MORPHOLOGICAL ANALYSIS OF WORDS USING GENETIC ALGORITHMS

by

Ernesto A. Pérez

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS
2013

Approved by:

_____	_____
José Fernando Vega Riveros, PhD President, Graduate Committee	Date
_____	_____
Hilton Alers-Valentín, PhD Member, Graduate Committee	Date
_____	_____
Bienvenido Vélez, PhD Member, Graduate Committee	Date
_____	_____
Reyes M. Ortiz-Albino, PhD Representative of Graduate Studies	Date
_____	_____
Pedro Rivera Vega, PhD Chairperson of the Department	Date

ABSTRACT

Morphology is the part of grammar concerned with word formation. It explains why people sometimes immediately understand words when coming across them for the first time. It is also useful in computer search queries by helping to determine related words through word analysis. This thesis investigates the use of genetic algorithms as a means to perform *morphological analysis* of words. Spanish is used to make tests since it has a richer morphology than English. The genetic algorithm system naturally improves its capability to perform morphological analysis as more words are taught to it. The system makes use of a custom word similarity measure algorithm during the morphological division process.

RESUMEN

La morfología es aquella parte de la gramática a la que le concierne la formación de palabras. Esto explica porqué las personas a veces entienden inmediatamente aquellas palabras con las que se encuentran por primera vez. También es útil en búsquedas por computadora al ayudar a determinar palabras relacionadas por medio del análisis de las mismas. Esta tesis investiga el uso de algoritmos genéticos como un medio para llevar a cabo un *análisis morfológico* de palabras. El español se utiliza para hacer pruebas ya que la morfología de esta lengua es más detallada que la del inglés. Por lo tanto se hace más retante llevar a cabo este análisis de palabras en español que en inglés. El sistema basado en algoritmos genéticos naturalmente mejora su capacidad para llevar a cabo divisiones de palabras a medida de que se le enseñen más palabras. Durante el proceso de división morfológica, el sistema hace uso de un algoritmo de medida de similitud de palabras.

To my family

ACKNOWLEDGEMENTS

I am grateful for the extensive help of the professors José Fernando Vega Riveros, Hilton Alers-Valentín and Bienvenido Vélez. I would not have been able to complete my thesis without their help. I also would like to acknowledge my family. Without their support, this work would not have been possible.

Table of Contents

Table List

Figure List

Listing List

1. INTRODUCTION

This thesis deals with morphological analysis. What is morphology? *"It's the part of grammar concerned with words and word formation. The most important component of the word structure is the morpheme, the smallest unit of language that carries information about meaning or function."* [1, 2]

The following morphological division of the spanish word *gatitas* (= *female kittens*) is useful in understanding morphology:

gat	-	*it*	-	*a*	-	*s*
(species)		(size)		(gender)		(number)

If we substituted the *gat* morpheme with the *perr* morpheme, the word would become *perritas* (= *small female dogs*). If we substituted the *it* morpheme with the *ot* morpheme, the word would be become *gatotas* (= *large female cats*). If we removed the *it* morpheme, the word would become *gatas* (= *female cats*). If we substituted the *a* morpheme with the *o* morpheme, the word would becomes *gatitos* (= *male kittens*). If we removed the *s* morpheme, the word would become *gatita* (= *single female kitten*). An example of English morphology can be seen in the following words from [1]:

- *act*
- *act-ive*
- *act-iv-ate*

- *re-act-iv-ate*
- *re-act-iv-at-ion*

In the above example, the morphemes *ive* and *iv* are variants of one another: they are written differently and appropriate to the word in which they are being used, but they carry the same meaning. The same can be said about the morphemes *ate* and *at.* However, since the morphemes *ate* and *at* are pronounced differently, they are considered to be *allomorphs* of one another [1, 3]. There is no allomorphic variation when there is no change in pronunciation. For example, when comparing *create* with *creat-ive*, there is no allomorphic variation since there is no difference in pronunciation between the morphemes *create* and *creat-* in this case. On the other hand, when comparing *electric* with *electric-ity,* there is allomorphic variation between *electric* and *electric-* even though the spelling is the same. [1]

Stemming is the process by which we reduce words, through the removal of morphemes, until we arrive at that which is the *root* that provides the core meaning of the word [4]. Consider the word *reactivation.* We can either remove *re* to obtain *activation,* or we can remove *ion* to obtain *reactivate.* From either of these two new words we can remove another morpheme to obtain *activate*, and we can continue the process by removing *ate* to obtain *active,* and finally by removing *ive* to obtain *act* which is the root of *reactivation.* In-depth coverage of stemming is provided by [5, 6].

A morpheme that can be a word by itself is called a *free morpheme*, whereas a morpheme that must be attached to another element is a *bound morpheme*. For example, *act* is a *free morpheme* whereas *re* is a *bound morpheme* [1, 3].

Morphological analysis is the process we used to perform morpheme identification. This analysis is used in different areas such as spelling error correction [7], machine translation [7, 8], speech recognition, information retrieval, text understanding, and statistical language modeling [8]; text to speech synthesis, hyphenation, and other language engineering tasks [9]. Google has been using stemming for its search and retrieval functions since at least 2006 [10].

According to [7], *"The complexity of word form morphology varies widely among the world's languages, but is regarded quite high even in the relatively simple cases, such as English."* Furthermore, the Spanish language has a richer morphology than English [11]. In-depth morphological analysis has been performed for a variety of languages. In [12], Tang analyzed the English language, in it a series of morphological rules were derived from word lists by means of statistical learning. Papers [13], [14], [15] and [16] analyzed the Spanish language. Tzoukermann and Liberman [13] used a directed graph, where each directed edge is labeled with word relations such as spelling, pronunciation, morphosyntactic features and others. According to the online Oxford Dictionaries, morphosyntactic is defined as involving both morphology and syntax. A program searches this graph to find all sequences of relations with some interesting property, such as the form of a certain spelling, a morphological division, etc. The graph is generated from approximately 55,000 basic words taken from a

dictionary. Velásquez et al [14] perform morphological analysis through generation: given a word, an hypothesis is formulated based on dictionary data and other criteria, and forms are generated based on that hypothesis. The generated forms are then compared to the original word and if a form coincides, then the hypothesis is considered correct. Moreno and Goñi [15] provided a Prolog implementation for morphological analysis with only six DCG rules, relying on a dictionary of more than 43,000 entries. A DCG, that is, a Definite Clause Grammar, represents a grammar as a set of definite clauses in first-order logic. They are usually associated with the Prolog language. According to the paper, only correct forms are analyzed. Zapata and Edison [16] used a lexicon of model verbs that group conjugations of a large set of verbs. Templates are used to store the resulting data in plain text files, using one file per template. This approach did not require datasets as large as those used in other systems. Al-Sughaiyer and Al-Kharashi [17] discussed a variety of techniques regarding the morphological analysis of the Arabic language, none of which were based on genetic algorithms. Van den Bosch and Daelemans [7] analyze Dutch, they used a windowing system to sequentially classify various “task instances” of the word being analyzed. In this case, task instances are word substrings that are to be classified. When the system comes across new instances, the system searches for the best matching instances in memory, and it then determines the classification of the new instance. When all task instances of a word have been classified, the morphological analysis is complete. Monson et al [18] discussed a system called ParaMor, which analyzes Spanish, English, and German. The system first finds sets of mutually exclusive strings such that each set is very similar to one of the inflection classes of a language. (Each inflection class is a set. An example of an inflection class for English is the

set of words that end in a silent *e*). Words are arranged in schemes, which are the sets of stems and suffixes; such schemes are arranged as nodes in a network. The ParaMor system searches the network for the schemes that cover portions of the true inflection classes of the language. Later, inflection classes are clustered and filtered. Finally, word segmentation occurs.

This thesis presents a system that seeks to delimit all of the morphemes of a word without having to carry out the stemming process.

1.1 System overview

In order to delimit the morphemes of a word, genetic algorithms as in [19] are used, which are a form of reinforcement learning [20]. Given that the system has learned a number of words and their morphological divisions, the system can be asked to compute the morphological division of a new word based on what the system has learned. The block diagram shown in Figure 1 illustrates the basics of this system.

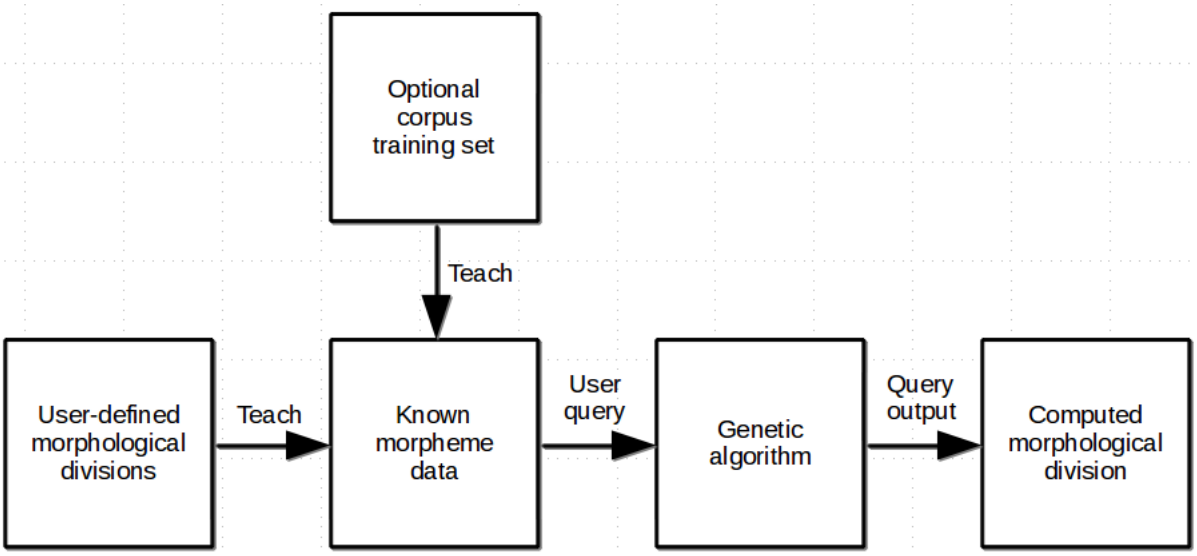


Figure 1. System Basics

The system has knowledge about some morpheme data that it has learned. The system learns these from morphological divisions that are taught either by the user at run-time and/or, optionally, by a corpus that is fed to the system at the moment in which it starts. The user can keep teaching and making queries at any moment. When a user query is made, the query is fed to the genetic algorithm which outputs a computed morphological division. This division will be correct if there is a reasonable amount of known morpheme data relevant to the query that is made. As shown later on Table 4 of Chapter 5, excessive knowledge of many different kinds of morphemes when the system is neither recording position nor frequency data pollutes the system to an extent where the genetic algorithm is not able to reliably compute a correct result.

As explained below, the fitness function makes use of a word similarity measure algorithm. A number of papers have been written regarding the matter of word similarity

measure [21, 22, 23]. In paper [21], the author's goal was the automatic generation of language-dependent string matching functions in order to improve the recognition of string similarity. There is a focus on the recognition of cognates from specific language pairs. In [22], the concept of *string block edit distance* was examined. Two strings A and B are compared by extracting collections of substrings and placing them into correspondence. There are variations of the problem depending on whether the strings must be matched in their entirety, and whether overlap is permitted. In [23], Mackay and Kondrak described a system to compute word similarity based on Pair Hidden Markov Models, a technique adapted from the field of bioinformatics. This technique uses training data that consists of word pairs known to be similar, as in [21], the tests focus on the identification of cognates. The system described in this thesis resembles that of paper [22] at a very basic level, however, the system uses different metrics. Also, this system uses a word similarity measure algorithm that is tailored to the Spanish language by considering diacritics and pronunciation similarity.

The standard operators of selection, crossover and mutation are to be used during the genetic algorithm process. After a number of generations, there is an expectation that the fittest chromosome represents the correct morphological division of the query's word. The genetic algorithm used in this system is discussed in more detail in Chapter 3.

There are similar works that use genetic algorithms to perform morphological division of words. This work seeks to perform full morphological division of words, which

may result in more than two morphemes per word, whereas the work presented in Gelbukh et al [24] is concerned only with a two-part prefix/suffix division.

Papers [25] and [26] analyzed French using genetic algorithms. Kazakov [25] performed two-part division of words recursively. The system divides a list of words into the smallest possible sets of stems and suffixes that it can find. Then, for each of the stem and suffix sets, the process is repeated until no more stem-suffix divisions can be made. Kazakov and Manandhar [26] extend the work presented in Kazakov [25]. The authors presented techniques for the generation of word segmentation rules (logic clauses in Prolog form) given a list of words. The result is a logic program that can be used for the segmentation of previously unseen words. Both [25] and [26] perform their work on a list of undivided words whereas the system described in this thesis depends on being taught the morphological division of words.

In this thesis we discuss a few distinct modes of operation for the genetic algorithm used in the morphological division of spanish words. First, we describe an algorithm that determines, as a percentage, the similarity of two words. Consider, for example, that after a number of spanish words have been learned, we want the algorithm to make a best guess of the conjugation of an irregular verb that the algorithm has learned. For example, given that it knows that the word *poner* (= *to put*) is divided as *pon-er,* the algorithm correctly guesses that the irregular conjugation *pondré* (= *I will put*) is divided as *pond-ré.* (The reverse, concluding *pon-er* given the knowledge of *pond-ré*, is not always correct and will depend on the information that the system has learned; namely, that the system has learned at least one

verb that ends in *-er,* such as *com-er* (= *to eat*).) Thus, the algorithm that determines the similarity of two words is fundamental to the genetic algorithm described in this thesis.

In Chapter 3 we discuss the genetic algorithm itself. There are two variations. Furthermore, for each of the two variations we will discuss how the genetic algorithm works from scratch (i.e., having learned nothing) vs. how the system runs given a corpus that consists of a number of words that have been previously morphologically divided.

