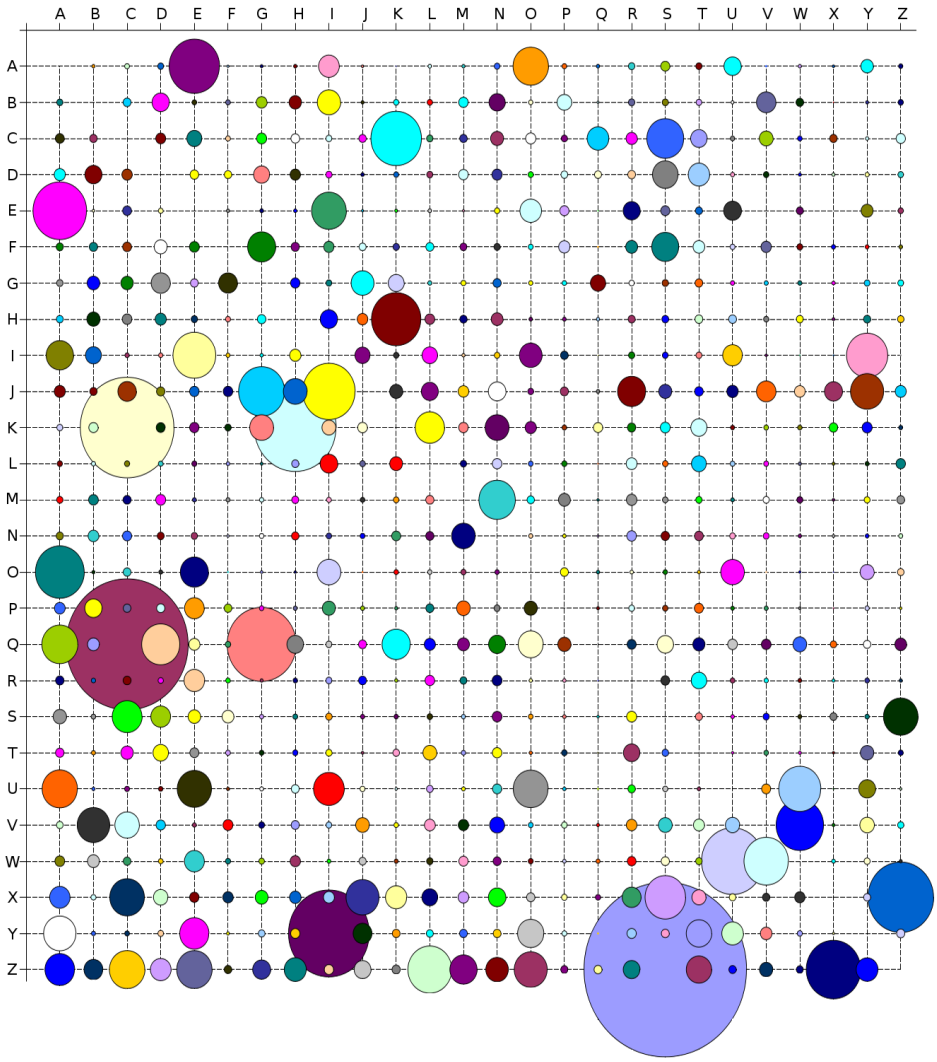

Figure 4: Bubble Plot

The letter with the highest rate is **Z**, having the **Z** ⇒ **S** transition as largest contributor to its rate. The high substitution rate can be explained with differences between American and British English. Words ending in **-ize**, like organize, realize and recognize are written with **z** in American English, whereas British English uses **-ise** and **-ize** (Wikipedia, 2010). The highest contributor to the **K** ⇒ **C** transition is the word **Slovakia** which is spelled **Slovacia** in Romanian.

To further analyze the rate matrix, we order every line corresponding to the rate and substitute the values with their corresponding letter. Additionally we order the lines (letters) according to their total rate that leaves the letter (diagonal element in the rate matrix $\boldsymbol{Q}$). This gives us now a table in which the transition with the highest rate is first, the transition with the second highest rate second and so on (see Figure 6).