1.2 Thesis contribution

This thesis uses genetic algorithms in such a way that words are morphologically divided all at once. Other systems that use genetic algorithms perform the division by recursively dividing words into two parts, as described in Section 1.1. In addition, this thesis focuses on the Spanish language, since it has a richer morphology than English.

The system is tested with a corpus as well as without. Furthermore, the system is tested in two different modalities: one that records the position and frequency of each morpheme in candidate charts, and one that does not. Details are given in Section 3.2, which describes the fitness function.

A significant contribution of this thesis is a custom word similarity algorithm, tailored to the Spanish language. Tests are run and compared with two different formulas: the average formula, discussed in Section 4.4, and the product formula, discussed in Section 5.4.

Appendix A shows the Java program that implements the genetic algorithm and the word similarity measure algorithm.

2. METHODOLOGY

The tests described in this thesis uses genetic algorithms to study the problem of performing correct morphological division of spanish words. Genetic algorithms seem to be a reasonable tool for the solution to the problem, and this thesis proves that they are indeed a very useful tool. As described in Chapter 1, there are papers ([25, 26]) that describe the recursive division of words into two parts until the word is fully divided. In contrast, this system divides a word all-at-once when queried about it.

A custom word similarity measure algorithm for the Spanish language was developed for this system. To design the measure of similarity a heuristic approach was used. A series of pairs of spanish words of various kinds of similarity were written down in paper: words were classified in five categories according to how similar they could be perceived. Some of these pairs were very similar, others were fairly similar, somewhat similar, slightly similar, and not-at-all similar. Then a word similarity measure metric was developed to compare pairs of spanish words. If the metric yields 100% both words should be the same, if the metric yields 0% then there should not be letters in common. The metric yields various other percentages in all other cases. The important part here is that if pair A scores higher than pair B, then pair A has more similarity among its words than pair B. For example, the pair of words “hice” and “rehice” should score a higher similarity value than the pair of words “hice” and “playa”. In order to be able to compute this metric, we developed 9 different scores. These are:

(1) the order score, which measures the relative order of letters between two similar word segments,

(2) the letter similarity score, which distinguishes between letters that have diacritics from those that do not,

(3) the size score, which measures how much a pair of word segments includes letters from both source words,

(4) the segment *pair* score, which is the result of combining the three previous scores, either by an average formula or by a product formula,

(5) the segment *pairs* score, which is the highest score from all segment *pair* scores,

(6) the position score, which measures how much variation is there between the relative positions of a pair of word segments,

(7) the coverage score, which measures how much all word segments include letters from both source words,

(8) the non-crossing score, which measures a pair of *a pair of* word segments, such that it yields 100% if both words of a pair of word segments lies to the left of both words of the other pair of word segments, yielding various other values otherwise, and

(9) the final result, also known as the word similarity score, which is the final metric that measures the amount of similarity between two words.

These scores are described in detail throughout Chapter 4.

A corpus of over 640 words was built and used to train and test the system. Each of the entries in this corpus is a spanish word that has been previously morphologically divided by the author. About 590 words are used to train the system and the remaining 50 words are used to test it. The corpus was built using [3] and is listed in Appendix B. An attempt was made to find a pre-existing corpus of spanish words that are morphologically divided, but no such corpus was found. According to the *corpora-request* mailing list, which is a very busy mailing list on the matter of corpora, no such dataset exists. The *corpora-request* mailing list can be reached at http://mailman.uib.no/listinfo/corpora. As for the data used by papers [13], [14], [15] and [16], the input are plain words that have not been previously morphologically divided.

Tests were run with each corpus entry being taught to the system 15, 25, 35 and 50 times. The tests performed best overall when each corpus entry was taught to the system 35 times. This is explained in Section 5.5. This learning is about updating some morpheme charts as will be described in Section 3.2. After that, a total of 50 tests were run. For each test, the system is asked to morphologically divide a spanish word that is not in the corpus. It is at this point where genetic algorithms are used. They are described in more detail in Chapter 3. Each chromosome is a hypothesis that represents a possible morphological division. The winning chromosome is, according to the genetic algorithm, the best hypothesis regarding the morphological division that can be applied to the word that is being queried. For each of those 50 tests the correct division is known as part of the test. For each test, if the winning chromosome indeed represents the correct division, the test result is recorded as an

“immediate result”. Otherwise, the correct division is taught to the system 35 times and the test is run again for the same word. If the winning chromosome represents the correct division, the test result is recorded as “requires teaching”. Otherwise the test result is recorded as “did not learn”. The result of these experiments as well as an explanation of these results is described in Chapter 5. The system was also tested with an alternative formula (the product formula) and the results were compared to the original formula (the average formula). These are described in Section 5.4.

The program was written in Java since it is a computer language that is easy to use. The program was run on an 1.86GHz 1GB Intel computer. This computer is over 10 times faster than the computer used in [25]. None of the other two papers that deal with morphology and genetic algorithms specify which kind of computer was used to run the tests. The outcome of the tests is completely independent of the speed of the computer.

3. The System And The Genetic Algorithm

In the software's learning system, there are two operations that can be carried out: one is to teach the morphological division of a new spanish word to the system; the other is to ask the system to compute the morphological division of a word that has been given.

To teach the system, we type an equals sign followed by a morphologically divided spanish word. For example,

=*viv-ir*

=*viv-i-mos*

=*viv-e*

To query the system, we simply type a single spanish word, and the system will output what it believes is the best morphological division for the word. For example, given that the system has learned enough about spanish verbs that end in *er,* then upon asking about *comer* it should come up with *com-er* as a solution.

There are two important details here. First, sometimes two or more morphological divisions will tie for the best solution. Second, the system will sometimes output inconsistent results given the same input. All of this is related to the kind and amount of information that the system has learned at the moment it is queried, in addition to genetic algorithms being stochastic.

3.1 The chromosomes

The goal of the genetic algorithm is to determine the morphological division of a word it has been given. Since the chromosomes represent potential solutions to the query, they are made up of a series of bits such that the presence of a 1 represents a boundary between morphemes, and the presence of a 0 represents the lack of such boundaries. For example, if we query the system about the word *comer,* then the chromosome 1100 represents the morphological division *c-o-mer.* (Observe that the length of the chromosome is one less than the length of the word.) When the query first runs, it begins with a pool of 400 such chromosomes, which are generated at random. There is always the possibility of there being multiple chromosomes which are the same, especially with shorter words.

Originally, the genetic algorithm started out with a pool of only 200 chromosomes, however, in order to better handle larger words the amount was raised to 400 chromosomes. Since the genetic algorithm performed satisfactorily with this amount, as it can be seen in the right column of tables 4 and 6, the value was kept permanently.

3.2 The fitness function

We now compute a fitness score for each chromosome. The system has two modes of operation. In one, the fitness function seeks to match each of the morphemes delimited by the chromosome to the morpheme it most resembles among all it has learned. It is at this point where the system uses the word similarity measure algorithm. Consider an example where

the system is initialized from scratch (i.e., it has learned nothing), and we teach it only the following three morphological divisions:

=*viv-ir*

=*viv-i-mos*

=*viv-e*

This means that the system knows about the morphemes *viv, ir, i, e,* and *mos.* Now, suppose we query it about the word *comer* and we have a chromosome with the makeup 1100 (representing *c-o-mer*). Then for each of the tentative morphemes *c, o* and *mer,* the system will find the morpheme among *viv, ir, i, e* and *mos* that most resembles the tentative morpheme. For *mer,* the system determines via the word similarity measure algorithm that the learned morpheme it most resembles is *mos.* Likewise for *o.* For *c* the word similarity score yields zero since none of the learned morphemes resembles it. In this mode of operation, the highest word similarity score for each match ($c \rightarrow 0$) ($o \leftrightarrow mos$) ($mer \leftrightarrow mos$) becomes the fitness value of the chromosome.

The second mode of operation is an extension of the first one. In addition to considering the word similarity score for each match, the system considers position and frequency data. When the system learns, it remembers the relative positions and frequency of each morpheme. For example, the morpheme *er* tends to be at the end (such as in *beb-er* and *com-er*), and also fairly close to the end (such as in *hacer-er-le*). Whereas the morpheme *super* tends to be at the beginning, unless we teach it something like *super-super-buen-o,* in

which case it will learn that it can be in the middle. The way the position and frequency data is learned can be thought to be similar to a stem graph with position percentages in the horizontal axis and frequency in the vertical axis. An example is shown in Figure 2.

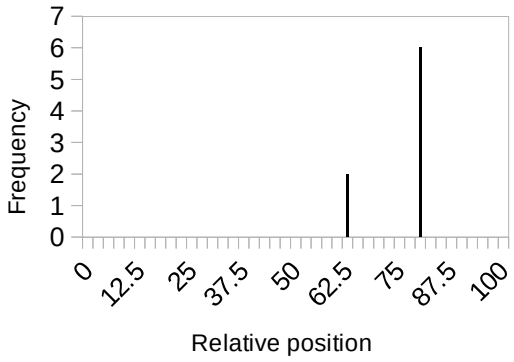


Figure 2. Possible Data For Morpheme *er*

We now proceed to compute a *scaling factor* to be used in the fitness function. For every relative position for which we have recorded frequency data the scaling factor will be 100%. For all other relative positions, the scaling factor will be less than 100% and greater than or equal to 50%. This is shown in Figure 3. It is desirable that the lowest scaling factor be a value higher than 0% since it is to be used in a multiplication. The lower scaling factor was chosen to be 50% and it was found to yield excellent results. This can be seen in the discussion of experimentation of Chapter 5.

Observe the point close to relative position 70%, which has a scaling factor of 50%. Note that this point is not exactly in the middle between the two relative positions 62.5% and 80% (the ones having recorded frequency values). Rather, it lies toward the left. That is because the right value (a frequency of 6 for relative position 80%) is greater than the left

value (a frequency of 2 for relative position 62.5%). In case that the value for relative position 80% keeps increasing, then the relative position whose scaling factor is 50% will keep moving towards the left, asymptotic to the left relative position. Likewise, if the left relative position had a frequency greater than the right relative position, the relative position whose scaling factor is 50% will be somewhere within the right half. If both relative positions had the same frequency, the relative position whose scaling factor is 50% will be right in the middle.

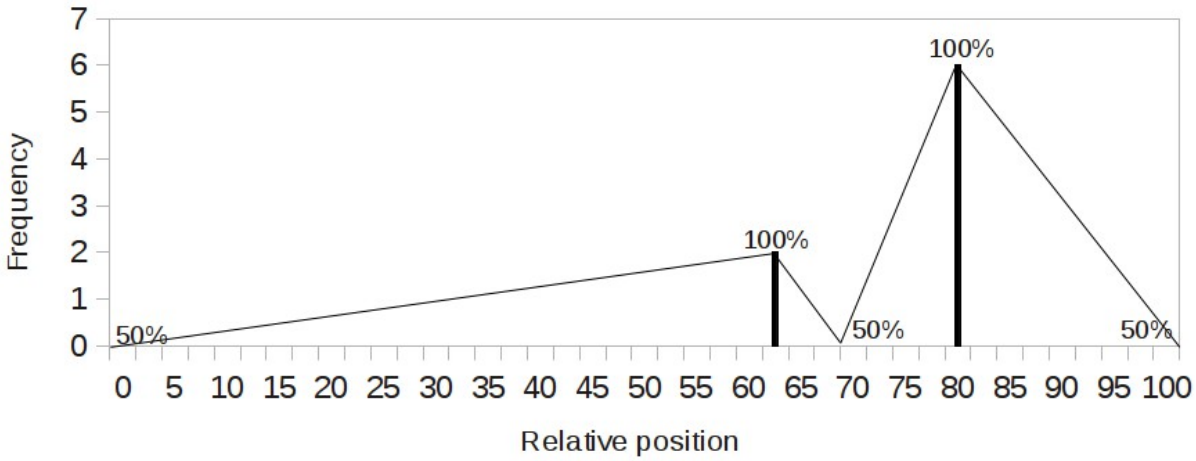


Figure 3. Sample Scaling Factor Chart

When the system is evaluating the fitness score for a chromosome, it first picks one of its morphemes. We call it the *check morpheme.* The relative position (0% to 100%) of the check morpheme within the chromosome is known as the *check position.*

The system now runs through every morpheme that it has learned and compares them to the check morpheme. Each of those morphemes is known as a *candidate morpheme.* The

system then retrieves the *candidate chart* for the candidate morpheme, this is a chart like the one shown in Figure 3.

The check position is looked up within the candidate chart and a scaling factor is obtained. Next, a similarity measure (0% to 100%) is obtained by comparing the check morpheme to the candidate morpheme using the word similarity measure algorithm described in Chapter 4. The similarity measure is multiplied by the scaling factor in order to obtain the *morpheme similarity score.*

The morpheme similarity score is computed for every check morpheme within the chromosome. The highest morpheme similarity score among these becomes the fitness value for the chromosome.

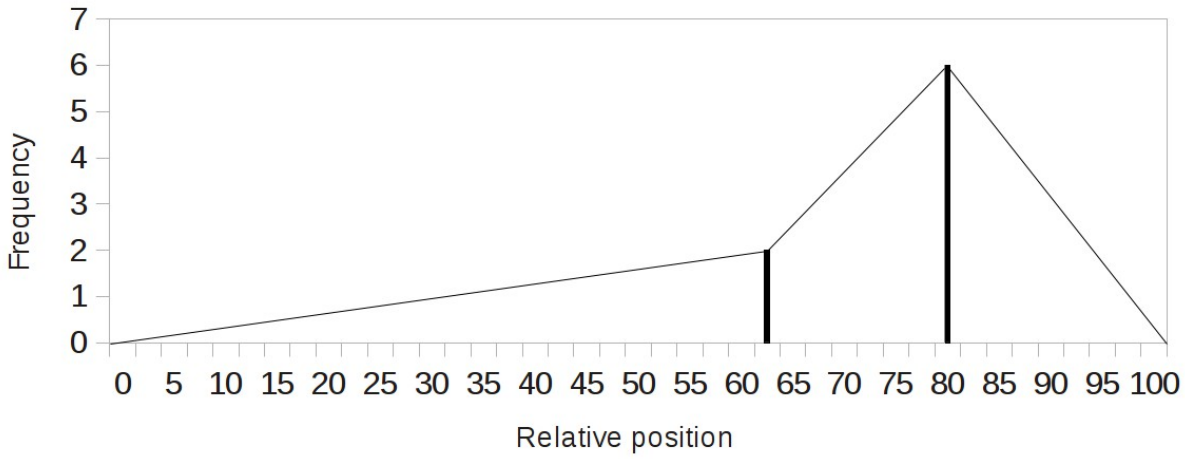


Figure 4. An Old Version Of The Scaling Factor Chart

Originally, the scaling factor was defined differently. An old version of the scaling factor chart is shown in Figure 4. The scaling factor used was computed by a straight line between each pair of relative positions for which data had been recorded. This type of scaling

factor computation gave poor results on the experiments. On the other hand, the scaling factor chart shown in Figure 3 provided excellent results as can be seen by the experimentation described in Chapter 5.

3.3 The selection process

All of the chromosomes in the pool are sorted descendingly by their fitness score. Whenever two or more chromosomes have the exact same makeup (i.e., they are literally the same), only one instance of them is kept during the sorting process.

Next, we divide the pool in two halves, such that the lowest scoring chromosome of the top half scores more than the highest scoring chromosome of the bottom half. This implies that the halves are not necessarily of the same size. When this is the case, the top half will be bigger than the bottom half. Consider for example a pool of 8 chromosomes with scores 80, 70, 60, 60, 60, 40, 35 and 10. Then the top half will turn out to be 80, 70, 60, 60 and 60; and the bottom half 40, 35 and 10.

The top half becomes the mating pool, i.e., its chromosomes become candidates for mating. Selection is made using weighted probabilities. Each mating candidate is assigned a relative score, which is the difference between its fitness score and the fitness score of the lowest scoring chromosome of the bottom pool. Those values are converted into probabilities that add to 1 by dividing each relative score by the sum of all relative scores. The system then uses those probabilities to randomly choose enough couples that go on to become parents and

generate an amount of offspring chromosomes that is equal to that in the initial pool. The amount is not always exactly equal due to the use of elitism, which is discussed below.

3.4 The next generation

Each couple of parents mate to create two offspring. According to [29], *"Two-point crossover performs the same task as one-point crossover (i.e., exchanging a single segment), but is more general. [...] Researchers now agree that two-point crossover is generally better than one-point crossover."* The following figure illustrates this concept. It originally appeared in Beasley et al [29].

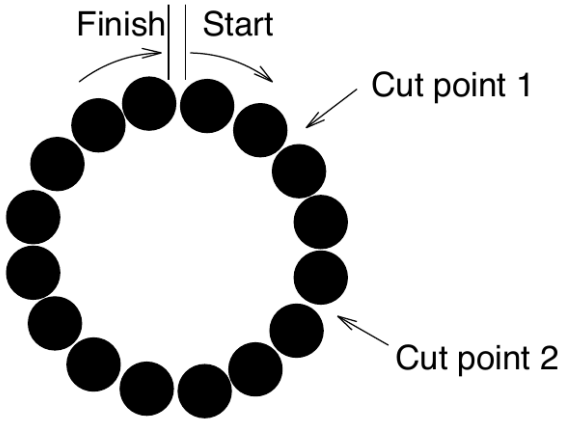


Figure 5. Chromosome Viewed As A Loop

All of the chromosomes in a single run of the genetic algorithm have the same length. To perform a two-point crossover, we first choose two cut points randomly, such that they are not the same. A cut point is either a position between bits or the position after the final bit, which is considered to be the same as the position before the first bit. The first offspring is the same as the first parent, except that the bits between the cut points are those of the second

parent. Likewise, the second offspring is the same as the second parent, except that the bits between the cut points are those of the first parent. One-point crossover is the same as two-point crossover with the exception that one of the points is fixed to be the boundary that joins the start and the finish of the chromosome. The system implementation only makes use of two-point crossover.

In addition to the new offspring, the chromosomes with the highest fitness score of this generation are passed on directly to the pool of the next generation. This is known as elitism.

3.5 Mutations

After the next generation has been created, each of the chromosomes has a 5% chance to mutate. According to the book “Practical Genetic Algorithms” by Randy L. Haupt and Sue Ellen Haupt, page 42: “Increasing the number of mutations increases the algorithm's freedom to search outside the current region of parameter space. It also tends to distract the algorithm from converging on a solution. Typically, on the order of 1% to 5% of the bits mutate per iteration […] We want to mutate 5% of the *population* (μ = 0.05) except for the best chromosome.” Given that the experimentation results shown on Chapter 5 are very good, it is understood that the choice of 5% for the mutation parameter is a reasonable one.

One of the elite chromosomes that were passed on from the previous generation is protected from mutation. The mutation process is very simple: one of the bits of the chromosome is selected at random, and its value is flipped to the opposite one.

3.6 The overall process

Because of the way the genetic algorithm has been implemented, the elite chromosomes get gradually better as generations go by. In case that an elite chromosome has a fitness score of 0.999 or more, out of a maximum of 1.0, then the algorithm stops immediately. This usually happens when the system is queried about a word it has already learned. Due to randomness, this may either happen in the first generation, or it may take several.

If 40 consecutive generations go by without improvement of the elite, then the algorithm stops. From experience, this could probably be reduced to about 20 or even less without adverse effects, but is kept at 40 to ensure good performance of the algorithm.

If a total of 150 generations go by, the algorithm stops immediately regardless of the state of the elite. From experience, this has never actually happened. The maximum amount of generations that have been seen is around 50, at a point in which the limit of 40 consecutive generations without improvement of the elite is reached and the algorithm stops.

3.7 An alternative to the genetic algorithm

If the genetic algorithm runs for only 40 generations, and every generation has about 400 chromosomes (a value which varies due to elitism), then the fitness function is evaluated about 16,000 times. However, if we generate all possible chromosome combinations for a word of 14 letters, then we would have 2^{13} = 8,192 chromosomes, since the length of every chromosome is the amount of letters in the word minus one. If we simply evaluate all those chromosomes one by one, and pick the best chromosome, we would finish faster than if we ran the genetic algorithm, assuming that the genetic algorithm runs for only 40 generations. For a word of 15 letters (such as *superestadistas*) we would have 2^{14} = 16,384 chromosomes. In that case the genetic algorithm would be faster assuming, again, that the algorithm runs for only 40 generations. However, words of at least 15 letters are uncommon, thus sequentially running through all possible combinations seems to be a better option than the use of the genetic algorithm. Nevertheless, the fitness function that was developed for the genetic algorithm remains a valuable contribution of this thesis, along with the word similarity measure algorithm discussed in Chapter 4.

However, for small words the genetic algorithm may finish in a few generations if it quickly finds a chromosome whose fitness value is at least 0.999. Due to the use of randomness in the genetic algorithm this is sometimes true, therefore, the use of all possible combinations is usually, but not always, a better alternative to the genetic algorithm.

4. THE WORD SIMILARITY MEASURE ALGORITHM

The word similarity measure algorithm begins with a process known as segment matching. A segment is a substring of a word, and a word in this context is a string of Spanish letters, with or without diacritics. An example is shown in Figure 6.

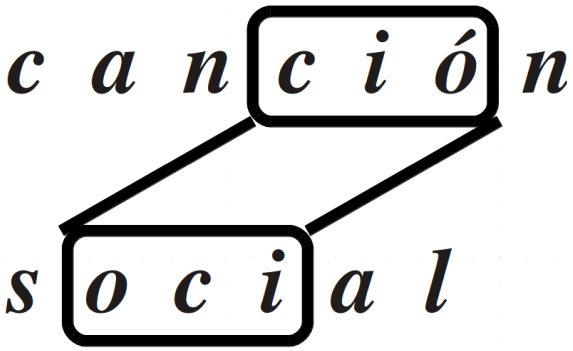


Figure 6. Segment Matching

There are three rules that govern the segment matching process. First, diacritics are ignored during the process. Thus, the *n* can match the *ñ*, the *u* can match *ú* and *ü*, and so on for the remaining vowels *a, e, i* and *o*. Second, the set of letters in one segment must be the same as that of the other segment (while ignoring diacritics, of course). Third, the order of the letters does not matter. Of course, segments must be continuous. Note that each segment in a pair has the same length, which can be as small as one letter or as big as the smallest word. When a pair entirely covers both words, it does not necessarily mean that the words are the same, since order is ignored during the matching process. The algorithm finds as many segment pairs as it is possible, with the exception that no segment pair can be wholly a subset

of another. To understand this, imagine a segment pair made up of segments A and B, and another segment pair made up of segments C and D. Then, if C is a subset of A it should not be the case that D is a subset of B. As an example, in Figure 6 there is no segment pair that can match *ci* in the first word with *ci* in the second word, since each is a subset of segments that already form a pair. To ensure this requirement, the algorithm begins by trying to find the biggest segment pair possible. Then, it progressively reduces the length of the matches attempted until individual letters are considered. While a segment pair cannot be wholly a subset of another, it is possible for a segment pair to partially share letters in common with another segment pair. The following figure demonstrates an example:

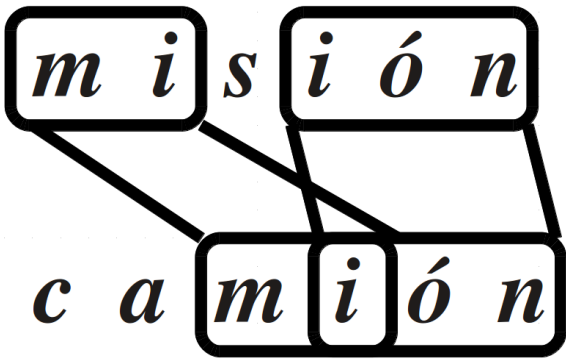


Figure 7. Partially Shared Matching

The system now proceeds to compute several scores for each segment pair before an overall *word similarity* score is computed. If the system finds no matching segment pairs, then the *word similarity* score will be 0%.

In order to obtain the *word similarity* score, the following scores are computed:

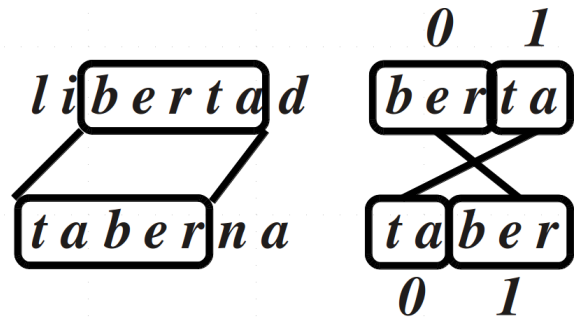
1. The *order* score.
2. The *letter similarity* score.
3. The *size* score.
4. The *segment pair* score.
5. The *segment pairs* score.

4.1 The *order* score

First, we compute a *disorder* score. An example is shown in Figure 8. The segment pair is further subdivided into subsegment pairs such that each pair is equivalent with diacritics ignored. In other words, order matters for each subsegment pair. For the example given, the segment pair *berta-taber* is divided into the subsegment pairs *ber-ber* and *ta-ta*. Then, for each segment, each of its subsegments is given an index number. In the figure these are the values 0 and 1. A subsegment score is given to each subsegment pair, which is the magnitude of the difference between indexes, with the total sum being the *disorder* score for the segment pair.

That *disorder* score is then divided by the maximum possible score given the amount of subsegment pairs. The result of that division is a value between 0 and 1, representing the *proportional disorder* score *x*. As a special case, if the amount of subsegment pairs is 1, then the result of the division is defined to be 0. This is because the maximum possible score for 1 subsegment pair is 0, and thus we would have to perform a division by 0.

A program was written to try all possibilities and report the maximum possible *disorder* score for each size. The program is listed in Appendix C. The results are shown in Table 1. Those results are in turn extrapolated and shown in Table 2.



These words share the segment pair shown.
There are two subsegment pairs.
The numbers are indexes.
The *disorder* score is: | 0 – 1 | + | 1 – 0 | = 2.

Figure 8. Subsegment Matching

Subsegment pairs	Maximum score
1	0
2	2
3	4
4	8
5	12
6	18
7	24
8	32
9	40
10	50
11	The program stops due to excessive memory usage

Table 1. Maximum *Disorder* Scores

Subsegment pairs	Maximum score
n = Even	$\frac{n^2}{2}$
n = Odd	$\frac{n^2-1}{2}$

Table 2. Extrapolated *Disorder* Scores

(Please note that applying n = 1 through 10 to Table 2 yields the values on Table 1.)

For the example given in Figure 8, we have that the *disorder* score is 2, and for the amount of subsegment pairs which is 2, the maximum score is 2. Dividing the *disorder* score 2 by the maximum score 2 yields a *proportional disorder* score of 1. We now proceed to evaluate $1 - x$ on the result in order to obtain an *order* score, with 0% resulting from the worst possible order and 100% resulting from perfect order. In the example given in Figure 8 we have that the *order* score is 1 – *proportional disorder* score = 1 – 1 = 0.

4.2 The *letter similarity* score

We match as many pairs of characters from one segment to the other (diacritics must match) as it is possible. For example, for the segment pair *amor* and *majó,* we have the pairs *a*-*a* and *m*-*m,* but not *o*-*ó.* The number of matching pairs *m* is divided by the segment length *s* in order to produce a value between 0 and 1, representing a percentage. Recall that both segments in a segment pair have the same segment length *s.* In the example for the segment pair *amor* and *majó, m* = 2 and *s* = 4. Thus the letter similarity score for this pair is 2 ÷ 4 = ½.