The entries in Figure 6 are colored by two criteria. First, yellow entries are all vowels and second, blue entries are the direct neighbors of the original letter (outer left column) on an English keyboard. Neighbors on an English keyboard are those letters, where the key is directly adjacent to another. A letter has at most six neighbors (**G**,**H**,...) and at least two (**P** and **Q**). An example of an English keyboard with letters and their neighbors is given in Figure 5.

Figure 5: US keyboard with selected letters (**Q**, **G** and **P**) and their neighbors

Vowels The coloring of Figure 6 shows that vowels are most likely to mutate to other vowels; except for the **U**, where the U ⇒ W transition is the most likely mutation and the **I**, where the transition I ⇒ Y is more likely than I ⇒ A,O or U. But for all vowels holds that the other 4 vowels appear in the first 5 columns of the table. Also very interesting is the letter **Y**, because the letter **Y** has also all 4 vowels as most likely transitions. The high rates in between the vowels might not just be due to type errors, but rather is a phenomena that people are not able to distinguish between written and spoken language. Especially in the English language many vowels are pronounced differently in different words. And some vowels sound the same, even though they are

Z	S	X	L	C	E	O	A	M	T	N	H	Y	D	B	G	R	J	V	I	K	Q	F	U	P	W
Q	C	G	D	A	K	O	N	H	S	W	P	T	M	B	Z	L	E	U	V	R	J	Y	X	I	F
K	C	H	L	N	G	T	I	O	Y	S	J	M	E	B	Q	D	X	R	F	A	V	P	W	U	Z
X	Z	S	C	J	K	A	R	N	L	D	T	G	H	I	M	F	W	E	O	V	Y	U	P	Q	B
Y	I	A	E	O	T	U	J	V	R	H	Z	S	M	N	K	L	G	D	W	P	C	B	F	X	Q
J	I	G	Y	R	H	V	C	X	N	L	S	K	U	A	Z	W	M	E	F	T	D	P	B	O	Q
W	U	V	E	B	H	A	M	R	S	N	C	J	T	L	Y	G	F	O	D	Q	K	I	X	P	Z
U	W	A	O	E	I	Y	N	V	H	R	L	S	J	C	D	G	M	T	P	B	F	K	Z	Q	X
I	E	Y	A	O	U	B	L	J	H	P	R	T	S	N	K	C	D	F	G	M	V	Z	X	W	Q
C	K	S	Q	T	E	V	N	R	O	G	D	Z	A	H	X	B	M	J	L	P	I	F	W	U	Y
V	W	B	C	N	Y	U	S	J	T	L	R	M	F	D	H	A	Z	G	P	I	O	K	E	Q	X
E	A	I	O	U	R	Y	P	S	C	W	T	Z	N	D	G	L	H	F	K	B	J	M	V	X	Q
A	E	O	I	U	Y	S	R	T	D	N	C	P	Z	L	Q	H	W	M	B	G	J	X	F	V	K
O	A	E	I	U	Y	P	C	Z	M	K	S	N	L	D	R	Q	T	B	G	H	W	V	F	J	X
H	K	I	B	N	D	J	C	L	G	T	U	M	Y	R	W	S	A	Z	E	V	F	P	O	Q	X
S	Z	C	D	A	E	F	R	N	X	T	I	V	L	H	O	B	Y	K	J	M	G	W	U	P	Q
F	G	S	D	R	T	P	V	I	E	C	H	B	L	A	J	M	K	N	W	U	O	X	Y	Z	Q
D	S	T	B	G	A	H	C	N	M	E	R	F	Q	P	I	L	Z	V	O	K	Y	U	J	X	W
G	J	D	F	K	Q	B	C	H	N	E	T	A	S	Z	I	Y	R	M	P	U	O	W	V	L	X
B	I	V	D	N	P	H	G	M	C	W	R	S	A	T	L	K	Z	F	E	O	J	Y	U	Q	X
M	N	P	R	D	B	L	C	Z	O	H	V	A	T	W	Y	K	S	I	J	G	F	E	U	X	Q
P	E	B	M	I	O	A	T	D	L	F	C	N	R	S	G	Y	J	H	V	U	K	Q	W	Z	X
T	R	D	L	Y	C	N	E	A	K	S	I	P	H	Z	F	V	G	M	B	O	W	J	U	X	Q
N	M	B	R	C	K	T	S	L	H	A	D	E	V	U	I	J	G	Z	O	Y	P	W	X	F	Q
R	E	T	N	L	S	A	C	J	M	I	D	U	Y	F	H	B	V	W	O	P	K	Z	G	X	Q
L	I	T	K	R	N	Z	H	M	S	J	A	P	C	E	V	U	O	D	B	Y	F	W	X	G	Q

Legend
Vowels
Neighbors on US keyboard

Figure 6: Transitions ordered by their probability

different. People then tend to write like they speak and therefore producing many typing errors.