4.3 The *size* score

The total sum of characters of both segments is divided by the total sum of characters of both words. This again produces a value between 0 and 1, representing a percentage. Consider the word pair *amor-amores* and the segment pair *amor-amor.* Here, the size score would be 8 (from *amor-amor*) divided by 10 (from *amor-amores*), which yields 0.8.

4.4 The *segment pair* and *segment pairs* scores

The average of the order score, the letter similarity score, and the size score is taken in order to produce the *segment pair* score. Since each score lies between 0 and 1, the *segment pair* score will likewise lie between 0 and 1.

The *segment pairs* score for the word is the highest value among all of the *segment pair* scores. Consider the word pair *revista-revisita.* There are two segment pairs: *revis-revis* and *ista-sita.*

To calculate the *disorder* score for *revis-revis,* we divide the segment pair into subsegment pairs. However, in this case there is only one subsegment pair, which is *revis-revis* itself. Thus, both *revis* bear an index number of 0. The score for the subsegment pair *revis-revis* is $| 0 – 0 | = 0$. From Table 1, the maximum score for 1 subsegment pair is 0, thus to obtain the *proportional disorder* score we would divide $0 \div 0$, but this is not possible, so we make use of the special rule that says that we take the result of the division to be 0. This was described in Section 4.1.

We now compute one minus the *proportional disorder* score to obtain the *order* score. Thus, the order score is 1 in this case.

To obtain the letter similarity score, we note that there are 5 matches in *revis-revis,* and that the length of each segment is 5, thus the letter similarity score is 5 ÷ 5 = 1.

To obtain the size score, we note that the *segment pair revis-revis* has 10 characters in total, whereas the words being compared, *revista* and *revisita*, have 15 characters in total. Thus the size score is 10 ÷ 15 = 2/3.

To obtain the *segment pair* score for the segment pair *revis-revis* within the word pair *revista-revisita*, we take average of the order score, the letter similarity score, and the size score. Thus, we take the average of 1, 1, and 2/3, which is 88.88889%.

To calculate the *disorder* score for *ista-sita,* we divide the segment pair into subsegment pairs. These are *i-i, s-s* and *ta-ta.* For *i* the indexes are 0 within *ista* and 1 within *sita.* Thus the value of *i* is | 0 – 1 | = 1. For *s* the indexes are 1 within *ista* and 0 within *sita.* Thus the value of *s* is | 1 – 0 | = 1. For *ta* the indexes are 2 within *ista* and 2 within *sita.* Thus the value of *ta* is | 2 – 2 | = 0. The *disorder* score for *ista-sita* is 1 + 1 + 0 = 2.

From Table 1, the 3 subsegments pairs *i-i, s-s* and *ta-ta.* have a maximum *disorder* score of 4. Thus the *proportional disorder* score is 2 ÷ 4 = ½. To obtain the *order* score we compute one minus the *proportional disorder* score ½. Thus the *order* score is ½.

To obtain the letter similarity score, we note that there are 4 matches in *ista-sita,* and that the length of each segment is 4, thus the letter similarity score is 4 ÷ 4 = 1.

To obtain the size score, we note that the *segment pair ista-sita* has 8 characters in total, whereas the words being compared, *revista* and *revisita,* have 15 characters in total. Thus the size score is 8 ÷ 15 = 8/15.

To obtain the *segment pair* score for the segment pair *ista-sita* within the word pair *revista-revisita*, we take average of the order score, the letter similarity score, and the size score. Thus, we take the average of ½, 1, and 8/15, which is 67.77778%.

The *segment pairs* score is the highest of the two *segment pair* scores, 88.88889% and 67.77778%. Thus the *segment pairs* score for *revista-revisita* is 88.88889%.

4.5 The *word similarity* score

For each of the two words, a set of words that are similar in pronunciation is calculated. Listing 1 shows all of the pronunciation equivalences used. They were compiled as a result of an analysis of equivalences – no source was used. For example, given the word *casa* (= *house*), the following words are computed: *caza, kasa, kaza.* Of these, only *caza* (= *hunt*) is a spanish word, but we are not concerned with meaning.

Each of the two sets of words become the row and column headers of a crossover table. Then, for each cell in the table, the segment pairs score is computed as described in the previous section. Table 3 shows an example for the pair *casa* and *moza* (= *young lady*).

ca ↔ *ka*[1] *cc* ↔ *x*
ce ↔ *se* *ge* ↔ *je*
ci ↔ *si* *gi* ↔ *ji*
co ↔ *ko* *que* ↔ *ke* ↔ *qe*[2]
cu ↔ *ku* *qui* ↔ *ki* ↔ *qi*
b ↔ *v* *s* ↔ *z*
ll → *y*
y (followed by a vowel) → *ll*
y (at the end or followed by a consonant) → *i*
i (at the end and not preceded by *gu, qu*) → *y*
u or *ú* or *ü* (neither preceded by *g* nor *q*) → *w*
w → *u* (if it would not become preceded by *g* nor *q*)
h (not as part of *ch*) → nothing[3]

[1] When a diacritic is present, it is kept in the conversion.
For example, *ca* ↔ *ka* implies *cá* ↔ *ká.*
[2] While the syllables *qe* and *qi* are not Spanish syllables,
they are nonetheless used as equivalencies.
[3] This conversion is one way, we certainly do not insert *h* characters
arbitrarily in any word!

Listing 1. Pronunciation Equivalencies

	moza	*mosa*
casa	25%	50%
caza	50%	25%
kasa	25%	50%
kaza	50%	25%

Table 3. Sample Pronunciation Table

The final *word similarity* score is the average of the segment pairs score for the original pair of words (in Table 3, at the top left) and the highest score among all of the cells in the table. For the example of *casa* and *moza,* the final *word similarity* score is the average between 25% and 50%, which is 37.5%.

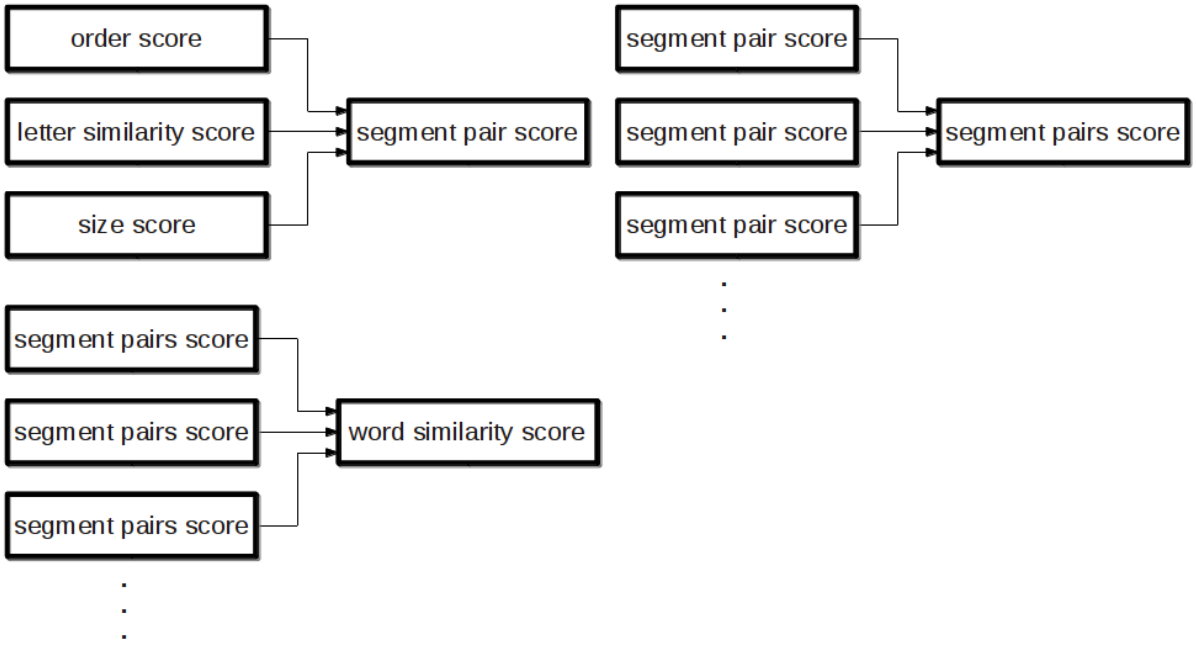


Figure 9. Scores Block Diagram

Why was this average formula chosen? The cell at the top left represents the comparison from a purely ortographical point of view. Meanwhile, all the other cells represent various scores from a pronunciation point of view, taking the highest value among them as the official score that is based on pronunciation. To allow both aspects to be taken into account, we average this value with the value computed from a purely ortographical point of view. This is the final *word similarity* score.

4.6 Testing the similarity algorithm

A program was written to receive as input an arbitrary long list of spanish words from the user, and to order them using the similarity algorithm in such a way that every pair of consecutive words is as similar as possible. Listings 2, 3 and 4 show three distinct sample

outputs. Occasionally, a pair of consecutive words could come out being quite dissimilar; this is usually the case, because there is no better choice of similarity out of the remaining words in the list, or simply because the similarity and ordering are suboptimal due to the complexity of the problem. In the case of the ordering algorithm, consider it to be similar to the Traveling Salesman Problem (TSP). We can compute a similarity score between every pair of words, must note that finding an ordering that maximizes the similarities between every pair of consecutive words is similar to finding an order that minimizes the distance between every pair of cities in a TSP. Thus, an approximation is used in the implementation. The way the ordering algorithm works: first pick the word that most resembles all of the rest. Successively pick the word that most resembles the previous one. Listing 2 shows a fairly good outcome of the algorithm; whereas the ordering in Listings 3 and 4 is not always obvious.

1. *pares*	6. *oso*	11. *visionario*	16. *llueve*
2. *par*	7. *vicio*	12. *cena*	17. *Jueves*
3. *parejo*	8. *servicio*	13. *cuece*	18. *mueves*
4. *parsimonia*	9. *visión*	14. *duele*	
5. *parsimonioso*	10. *televisión*	15. *telepatía*	

Listing 2. Similarity Test Run #1

1. *ría*	7. *gracias*	13. *subir*	19. *adoquín*
2. *avería*	8. *arco*	14. *urgir*	20. *Utuado*
3. *aerolito*	9. *dar*	15. *surgir*	21. *acentuado*
4. *arbolito*	10. *brindar*	16. *armonía*	22. *hacendoso*
5. *acólito*	11. *brincar*	17. *letanía*	23. *bonito*
6. *litografía*	12. *cubrir*	18. *arlequín*	24. *quingombó*

Listing 3. Similarity Test Run #2

1. *generoso*	9. *rosa*	17. *céntimo*	25. *onda*
2. *genes*	10. *jardinero*	18. *mover*	26. *monseñor*
3. *virgen*	11. *ceder*	19. *verificar*	27. *señorío*
4. *origen*	12. *creer*	20. *perilla*	28. *villorrio*
5. *genético*	13. *crear*	21. *periscopio*	29. *excesivo*
6. *timón*	14. *crecer*	22. *perlado*	30. *jazmín*
7. *jamón*	15. *acrecentar*	23. *lado*	
8. *roja*	16. *sentar*	24. *ondulado*	

Listing 4. Similarity Test Run #3

4.7 Discarded prototype scores

There were three types of score for which the *word similarity measure algorithm* was found to perform poorly when they were factored into it. The first was a *position* score for segment pairs, which would score higher as the relative positions were more similar (i.e., both segments toward the middle or towards the end of their respective words), and would score lower as the relative positions were less similar (i.e., one segment towards the beginning of its word, and the other toward the end of its word). The second was a *coverage* score that would score higher the more letters were part of segment pairs. The third was a

non-crossing score for pairs of segment pairs. If one were to draw lines between segment pairs, then it would score lower the more those lines crossed, scoring highest when they did not.

4.8 The cache system

A cache system was implemented into the word similarity measure algorithm that significantly speeds up the performance of the genetic algorithm. It consists of 100,000 entries, each one holding the word similarity measure score for a pair of words. It was implemented using the associative array facilities already present in the Java language.

4.9 Other languages

While the word similarity measure algorithm described in this thesis is tailored to the Spanish language, the language that was chosen to test the system, it can be modified so that the system can be made to work with other languages.

For example, French makes use of 5 diacritic marks. These are the *acute accent* (as in *é*), the *grave accent* (as in *è*), the *circumflex* (as in *ê*), the *diaeresis* (as in *ë*), and the *cedilla* (as in *ç*). It also makes use of 2 ligatures, “œ” and “æ”. In addition, French lacks the letter *ñ*.

Special rules need to be considered. For example, when identifying pairs of segments, under which circumstances should “œ” match “oe”? And, should “œ” be able to match “eo”?

Regardless of the word similarity measure algorithm in use, the genetic algorithm does not need to be modified. In particular, the fitness function can be used as is.

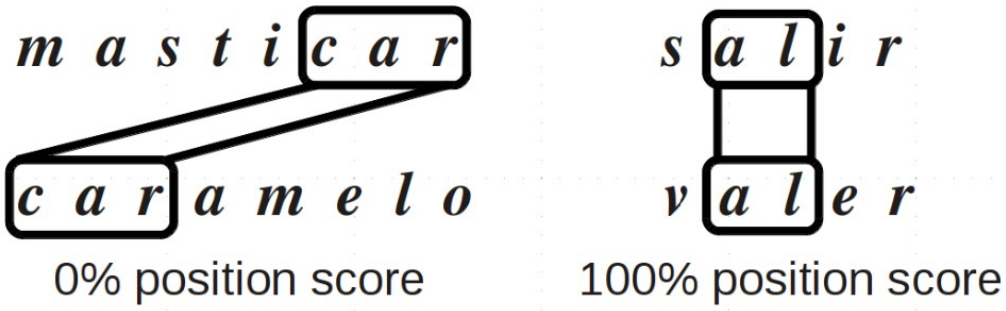


Figure 10. Position Score

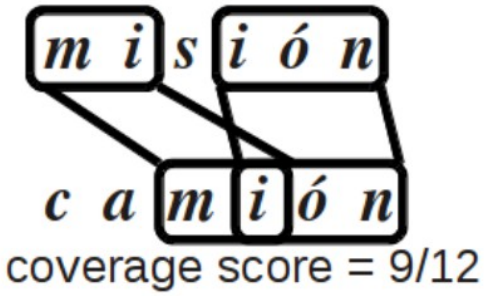


Figure 11. Coverage Score

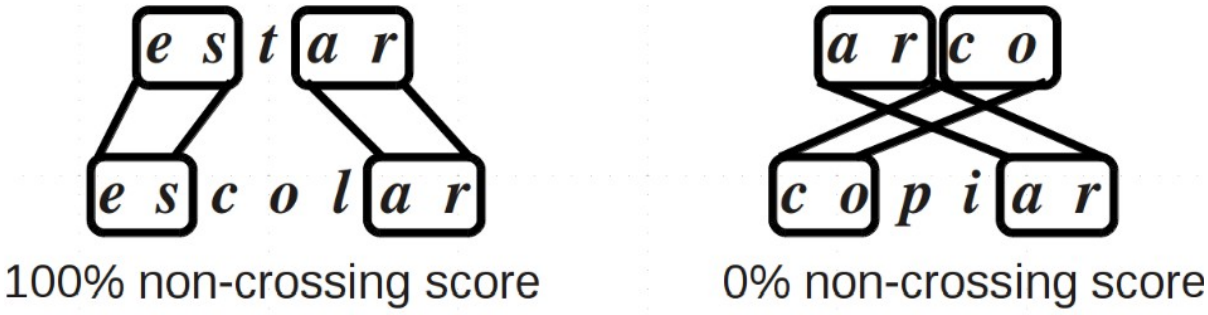


Figure 12. Non-Crossing Score

4.10 Comparison against sequence alignment

According to [27], *"Sequence alignment is the problem of finding the optimal character-by-character correspondence between two sequences."* The paper gives an example of performing sequence alignment on the pair of words *traceback* and *backtrack.* There are two possible alignments:

```
T R – A C E B – – – A C K
– – B A C – – K T R A C K

– – – – T R A C E B A C K
B A C K T R A C – – – – K
```

Sequence alignment is often used for the alignment of DNA and protein sequences in [28]. It is used in the alignment tasks where the order is important. The word similarity measure algorithm presented in this thesis does not work sequentially. Figures 6, 7 and 8 provide examples of the non-sequential nature of this algorithm. Therefore, the use of a sequence alignment algorithm is not an appropriate substitute to the word similarity algorithm.

5. EXPERIMENTATION

Here we present a sample run of the system from scratch (i.e., with no data previously learned). There are two modalities. Listing 5 shows the system running when it does *not* record position and frequency data. In contrast, Listing 6 shows the system running when it *does* record position and frequency. The run is similar to that of Listing 5, with the differences shown in bold. The important detail is that the modality that records position and frequency correctly divided one word that the other modality did not (*gatotes*), and failed at correctly dividing two words that the other modality did (*chica* and *chiquitiquitica*). In the next section we will see a more significant difference between the two modalities.

5.1 Corpus experimentation

A corpus of over 640 spanish words has been created, with each word being morphologically divided. It includes nouns, adjectives, verbs and adverbs, and is listed in Appendix B. An extensive Internet search was conducted in an attempt to find another corpus of spanish words that have already been morphologically divided, but no such corpus was found.

About 590 words from the corpus are loaded at the beginning of the system run, and we then proceed to test its ability to correctly perform morphological divisions on the remaining 50 words. The results are shown in Table 4, with a summary of results at the bottom. The results show that the modality of using position and frequency data is much

=gat-it-o-s
gatitos (Correctly yields *gat-it-o-s*)
gatitas (Incorrectly yields *gat-ita-s*)
=gat-it-a-s
gatitas (Correctly yields *gat-it-a-s*)
gatititas (Correctly yields *gat-it-it-a-s*)
gato (Correctly yields *gat-o*)
gata (Correctly yields *gat-a*)
gatos (Correctly yields *gat-o-s*)
gatas (Correctly yields *gat-a-s*)
gatotas (Incorrectly yields *gat-o-t-a-s*)
=gat-ot-a-s
gatotas (Correctly yields *gat-ot-a-s*)
gatitotas (Correctly yields *gat-it-ot-a-s*)
gatotes (Incorrectly yields *gat-ote-s*)
=gat-ot-es
gatotes (Correctly yields *gat-ot-es*)
gatototes (Correctly yields *gat-ot-ot-es*)
conejitas (Correctly yields *conej-it-a-s*)
conejotes (Correctly yields *conej-ot-es*)
perras (Correctly yields *perr-a-s*)
perrototes (Correctly yields *perr-ot-ot-es*)
chica (Correctly yields *chic-a*)
=ch-ic-a
chica (Correctly yields *ch-ic-a*)
chico (Correctly yields *ch-ic-o*)
chiquitico (Correctly yields *ch-iquit-ic-o*)
chiquitiquitica (Correctly yields *ch-iquit-iquit-ic-a*)
comer (Incorrectly yields *c-o-mer*)
=com-e-r
comer (Correctly yields *com-e-r*)
beber (Correctly yields *beb-e-r*)
remover (Incorrectly yields *r-e-mov-e-r*)
=re-mov-er
remover (Incorrectly yields *remov-e-r*)

Listing 5. Sample Run (Modality #1)

=gat-it-o-s
gatitos (Correctly yields *gat-it-o-s*)
gatitas **(Incorrectly yields *gat-it-as*)**
=gat-it-a-s
gatitas (Correctly yields *gat-it-a-s*)
gatititas (Correctly yields *gat-it-it-a-s*)
gato (Correctly yields *gat-o*)
gata (Correctly yields *gat-a*)
gatos (Correctly yields *gat-o-s*)
gatas (Correctly yields *gat-a-s*)
gatotas **(Incorrectly yields *gatot-a-s*)**
=gat-ot-a-s
gatotas (Correctly yields *gat-ot-a-s*)
gatitotas (Correctly yields *gat-it-ot-a-s*)
gatotes **(Correctly yields *gat-ot-es*)**
gatototes (Correctly yields *gat-ot-ot-es*)
conejitas (Correctly yields *conej-it-a-s*)
conejotes (Correctly yields *conej-ot-es*)
perras (Correctly yields *perr-a-s*)
perrototes (Correctly yields *perr-ot-ot-es*)
chica **(Incorrectly yields *c-hic-a*)**
=ch-ic-a
chica (Correctly yields *ch-ic-a*)
chico (Correctly yields *ch-ic-o*)
chiquitico (Correctly yields *ch-iquit-ic-o*)
chiquitiquitica **(Incorrectly yields *ch-iquitiquit-ic-a*)**
=ch-iquit-iquit-ic-a
chiquitiquitica (Correctly yields *ch-iquit-iquit-ic-a*)
comer (Incorrectly yields *c-o-mer*)
=com-e-r
comer (Correctly yields *com-e-r*)
beber (Correctly yields *beb-e-r*)
remover **(Incorrectly yields *remov-e-r*)**

Listing 6. Sample Run (Modality #2)

better than the other modality when a corpus is used. This states the benefit both of recording position and frequency data as well as using a corpus.

The fact that the system does not learn in some instances is significant. When no position or frequency data is recorded, the system could not learn 13 out of 50 words – that is 26% of the words given. Since this modality relies only on a list of morphemes learned, it means that the use of the corpus in this modality was counterproductive. Meanwhile, the recording of position and frequency data improved the performance of the system because there were no words it could not learn (versus 13 in the other modality).

5.2 Reinforcement learning

A total of 50 tests were run with the results shown in Table 4. In preparation for these tests, each corpus entry was taught to the system 35 times. For each test that did not yield the correct result immediately after running the query for that word, we taught the correct morphological division 35 times and ran the test again.

Must note that there are two types of reinforcement learning that occur in this system. When a query is made, genetic algorithms are used, which use a form of reinforcement learning. On the other hand, when the system is taught a morphological division, position and frequency data is updated according to what is newly taught. By teaching the same division many times, the system gives more weight (records an increased frequency) to the morpheme-position pairs of that word. This type of reinforcement learning cannot occur in the modality that does not record position nor frequency. In that modality, only morphemes are learned, therefore, teaching the same division more than once has no effect.

Word	Divided Word	Without Position Data	With Position Data
gatotes	*gat-ot-es*	Immediate result, ties with 4 others[1]	Immediate result, ties with 1 other
ultrafino	*ultra-fin-o*	Immediate result[2]	Immediate result
ultraconservador	*ultra-conserv-a-dor*	Did not learn	Immediate result
taxis	*taxi-s*	Requires teaching[3]	Immediate result
beben	*beb-e-n*	Immediate result, ties with 1 other	Immediate result
cosedora	*cos-e-dor-a*	Immediate result, ties with 14 others	Immediate result
repartirse	*repart-ir-se*	Did not learn	Immediate result
recibido	*recib-id-o*	Immediate result, ties with 1 other	Immediate result, ties with 1 other
recibís	*recib-í-s*	Immediate result	Immediate result
cuéntale	*cuént-a-le*	Requires teaching, ties with 2 others[4]	Requires teaching, ties with 1 other
silloncito	*sill-on-cit-o*	Immediate result, ties with 5 others	Immediate result
ininterrumpido	*in-interrump-id-o*	Requires teaching	Immediate result
reinscribir	*re-inscrib-ir*	Did not learn	Immediate result
inactivo	*in-act-iv-o*	Immediate result, ties with 10 others	Immediate result
sobrepuesta	*sobre-puest-a*	Immediate result	Immediate result
preacuerdo	*pre-acuerd-o*	Did not learn	Immediate result
exnovia	*ex-novi-a*	Immediate result, ties with 1 other	Immediate result
vicepresidente	*vice-presid-ente*	Did not learn	Immediate result, ties with 1 other
vicedecanato	*vice-decan-ato*	Did not learn	Immediate result
extracurricular	*extra-curricul-ar*	Did not learn	Immediate result
anticonstitucionales	*anti-constitu-cion-al-es*	Did not learn	Immediate result
observarán	*observ-a-rá-n*	Immediate result, ties with 1 other	Immediate result
extensionalidad	*extension-al-i-dad*	Did not learn	Immediate result, ties with 1 other
principado	*princip-ado*	Did not learn	Immediate result, ties with 1 other
tolerancia	*toler-ancia*	Requires teaching, ties with 3 others	Immediate result
bienaventuranza	*bien-aventur-anza*	Did not learn	Immediate result

[1] "Immediate result, ties with 4 others" means that before teaching, five solutions of equal fitness score tied up for the best, and one of them was the correct morphological division for that word. There is no need to teach.

[2] "Immediate result" means that the system was able to perform the correct morphological division for that word without having to teach it the correct morphological division.

[3] "Requires teaching" means that the system was not able to perform the correct morphological division for that word before having to teach it the correct morphological division. After teaching, the system does perform the correct morphological division.

[4] "Requires teaching, ties with 1 other" means that after teaching, two solutions of equal fitness score tied up for the best, and one of them was the correct morphological division for that word.

Word	Divided Word	Without Position Data	With Position Data
incertidumbre	*in-certi-dumbre*	Requires teaching, ties with 1 other	Immediate result
torcedura	*torc-edura*	Requires teaching, ties with 6 others	Immediate result
cabezota	*cabez-ot-a*	Requires teaching, ties with 1 other	Immediate result
panezote	*pan-ezot-e*	Requires teaching, ties with 3 others	Immediate result
villorrio	*vill-orri-o*	Requires teaching, ties with 5 others	Immediate result
colombiano	*colombi-an-o*	Did not learn	Immediate result
vizcaína	*vizca-ín-a*	Requires teaching, ties with 3 others	Immediate result
tepozteco	*tepoz-tec-o*	Requires teaching, ties with 2 others	Immediate result
comías	*com-í-a-s*	Immediate result, ties with 1 other	Immediate result
comiste	*com-iste*	Immediate result, ties with 4 others	Immediate result
cantaremos	*cant-a-re-mos*	Immediate result, ties with 4 others	Immediate result, ties with 1 other
viviré	*viv-i-ré*	Immediate result, ties with 8 others	Immediate result, ties with 1 other
vivirá	*viv-i-rá*	Immediate result, ties with 8 others	Immediate result
viviríamos	*viv-i-ría-mos*	Immediate result, ties with 16 others	Immediate result
vivirían	*viv-i-ría-n*	Immediate result, ties with 20 others	Immediate result
cantásemos	*cant-á-se-mos*	Immediate result, ties with 3 others	Immediate result
cantáremos	*cant-á-re-mos*	Immediate result, ties with 2 others	Immediate result
cantareis	*cant-a-re-is*	Immediate result, ties with 10 others	Immediate result
vivieren	*viv-ie-re-n*	Immediate result, ties with 21 others	Immediate result, ties with 1 other
librería	*libr-ería*	Immediate result, ties with 4 others	Immediate result, ties with 1 other
caserío	*cas-erío*	Requires teaching, ties with 6 others	Immediate result, ties with 1 other
jugaré	*jug-a-ré*	Requires teaching, ties with 2 others	Immediate result, ties with 1 other
superestadistas	*super-estad-ista-s*	Immediate result, ties with 2 others	Immediate result
escogería	*escog-e-ría*	Did not learn	Immediate result, ties with 1 other
Note: Due to randomness, results may vary slightly when retrying some of the tests that end in ties, particularly if there are many, which are a result of a set of morphemes not being properly trained (i.e., too little training).		Did not learn: 26% Immediate result (no ties): 6% Immediate result (with ties): 42% Requires teaching (no ties): 4% Requires teaching (with ties): 22%	Did not learn: 0% Immediate result (no ties): 74% Immediate result (with ties): 24% Requires teaching (no ties): 0% Requires teaching (with ties): 2%

Table 4. Results Of Corpus Experimentation

5.3 Benchmarking

An attempt was made to contrast the results obtained by this system with the results obtained by similar ones. Four papers about morphological division of spanish words were found: [13, 14, 15, 16]. None of these offer data that can be used to compare this system against those. In addition, three papers about morphological division using genetic algorithms were found [24, 25, 26]. Of these, [24] offers no comparable data, [25] only offers average frequencies of words obtained from a French lexicon (not useful for benchmarking), and [26] only performs divisions of each word into two parts (morphemes).