Neighborhood If one looks at typing errors, it comes immediately to the mind, that typing errors to letters, that are close on the keyboard to the letter intended to type, are more likely than others. For example we expect that the rate of substitution of a **G** by an **H** is higher than the rate of substituting by a **P**. Looking at Figure 6 the letters that are neighbors to the original are much more concentrated on the left side of the table. We count the blue colored cells per column and add those numbers up until we reach the last column. By dividing the cumulative sum of every column by the total sum of neigbors (blue colored cells), we can calculate the cumulative proportion of neighbors per column. We see that already after the 8th column 50% of all neighbors are included (red line in Figure 7). We can conclude that it is more likely that letters mutate to their neighboring letters, because of typing errors.

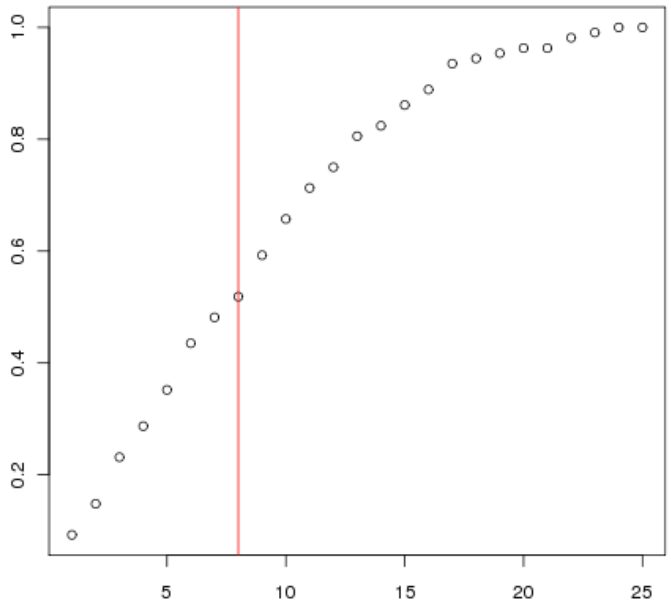

Figure 7: Cumulative distribution of neighbors per column

The **Z** $\Rightarrow$ **S** has been earlier explained by the American and British English differences, but **Z** and **S** are also neighbors on the keyboard. Which of those two has now the higher effect on the substitution rate? If we compare the **Z**-**S** neighborhood to other

neigborhoods and calculate their probabilities by using the numbers of our rate matrix $\boldsymbol{Q}$ we can see that the probability of $\mathbf{Z} \Rightarrow \mathbf{S}$ is much larger than the probability of other comparable neighborhoods (see Table 8). Therefore we are able to conclude that the high substitution rate is largely due to the American and British English differences.

Transition	Probability
$\mathbf{Z} \Rightarrow \mathbf{S}$	$6.0823/9.4219 \approx 0.646$
$\mathbf{J} \Rightarrow \mathbf{I}$	$0.6343/2.3991 \approx 0.264$
$\mathbf{F} \Rightarrow \mathbf{T}$	$0.0319/0.7063 \approx 0.045$
$\mathbf{S} \Rightarrow \mathbf{E}$	$0.0364/0.8222 \approx 0.044$
$\mathbf{A} \Rightarrow \mathbf{W}$	$0.0223/1.1704 \approx 0.019$
$\mathbf{X} \Rightarrow \mathbf{D}$	$0.0492/2.6643 \approx 0.018$