5.4 Results For The Alternative Formula

In Section 4.4, we saw that we calculate the *segment pair* scores by averaging the respective order score, the letter similarity score, and the size score. An alternative formula can be used, which is the product of all of these three scores. However, if one of these values is zero then the resulting *segment pair* score will be zero, eliminating the contribution of the other values. Therefore, we want to make it so that zero cannot be one of the possible values for any of those three scores.

First we handle the *order* score. We add 1 to the maximum possible *disorder* score that is shown in Table 2. The new formulas can be seen in Table 5. Since we are dividing a *disorder* score by a maximum possible *disorder* score in order to obtain the *proportional disorder* score *x*, this will cause 0% to be possible for *x* whereas 100% may not be. When we

then compute $1 - x$, we obtain as the *order* score a number that lies between 0% (exclusive) to 100% (inclusive), as desired.

Subsegment pairs	New maximum score
n = Even	$\frac{n^2}{2}+1 = \frac{n^2+2}{2}$
n = Odd	$\frac{n^2-1}{2}+1 = \frac{n^2+1}{2}$

Table 5. New Maximum *Disorder* Scores

Regarding the *letter similarity* score, we use the following formula:

$$\frac{m+1}{s+1}$$

where m is the number of matching pairs and s is the segments' length. This makes it impossible for zero to be a result of computing the *letter similarity* score.

Regarding the *size* score, the value will never be zero due to the words and the segment length always being at least one.

The tests that were ran in Table 4 were re-run with these changes. The results follow:

Word	Divided Word	Without Position Data	With Position Data
gatotes	*gat-ot-es*	Immediate result, ties with 4 others[5]	Immediate result, ties with 1 other
ultrafino	*ultra-fin-o*	Immediate result	Immediate result
ultraconservador	*ultra-conserv-a-dor*	Did not learn	Did not learn
taxis	*taxi-s*	Requires teaching	Immediate result
beben	*beb-e-n*	Immediate result, ties with 1 other	Immediate result
cosedora	*cos-e-dor-a*	Immediate result, ties with 13 others	Immediate result
repartirse	*repart-ir-se*	Did not learn	Immediate result

[5] For an explanation of these terms please refer to Table 4.

Word	Divided Word	Without Position Data	With Position Data
recibido	*recib-id-o*	Immediate result, ties with 1 other	Immediate result, ties with 1 other
recibís	*recib-í-s*	Immediate result	Immediate result
cuéntale	*cuént-a-le*	Did not learn	Immediate result, ties with 1 other
silloncito	*sill-on-cit-o*	Did not learn	Immediate result
ininterrumpido	*in-interrump-id-o*	Did not learn	Immediate result
reinscribir	*re-inscrib-ir*	Requires teaching	Immediate result
inactivo	*in-act-iv-o*	Immediate result, ties with 11 others	Immediate result
sobrepuesta	*sobre-puest-a*	Immediate result	Immediate result
preacuerdo	*pre-acuerd-o*	Requires teaching, ties with 3 others	Immediate result
exnovia	*ex-novi-a*	Immediate result, ties with 1 other	Immediate result
vicepresidente	*vice-presid-ente*	Did not learn	Immediate result, ties with 1 other
vicedecanato	*vice-decan-ato*	Did not learn	Immediate result
extracurricular	*extra-curricul-ar*	Did not learn	Immediate result
anticonstitucionales	*anti-constitu-cion-al-es*	Did not learn	Immediate result
observarán	*observ-a-rá-n*	Requires teaching, ties with 1 other	Immediate result
extensionalidad	*extension-al-i-dad*	Did not learn	Requires teaching
principado	*princip-ado*	Requires teaching, ties with 1 other	Immediate result, ties with 1 other
tolerancia	*toler-ancia*	Requires teaching, ties with 8 others	Immediate result
bienaventuranza	*bien-aventur-anza*	Did not learn	Immediate result
incertidumbre	*in-certi-dumbre*	Requires teaching	Immediate result
torcedura	*torc-edura*	Requires teaching, ties with 4 others	Immediate result
cabezota	*cabez-ot-a*	Requires teaching, ties with 1 other	Immediate result
panezote	*pan-ezot-e*	Requires teaching, ties with 3 others	Immediate result
villorrio	*vill-orri-o*	Requires teaching, ties with 7 others	Immediate result
colombiano	*colombi-an-o*	Requires teaching	Immediate result
vizcaína	*vizca-ín-a*	Requires teaching, ties with 3 others	Immediate result
tepozteco	*tepoz-tec-o*	Did not learn	Immediate result
comías	*com-í-a-s*	Immediate result, ties with 1 other	Immediate result
comiste	*com-iste*	Immediate result, ties with 4 others	Immediate result
cantaremos	*cant-a-re-mos*	Immediate result, ties with 4 others	Immediate result, ties with 1 other
viviré	*viv-i-ré*	Immediate result, ties with 8 others	Immediate result, ties with 1 other
vivirá	*viv-i-rá*	Immediate result, ties with 8 others	Immediate result

Word	Divided Word	Without Position Data	With Position Data
viviríamos	*viv-i-ría-mos*	Immediate result, ties with 15 others	Immediate result
vivirían	*viv-i-ría-n*	Immediate result, ties with 20 others	Immediate result
cantásemos	*cant-á-se-mos*	Did not learn	Immediate result
cantáremos	*cant-á-re-mos*	Requires teaching, ties with 3 others	Immediate result
cantareis	*cant-a-re-is*	Requires teaching, ties with 8 others	Immediate result
vivieren	*viv-ie-re-n*	Immediate result, ties with 21 others	Immediate result, ties with 1 other
librería	*libr-ería*	Immediate result, ties with 5 others	Immediate result, ties with 1 other
caserío	*cas-erío*	Requires teaching, ties with 6 others	Immediate result, ties with 1 other
jugaré	*jug-a-ré*	Requires teaching, ties with 2 others	Immediate result, ties with 1 other
superestadistas	*super-estad-ista-s*	Did not learn	Immediate result
escogería	*escog-e-ría*	Requires teaching, ties with 3 others	Immediate result
Note: Due to randomness, results may vary slightly when retrying some of the tests that end in ties, particularly if there are many, which are a result of a set of morphemes not being properly trained (i.e., too little training).		Did not learn: 28% Immediate result (no ties): 6% Immediate result (with ties): 30% Requires teaching (no ties): 8% Requires teaching (with ties): 28%	Did not learn: 2% Immediate result (no ties): 74% Immediate result (with ties): 22% Requires teaching (no ties): 2% Requires teaching (with ties): 0%

Table 6. Results Of Alternative Corpus Experimentation

Comparing tables 4 and 6:

Without Position Data					
			Average Formula	Product Formula	Improvement
1	Did not learn	less is better	26%	28%	-2%
2	Immediate result (no ties)	more is better	6%	6%	0%
3	Immediate result (with ties)	more is better	42%	30%	-12%
4	Requires teaching (no ties)	depends on context	4%	8%	
5	Requires teaching (with ties)	depends on context	22%	28%	
6	Sum of items 2 through 5	more is better	74%	72%	-2%
7	Sum of items 1, 4 and 5	less is better	52%	64%	-12%

Table 7. Summary Comparison (Without Position Data)

This shows that it is better to use the average formula, at least when working with the modality that does not record position nor frequency data.

With Position Data					
			Average Formula	Product Formula	Improvement
1	Did not learn	less is better	0%	2%	-2%
2	Immediate result (no ties)	more is better	74%	74%	0%
3	Immediate result (with ties)	more is better	24%	22%	-2%
4	Requires teaching (no ties)	depends on context	0%	2%	
5	Requires teaching (with ties)	depends on context	2%	0%	
6	Sum of items 2 through 5	more is better	100%	98%	-2%
7	Sum of items 1, 4 and 5	less is better	2%	4%	-2%

Table 8. Summary Comparison (With Position Data)

It can be seen that in the modality that does record position and frequency data, which is the most useful modality, the average formula is also better than the product formula.

5.5 Morpheme Learning

Tests were run while teaching morphemes to the system a different amount of times: 1, 15, 25, 35 and 50 times. The summary of results is shown in Table 9. They were run in the modality that records position and frequency data. It can be seen that teaching morphemes 35 times is the best choice among these options, although the results for teaching morphemes 15, 25, 35 and 50 times is very similar. By teaching the same division many times, it gives more weight to the morpheme-position pairs of that word. This influences the outcome of the fitness function during the execution of the genetic algorithm.

Note that teaching the morphological division only once yields poor results. The percentages of immediate results is significantly lower than the percentages obtained when teaching the morphemes 15, 25, 35 or 50 times. Furthermore, the execution was slower when teaching the morphemes only once, because there are many more “requires teaching” results, which require that the genetic algorithm runs twice: once before teaching the correct division to the system and again afterwards. Compare this to the system yielding immediate results, which requires that the genetic algorithm runs only once.

Average formula	Learn 1 time	Learn 15 times	Learn 25 times	Learn 35 times	Learn 50 times
Did not learn	0%	0%	2%	0%	0%
Immediate result (no ties)	30%	74%	74%	74%	78%
Immediate result (with ties)	14%	22%	24%	24%	16%
Requires teaching (no ties)	50%	4%	0%	0%	4%
Requires teaching (with ties)	6%	0%	0%	2%	2%
Product formula	Learn 1 time	Learn 15 times	Learn 25 times	Learn 35 times	Learn 50 times
Did not learn	10%	2%	6%	2%	0%
Immediate result (no ties)	30%	74%	72%	74%	74%
Immediate result (with ties)	12%	18%	22%	22%	16%
Requires teaching (no ties)	44%	2%	0%	2%	2%
Requires teaching (with ties)	4%	4%	0%	0%	8%

Table 9. Results For Various Amounts Of Morpheme Learning

5.6 Suppletive Words

Allomorphs are different forms of the same morpheme, where a form can vary in sound without varying in meaning. For example, in Spanish, the allomorphs of the plural morpheme have the three forms -s, -es and -ø.

casa (*house*)
casa-s (*houses*)

reloj (*clock*)
reloj-es (*clocks*)

análisis (*analysis* or *analyses*)

Other examples of allomorphs are:

extend-er (*to extend*)
extens-ión (*extension*)

imprim-ir (*to print*)
impres-ión (*printout*)

The above are examples of partial suppletion, where the alternate forms have some degree of similarity. Total suppletion occurs where the forms are entirely different, such as:

buen-o (*good*)
mejor (*better*)

hag-o (*[I] do*)
hic-e (*[I] did*)

The corpus includes the two pairs of suppletive words above described, plus the following two additional pairs:

v-oy (*[I] go*)
fu-i (*[I] went*)

mal-o (*bad*)
peor (*worse*)

As can be seen in Tables 4 and 6, the inclusion of these suppletive words in the corpus does not prevent the system from achieving near-perfect performance in the modality where position and frequency data are recorded.

6. Conclusions

Genetic algorithms coupled with corpora are a viable option to the problem of morphological analysis. Without the use of a corpus, the system performs reasonably well as long as the queried words are fairly similar to those that have been taught to the system. That is because without the use of a corpus the amount of words taught to the system is small due to the manual nature of the teaching process. On the other hand, the use of a corpus improves the ability of the system to correctly perform morphological divisions on a variety of different words. This is true even when there are suppletive words present in the corpus.

The system performs in a near-perfect fashion when there is recording of position and frequency data. On the other hand, it performs poorly when no position or frequency data is recorded.

When morphemes are taught to the system 35 times the results were better than when the morphemes are taught to the system 1, 15, 25 or 50 times. The results for teaching 15, 25, 35 and 50 times are similar. However, when teaching only once the system produced poor results and the execution of the tests was slower.

The word similarity measure algorithm, tailored to the Spanish language since it has a richer morphology than English, leads to a satisfactory performance of the genetic algorithm.

The default formula used to compute the *segment pair* score is the average formula. A different formula was tested for comparison: the product formula. The tests and comparisons demonstrate that the use of the average formula is better than the product formula.

Further research on this topic can include the use of chromosomes that carry more data than just possible morphological divisions, the use of an improved fitness function, the use of an improved selection process, the use of an enhanced word similarity measure algorithm, and/or the use of paralellism in a more powerful architecture.

~ * ~

APPENDIX A. JAVA CODE WITH ALGORITHMS

The following program implements the genetic algorithm and the word similarity measure algorithm. The main menu provides options for:

(1) Use the word similarity measure algorithm to perform a detailed comparison between two words,

(2) Receive a list of words from the user and order them such that every pair of consecutive words is as similar as possible,

(3) Run the main system either in the modality that records position and frequency data or in the one that does not, optionally making use of training words from a corpus, and allowing the user to teach the system new morphological divisions manually and to ask the system for the morphological division of any word given,

(4) Automatically test the 50 words in the modality that records position and frequency data and in the one that does not.

The program is 2,525 lines long.

```
package morphologicalAnalysisOfSpanishWordsUsingGeneticAlgorithms;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Random;
import java.util.Scanner;
import java.util.TreeMap;
```

```
import java.util.TreeSet;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;

public class MorphologicalAnalysisOfSpanishWordsUsingGeneticAlgorithms {
	private static enum FormulaKind {
		AVERAGE,
		PRODUCT
	}
	private static FormulaKind formulaKind = FormulaKind.AVERAGE;
	private static int LEARN_REPETITIONS = 35;
	private static Random randomNumberGenerator = new Random();
	private static interface ChromosomePair<ChromosomeType> {
		Chromosome<ChromosomeType> getFirstChromosome();
		Chromosome<ChromosomeType> getSecondChromosome();
	}
	private static interface Chromosome<ChromosomeType> {
		ChromosomePair<ChromosomeType> crossover(ChromosomeType mate);
		ChromosomeType getMutation();
		ChromosomeType getCopy();
		double getFitnessScore();
	}
	private static interface ChromosomePool<ChromosomeType> {
		int getSize();
		int getGenerationNumber();
		ArrayList<ChromosomeType> getChromosomes();
		double getFitnessScore();
		double getStandardDeviation();
		ChromosomePool<ChromosomeType> getNextGeneration(int maximumPoolSize);
		TreeSet<ChromosomeType> getWinners();
		ChromosomeType getRandomFromWinners();
	}
	private static interface GeneticAlgorithm<ResultType> {
		ResultType run();
	}
	private static class WordsMustMatchException extends RuntimeException {
		private static final long serialVersionUID = -2625331807823094688L;
		public WordsMustMatchException() {
			super("Chromosome words must match during crossover!");
		}
	}
	private static class ChromosomeLengthMismatchException extends RuntimeException {
		private static final long serialVersionUID = 6115325754890060239L;
		public ChromosomeLengthMismatchException() {
			super("Chromosome length does not match word length!");
		}
	}
	private static class PoolsMustBeOfAtLeastSizeTwoException extends RuntimeException {
		private static final long serialVersionUID = -5089017810768240136L;
		public PoolsMustBeOfAtLeastSizeTwoException() {
			super("Pools must be of at least size 2!");
		}
	}
	private static class InvalidGenerationNumberException extends RuntimeException {
		private static final long serialVersionUID = -6029748470529970745L;
		public InvalidGenerationNumberException() {
			super("Invalid generation number!");
		}
	}
	private static class MalformedMorphemyzedWordException extends RuntimeException {
		private static final long serialVersionUID = -967849264232856193L;
		public MalformedMorphemyzedWordException() {
			super("Malformed morphemyzed word!");
```

```
		}
	}
	private static class MorphemyzedWordPair implements ChromosomePair<MorphemyzedWord> {
		private MorphemyzedWord firstChromosome;
		private MorphemyzedWord secondChromosome;
		public MorphemyzedWordPair(MorphemyzedWord firstChromosome, MorphemyzedWord
secondChromosome) {
			this.firstChromosome = firstChromosome;
			this.secondChromosome = secondChromosome;
		}
		public MorphemyzedWord getFirstChromosome() {
			return firstChromosome;
		}
		public MorphemyzedWord getSecondChromosome() {
			return secondChromosome;
		}
	}
	private static class MorphemyzedWord implements Chromosome<MorphemyzedWord>,
Comparable<MorphemyzedWord> {
		private String word;
		private int wordLength;
		private int chromosomeLength;
		private boolean[] chromosome;
		private boolean computedFitnessScore;
		private double fitnessScore;
		private MorphemeDataGetter getter;
		public MorphemyzedWord(String word, MorphemeDataGetter getter) {
			if (getter == null) {
				throw new NullPointerException();
			}
			this.word = word;
			wordLength = word.length();
			chromosomeLength = wordLength - 1;
			if (chromosomeLength < 0) {
				chromosomeLength = 0;
			}
			chromosome = new boolean[chromosomeLength];
			for (int index = 0; index < chromosomeLength; index++) {
				chromosome[index] = randomNumberGenerator.nextInt(2) == 1;
			}
			computedFitnessScore = false;
			fitnessScore = 0;
			this.getter = getter;
		}
		public MorphemyzedWord(MorphemeDataGetter getter, String morphemyzedWord) {
			if (getter == null) {
				throw new NullPointerException();
			}
			if (morphemyzedWord == null) {
				throw new NullPointerException();
			}
			if (morphemyzedWord.isEmpty()) {
				throw new MalformedMorphemyzedWordException();
			}
			if (morphemyzedWord.charAt(0) == '-') {
				throw new MalformedMorphemyzedWordException();
			}
			int morphemyzedWordLength = morphemyzedWord.length();
			if (morphemyzedWord.charAt(morphemyzedWordLength - 1) == '-') {
				throw new MalformedMorphemyzedWordException();
			}
			word = "";
			for (int index = 0; index < morphemyzedWordLength; index++) {
```

```
                        char character = morphemyzedWord.charAt(index);
                        if (character != '-') {
                                word += character;
                        }
                }
                wordLength = word.length();
                chromosomeLength = wordLength - 1;
                chromosome = new boolean[chromosomeLength];
                int chromosomeIndex = 0;
                for (int index = 1; index < morphemyzedWordLength; index++) {
                        char character = morphemyzedWord.charAt(index);
                        if (character == '-') {
                                chromosome[chromosomeIndex++] = true;
                                index++;
                        } else {
                                chromosome[chromosomeIndex++] = false;
                        }
                }
                computedFitnessScore = false;
                fitnessScore = 0;
                this.getter = getter;
        }
        private MorphemyzedWord(String word, MorphemeDataGetter getter, boolean[]
chromosome, boolean computedFitnessScore, double fitnessScore) {
                if (getter == null) {
                        throw new NullPointerException();
                }
                this.word = word;
                wordLength = word.length();
                chromosomeLength = wordLength - 1;
                if (chromosomeLength < 0) {
                        chromosomeLength = 0;
                }
                if (chromosome.length != chromosomeLength) {
                        throw new ChromosomeLengthMismatchException();
                }
                this.chromosome = chromosome;
                this.computedFitnessScore = computedFitnessScore;
                this.fitnessScore = fitnessScore;
                this.getter = getter;
        }
        public boolean equals(Object obj) {
                if (obj == null) {
                        return false;
                }
                if (!(obj.getClass().equals(MorphemyzedWord.class))) {
                        return false;
                }
                MorphemyzedWord Obj = (MorphemyzedWord)obj;
                if (!(word.equals(Obj.word))) {
                        return false;
                }
                if (chromosomeLength != Obj.chromosomeLength) {
                        return false;
                }
                for (int i = 0; i < chromosomeLength; i++) {
                        if (chromosome[i] != Obj.chromosome[i]) {
                                return false;
                        }
                }
                return getter.equals(getter);
        }
        public int hashCode() {
```

```
        return word.hashCode() ^ chromosome.hashCode() ^ getter.hashCode();
}
public int compareTo(MorphemyzedWord o) {
        if (o == null) {
                throw new NullPointerException();
        }
        if (!(o.getClass().equals(MorphemyzedWord.class))) {
                throw new ClassCastException();
        }
        int result = word.compareTo(o.word);
        if (result < 0) {
                return -1;
        }
        if (result > 0) {
                return 1;
        }
        result = chromosomeLength - o.chromosomeLength;
        if (result != 0) {
                return result;
        }
        for (int i = 0; i < chromosomeLength; i++) {
                if ((!chromosome[i]) && o.chromosome[i]) {
                        return -1;
                }
                if (chromosome[i] && (!o.chromosome[i])) {
                        return 1;
                }
        }
        return getter.hashCode() - o.getter.hashCode();
}
/*public String getWord() {
        return word;
}
public int getWordLength() {
        return wordLength;
}
public MorphemeDataGetter getGetter() {
        return getter;
}
public int getChromosomeLength() {
        return chromosomeLength;
}
public boolean[] getChromosome() {
        return Arrays.copyOf(chromosome, chromosomeLength);
}*/
public String getDisplayString() {
        String result = "";
        int chromosomeLength = chromosome.length;
        ArrayList<String> morphemes = new ArrayList<String>();
        int startIndex = 0;
        for (int index = 0; index < chromosomeLength; index++) {
                if (chromosome[index]) {
                        morphemes.add(word.substring(startIndex, index + 1));
                        startIndex = index + 1;
                }
        }
        morphemes.add(word.substring(startIndex, wordLength));
        int morphemesSize = morphemes.size();
        for (int index = 0; index < morphemesSize; index++) {
                result += morphemes.get(index);
                if (index < morphemesSize - 1) {
                        result += "-";
                }
```

```
                        }
                        return result;
                }
                public ArrayList<String> getMorphemes() {
                        ArrayList<String> result = new ArrayList<String>();
                        String morpheme = String.valueOf(word.charAt(0));
                        for (int index = 0; index < chromosomeLength; index++) {
                                if (chromosome[index]) {
                                        result.add(morpheme);
                                        morpheme = String.valueOf(word.charAt(index + 1));
                                } else {
                                        morpheme += String.valueOf(word.charAt(index + 1));
                                }
                        }
                        result.add(morpheme);
                        return result;
                }
                public MorphemyzedWordPair crossover(MorphemyzedWord mate) {
                        if (!(mate.word.equals(word))) {
                                throw new WordsMustMatchException();
                        }
                        if (chromosomeLength < 2) {
                                return new MorphemyzedWordPair(new MorphemyzedWord(word,
getter), new MorphemyzedWord(word, getter));
                        }
                        int endPoint1 = randomNumberGenerator.nextInt(chromosomeLength);
                        int endPoint2 = randomNumberGenerator.nextInt(chromosomeLength - 1);
                        if (endPoint2 >= endPoint1) {
                                endPoint2++;
                        }
                        if (endPoint2 < endPoint1) {
                                int temp = endPoint1;
                                endPoint1 = endPoint2;
                                endPoint2 = temp;
                        }
                        boolean[] newChromosome1 = new boolean[chromosomeLength];
                        boolean[] newChromosome2 = new boolean[chromosomeLength];
                        for (int index = 0; index < chromosomeLength; index++) {
                                if (index < endPoint1 || index >= endPoint2) {
                                        newChromosome1[index] = chromosome[index];
                                        newChromosome2[index] = mate.chromosome[index];
                                } else {
                                        newChromosome1[index] = mate.chromosome[index];
                                        newChromosome2[index] = chromosome[index];
                                }
                        }
                        return new MorphemyzedWordPair(new MorphemyzedWord(word, getter,
newChromosome1, false, 0), new MorphemyzedWord(word, getter, newChromosome2, false, 0));
                }
                public MorphemyzedWord getMutation() {
                        MorphemyzedWord newChromosome = getCopy();
                        if (chromosomeLength > 0) {
                                int mutationIndex =
randomNumberGenerator.nextInt(chromosomeLength);
                                newChromosome.chromosome[mutationIndex] = !
chromosome[mutationIndex];
                                newChromosome.computedFitnessScore = false;
                        }
                        return newChromosome;
                }
                public MorphemyzedWord getCopy() {
                        return new MorphemyzedWord(word, getter, chromosome,
computedFitnessScore, fitnessScore);
```

```
		}
		public double getFitnessScore() {
			if (computedFitnessScore) {
				return fitnessScore;
			}
			ArrayList<String> morphemes = new ArrayList<String>();
			ArrayList<Float> morphemePositions = new ArrayList<Float>();
			int startIndex = 0;
			for (int index = 0; index < chromosomeLength; index++) {
				if (chromosome[index]) {
					String morpheme = word.substring(startIndex, index + 1);
					morphemes.add(morpheme);
					int morphemeLength = morpheme.length();
					morphemePositions.add(startIndex / (float)(wordLength -
morphemeLength));   //If denominator 0, highestScore will be 0
					startIndex = index + 1;
				}
			}
			String morpheme = word.substring(startIndex, wordLength);
			morphemes.add(morpheme);
			int morphemeLength = morpheme.length();
			morphemePositions.add(startIndex / (float)(wordLength -
morphemeLength));   //If denominator 0, highestScore will be 0
			//Got morphemes
			int morphemesSize = morphemes.size();
			fitnessScore = 0;
			for (int index = 0; index < morphemesSize; index++) {
				morpheme = morphemes.get(index);
				morphemeLength = morpheme.length();
				double highestScore = (double)0;
				for (String candidateMorpheme: getter.getCandidateMorphemes())
{
					double morphemeSimilarityScore;
					if (getter instanceof FullLearningMorphemeDataGetter) {
						float checkPosition =
morphemePositions.get(index);
						TreeMap<Float, Integer> positionData =
((FullLearningMorphemeDataGetter)getter).getPositionData(candidateMorpheme);
						morphemeSimilarityScore =
getMorphemeSimilarityScore(morpheme, checkPosition, candidateMorpheme, positionData);
					} else {
						morphemeSimilarityScore =
getWordSimilarityScore(morpheme, candidateMorpheme);
					}
					if (morphemeSimilarityScore > highestScore) {
						highestScore = morphemeSimilarityScore;
					}
				}
				fitnessScore += highestScore;
			}
			fitnessScore /= (double)morphemesSize;
			computedFitnessScore = true;
			return fitnessScore;
		}
	}
	private static class MorphemyzedWordPool implements ChromosomePool<MorphemyzedWord> {
		private String word;
		private int size;
		private MorphemyzedWord[] chromosomes;
		private int generationNumber;
		private boolean computedFitnessScore = false;
		private double fitnessScore = 0;
		private boolean computedStandardDeviation = false;
```

```
		private double standardDeviation = 0;
		private boolean computedWinners = false;
		private TreeSet<MorphemyzedWord> winners = null;
		private MorphemeDataGetter getter;
		public MorphemyzedWordPool(String word, MorphemeDataGetter getter, int size,
boolean automatic) {
			if (getter == null) {
				throw new NullPointerException();
			}
			if (size < 2) {
				throw new PoolsMustBeOfAtLeastSizeTwoException();
			}
			this.word = word;
			this.size = size;
			chromosomes = new MorphemyzedWord[size];
			for (int index = 0; index < size; index++) {
				chromosomes[index] = new MorphemyzedWord(word, getter);
			}
			generationNumber = 1;
			this.getter = getter;
			if (!automatic) {
				System.out.println("Generation #1, size = " + size);
			}
		}
		private MorphemyzedWordPool(String word, MorphemeDataGetter getter,
MorphemyzedWord[] chromosomes, int generationNumber, boolean computedFitnessScore, double
fitnessScore, boolean computedStandardDeviation, double standardDeviation, boolean
computedWinners, TreeSet<MorphemyzedWord> winners, boolean automatic) {
			if (getter == null) {
				throw new NullPointerException();
			}
			size = chromosomes.length;
			if (size < 2) {
				throw new PoolsMustBeOfAtLeastSizeTwoException();
			}
			if (generationNumber < 1) {
				throw new InvalidGenerationNumberException();
			}
			this.word = word;
			this.chromosomes = chromosomes;
			this.generationNumber = generationNumber;
			this.computedFitnessScore = computedFitnessScore;
			this.fitnessScore = fitnessScore;
			this.computedStandardDeviation = computedStandardDeviation;
			this.standardDeviation = standardDeviation;
			this.computedWinners = computedWinners;
			this.winners = winners;
			this.getter = getter;
			if (!automatic) {
				System.out.println("Generation #" + generationNumber + ", size
= " + size);
			}
		}
		/*public String getWord() {
			return word;
		}
		public int getWordLength() {
			return wordLength;
		}
		public MorphemeDataGetter getGetter() {
			return getter;
		}*/
		public int getSize() {
```

```