Table 8: Comparison of different neighborhood probabilities

Proper nouns are also included in our dictionary. With the database we can find out which words contribute with what amount to which transition. When we extract the exact list of words per transition, we often find transitions that appear in proper nouns (e.g. Slovakia is also a proper noun). This is not surprising since most people search for proper nouns and also the search engine rather indexes proper nouns than other words, giving proper nouns a higher priority in web search. Table 9 lists the top ten counts of the **K** to **C** transition, with the substituted letters marked in bold. Nine of ten are proper nouns.

word	counts
Slova**k**ia	465000
Blac**k**pool	15800
Ale**k**sandr	13500
Yor**k**shire	7180
Bur**k**e	3340
bro**k**en	2540
Wil**k**es	2500
Nik**k**i	2430
Mic**k**ey	2220
Spar**k**s	2040

Table 9: Top 10 of the K $\Rightarrow$ C transition

5 Conclusion

We presented a way to estimate substitution models for a language. We were able to explain some characteristics of our substitution model. Those are due to the sound similarity of vowels, the neighborhood of letters on an English keyboard and the over-representation of Proper Nouns in our and Google's database.

A drawback to the current solution is, that the Google search results change from day to day and therefore we would need a method which continuously reestimates the substitution model. Additionally our method uses the HTTP protocol to get the Google results. This has the disadvantage that we download the whole webpage, but only need the number of search results and the contents of the *Did you mean* field. This might be improved by getting access directly to the Google databases.

5.1 Future work

The possible applications for such substitution models range from the improvement of spell checkers to estimating phylogenies of languages. In spell checking one misspelled word can have various corrections. For example the word *acress* can be corrected to *actress*, *access*, *acres* and so on. Intelligent spell checkers list the most common word first. Having a substitution model can help us to calculate a probability for each correction. The *correct* spelling program already tries such an approach, by assigning probabilities to each of the possible corrections and suggesting those corrections to the user first, which are more likely (Kernighan *et al.*, 1990). The matrices in (Kernighan *et al.*, 1990) are trained starting from a uniform distribution using a training set of spelling errors. Our matrix has the advantage being estimated from a much larger data set than the matrices in (Kernighan *et al.*, 1990).

Our substitution model can be used to estimate a phylogenetic tree. Although this tree reconstruction is probably limited to Latin languages. For such a phylogenetic tree an estimation of insertions, deletions and transversions might also be necessary.

Furthermore we would like to estimate a substitution model for insertions, deletions and transversions. This can be easily implemented by adding a `INS` and a `DEL` table to the database and a small modification to the polling process.

List of Figures

List of Tables

References

Bunce, T. (2010) Dbi. `http://search.cpan.org/perldoc?DBI`.

Fielding, R., Irvine, U., Gettys, J. and Mogul, J. (1999) Hypertext transfer protocol – http/1.1. `http://tools.ietf.org/html/rfc2616`.

International Organization for Standardization (ISO) (1992) Information technology - database language sql. `http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt`.

Jukes, T. H. and Cantor, C. R. (1969) Evolution of protein molecules. *Mammalian Protein Metabolism*, **III**, 21–132.

Kernighan, M. D., Church, K. W. and Gale, W. A. (1990) A spelling correction program based on a noisy channel model. *Proceedings of COLING*.

Lewand, R. E. (2000) *Cryptological mathematics*. The Mathematical Association of America.

R Development Core Team (2010) *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, iSBN 3-900051-07-0.

Strimmer, K. and von Haeseler, A. (2009) Genetic distances and nucleotide substitution models. In Lemey, P., Salemi, M. and Anne-Mieke, V. (eds.), *The Phylogenetic Handbook: a Practical Approach to Phylogenetic Analysis and Hypothesis Testing*, 2nd edition, pp. 111–141, Cambridge University Press, Cambridge.

Tavaré, S. (1986) Some probabilistic and statistical problems on the analysis of DNA sequences. *Lec. Math. Life Sci.*, **17**, 57–86.

Weiss, G. and von Haeseler, A. (2003) Testing substitution models within a phylogenetic tree. *Mol. Biol. Evol.*, **20**, 572–578.

Wikipedia (2010) American and british english spelling differences. `http://en.wikipedia.org/wiki/American_and_British_English_spelling_differences`.

Wilkinson, D. J. (2006) *Stochastic Modelling for Systems Biology*. CRC Press.