                        return size;
                }
                public int getGenerationNumber() {
                        return generationNumber;
                }
                public ArrayList<MorphemyzedWord> getChromosomes() {
                        return new ArrayList<MorphemyzedWord>(Arrays.asList(chromosomes));
                }
                /*public MorphemyzedWordPool getCopy() {
                        return new MorphemyzedWordPool(word, getter, chromosomes,
generationNumber, computedFitnessScore, fitnessScore, computedStandardDeviation,
standardDeviation, computedWinners, winners);
                }*/
                public double getFitnessScore() {
                        if (computedFitnessScore) {
                                return fitnessScore;
                        }
                        boolean firstComputation = true;
                        for (MorphemyzedWord chromosome: chromosomes) {
                                if (firstComputation) {
                                        fitnessScore = chromosome.getFitnessScore();
                                        firstComputation = false;
                                } else {
                                        double currentFitnessScore =
chromosome.getFitnessScore();
                                        if (currentFitnessScore > fitnessScore) {
                                                fitnessScore = currentFitnessScore;
                                        }
                                }
                        }
                        computedFitnessScore = true;
                        return fitnessScore;
                }
                public double getStandardDeviation() {
                        if (computedStandardDeviation) {
                                return standardDeviation;
                        }
                        double mean = 0;
                        for (MorphemyzedWord chromosome: chromosomes) {
                                mean += chromosome.getFitnessScore();
                        }
                        mean /= (float)size;
                        standardDeviation = 0;
                        for (MorphemyzedWord chromosome: chromosomes) {
                                double toBeSquared = chromosome.getFitnessScore() - mean;
                                standardDeviation += toBeSquared * toBeSquared;
                        }
                        standardDeviation = Math.sqrt(standardDeviation / (float)size);
                        computedStandardDeviation = true;
                        return standardDeviation;
                }
                public MorphemyzedWordPool getNextGeneration(int maximumPoolSize) {
                        return getNextGeneration(maximumPoolSize, false);
                }
                public MorphemyzedWordPool getNextGeneration(int maximumPoolSize, boolean
automatic) {
                        maximumPoolSize = (int)Math.floor(maximumPoolSize / (float)2) * 2;
                        if (maximumPoolSize < 2) {
                                throw new PoolsMustBeOfAtLeastSizeTwoException();
                        }
                        TreeMap<Double, TreeSet<MorphemyzedWord>> scoresMap = new
TreeMap<Double, TreeSet<MorphemyzedWord>>();
                        for (MorphemyzedWord chromosome: chromosomes) {
```

```
                                double chromosomeFitnessScore = chromosome.getFitnessScore();
                                if (!(scoresMap.containsKey(chromosomeFitnessScore))) {
                                        scoresMap.put(chromosomeFitnessScore, new
TreeSet<MorphemyzedWord>());
                                }
                                scoresMap.get(chromosomeFitnessScore).add(chromosome);
                        }
                        ArrayList<Double> scoresList = new
ArrayList<Double>(scoresMap.keySet());
                        Collections.sort(scoresList, Collections.reverseOrder());
                        int scoresListSize = scoresList.size();
                        double highestScore = scoresList.get(0);
                        double lowestScore = scoresList.get(scoresListSize - 1);
                        TreeSet<MorphemyzedWord> elite = scoresMap.get(highestScore);
                        int eliteSize = elite.size();
                        int totalSize = 0;
                        for (double score: scoresMap.keySet()) {
                                totalSize += scoresMap.get(score).size();
                        }
                        int halfSize = (int)Math.ceil(totalSize / (float)2);
                        int chromosomesFound = 0;
                        int highestPassingIndex = scoresListSize;
                        int newSize = 0;
                        for (int index = 0; index < scoresListSize; index++) {
                                double score = scoresList.get(index);
                                chromosomesFound += scoresMap.get(score).size();
                                if (chromosomesFound >= halfSize) {
                                        highestPassingIndex = index;
                                        newSize = (int)Math.floor((chromosomesFound + 1) /
(float)2) * 2 * 2 + eliteSize;
                                        if (newSize > maximumPoolSize) {
                                                newSize = maximumPoolSize;
                                                if ((newSize - eliteSize) % 2 == 1) {
                                                        newSize--;
                                                }
                                        }
                                        break;
                                }
                        }
                        if (newSize < size) {
                                newSize = size;
                                if ((newSize - eliteSize) % 2 == 1) {
                                        newSize++;
                                        if (newSize > maximumPoolSize) {
                                                newSize -= 2;
                                        }
                                }
                        }
                        while (newSize < 2) {
                                newSize += 2;
                        }
                        if (!automatic) {
                                System.out.println("New total size (after removal of
duplicates) = " + totalSize + ", halfSize = " + halfSize);
                        }
                        if (highestPassingIndex == 0) {
                                if (scoresListSize > 1) {
                                        highestPassingIndex = 1;
                                }
                        }
                        //Got highestScore, lowestScore, halfSize, highestPassingIndex,
eliteSize, newSize
                        if (!automatic) {
```

```
                                System.out.println("highestScore = " + highestScore + ",
lowestScore = " + lowestScore);
                        }
                        int elementsToRemove = 0;
                        for (int index = highestPassingIndex + 1; index < scoresListSize;
index++) {
                                double score = scoresList.get(index);
                                scoresMap.remove(score);
                                elementsToRemove++;
                        }
                        for (int index = 0; index < elementsToRemove; index++) {
                                scoresList.remove(scoresList.size() - 1);
                        }
                        scoresListSize -= elementsToRemove;
                        //Got the top half in scoresMap and scoresList. Got newSize (it has
space for the elites).
                        if (!automatic) {
                                System.out.println("Passing distinct scores = " +
scoresList.size() + ", elite = " + eliteSize + ", non-elite = " + (newSize - eliteSize) + ",
newSize = " + newSize + "\n");
                        }
                        TreeMap<Double, Double> probabilitiesMap = new TreeMap<Double,
Double>();
                        double runningTotal = 0;
                        for (double score: scoresList) {
                                runningTotal += (score - lowestScore) *
scoresMap.get(score).size();
                        }
                        if (runningTotal > 0) {
                                for (double score: scoresList) {
                                        probabilitiesMap.put(score, (score - lowestScore) *
scoresMap.get(score).size() / (float)runningTotal);
                                }
                        } else {
                                for (double score: scoresList) {
                                        runningTotal += scoresMap.get(score).size();
                                }
                                for (double score: scoresList) {
                                        probabilitiesMap.put(score, (double)
(scoresMap.get(score).size() / (float)runningTotal));
                                }
                        }
                        TreeMap<Double, Double> cumulativeProbabilitiesMap = new
TreeMap<Double, Double>();
                        double cumulativeProbability = 0;
                        for (int index = 0; index < scoresListSize; index++) {
                                double score = scoresList.get(index);
                                if (index < scoresListSize - 1) {
                                        cumulativeProbability += probabilitiesMap.get(score);
                                } else {
                                        cumulativeProbability = 1;
                                }
                                cumulativeProbabilitiesMap.put(cumulativeProbability, score);
                        }
                        ArrayList<Double> probabilitiesList = new
ArrayList<Double>(cumulativeProbabilitiesMap.keySet());
                        Collections.sort(probabilitiesList);
                        int probabilitiesListSize = probabilitiesList.size();
                        //Got probabilitiesList, cumulativeProbabilitiesMap
                        MorphemyzedWord[] newChromosomes = new MorphemyzedWord[newSize];
                        TreeSet<MorphemyzedWord> eliteList = scoresMap.get(highestScore);
                        int index = 0;
                        for (MorphemyzedWord word: eliteList) {
```

```
                                if (index == 0) {
                                        newChromosomes[0] = word;
                                } else {
                                        if (randomNumberGenerator.nextDouble() < 0.05) {
                                                newChromosomes[index] = word.getMutation();
                                        } else {
                                                newChromosomes[index] = word;
                                        }
                                }
                                index++;
                        }
                        MorphemyzedWord[] parents = new MorphemyzedWord[2];
                        for (int outerIndex = eliteSize; outerIndex < newSize; outerIndex += 2)
{       //Guaranteed to be even
                                for (int middleIndex = 0; middleIndex < 2; middleIndex++) {
                                        double randomValue = randomNumberGenerator.nextDouble();
                                        double previousCheckProbability = 0;
                                        for (int innerIndex = 0; innerIndex <
probabilitiesListSize; innerIndex++) {
                                                double checkProbability =
probabilitiesList.get(innerIndex);
                                                if (randomValue < checkProbability) {
                                                        double checkScore =
cumulativeProbabilitiesMap.get(checkProbability);
                                                        ArrayList<MorphemyzedWord> checkList =
new ArrayList<MorphemyzedWord>(scoresMap.get(checkScore));
                                                        double intervalSize = (checkProbability -
previousCheckProbability) / (float)checkList.size();
                                                        randomValue -= previousCheckProbability;
                                                        int checkListIndex =
(int)Math.floor(randomValue / intervalSize);
                                                        MorphemyzedWord parentChromosome =
checkList.get(checkListIndex);
                                                        parents[middleIndex] = parentChromosome;
                                                        break;
                                                }
                                                previousCheckProbability = checkProbability;
                                        }
                                }
                                MorphemyzedWordPair pair = parents[0].crossover(parents[1]);
                                if (randomNumberGenerator.nextDouble() < 0.05) {
                                        newChromosomes[outerIndex] =
pair.getFirstChromosome().getMutation();
                                } else {
                                        newChromosomes[outerIndex] = pair.getFirstChromosome();
                                }
                                if (randomNumberGenerator.nextDouble() < 0.05) {
                                        newChromosomes[outerIndex + 1] =
pair.getSecondChromosome().getMutation();
                                } else {
                                        newChromosomes[outerIndex + 1] =
pair.getSecondChromosome();
                                }
                        }
                        return new MorphemyzedWordPool(word, getter, newChromosomes,
generationNumber + 1, false, 0, false, 0, false, null, automatic);
                }
                public TreeSet<MorphemyzedWord> getWinners() {
                        if (computedWinners) {
                                return winners;
                        }
                        double eliteScore = getFitnessScore();
                        winners = new TreeSet<MorphemyzedWord>();
```

```
                        for (MorphemyzedWord chromosome: chromosomes) {
                                if (chromosome.getFitnessScore() == eliteScore) {
                                        winners.add(chromosome);
                                }
                        }
                        computedWinners = true;
                        return winners;
                }
                public MorphemyzedWord getRandomFromWinners() {
                        ArrayList<MorphemyzedWord> winners = new
ArrayList<MorphemyzedWord>(getWinners());
                        return winners.get(randomNumberGenerator.nextInt(winners.size()));
                }
        }
        private static class WordMorphemyzerGeneticAlgorithm implements
GeneticAlgorithm<TreeSet<MorphemyzedWord>> {
                private String word;
                private MorphemeDataGetter getter;
                public WordMorphemyzerGeneticAlgorithm(String word, MorphemeDataGetter getter)
{
                        if (getter == null) {
                                throw new NullPointerException();
                        }
                        this.word = word;
                        this.getter = getter;
                }
                /*public String getWord() {
                        return word;
                }
                public MorphemeDataGetter getGetter() {
                        return getter;
                }*/
                public TreeSet<MorphemyzedWord> run() {
                        return run(false);
                }
                public TreeSet<MorphemyzedWord> run(boolean automatic) {
                        final int INITIAL_POOL_SIZE = 400;   //Old=200
                        final int MAXIMUM_POOL_SIZE = 1000;
                        final int MAXIMUM_AMOUNT_OF_GENERATIONS = 150;
                        final int MAXIMUM_AMOUNT_OF_GENERATIONS_WITHOUT_IMPROVEMENT = 40;
//Old=20
                        final double MINIMUM_SCORE_TO_FINISH_IMMEDIATELY = 0.999;   //Old=0.995
                        MorphemyzedWordPool pool = new MorphemyzedWordPool(word, getter,
INITIAL_POOL_SIZE, automatic);
                        boolean firstPool = true;
                        int consecutiveWithoutImprovement = 1;
                        double lastFitnessScore = 0;
                        boolean maxGenMessage = true;
                        do {
                                if (pool.getFitnessScore() >=
MINIMUM_SCORE_TO_FINISH_IMMEDIATELY) {
                                        if (!automatic) {
                                                System.out.println("Fitness score = " +
pool.getFitnessScore() + ", MINIMUM_SCORE_TO_FINISH_IMMEDIATELY = " +
MINIMUM_SCORE_TO_FINISH_IMMEDIATELY);
                                        }
                                        return pool.getWinners();
                                }
                                if (firstPool) {
                                        lastFitnessScore = pool.getFitnessScore();
                                        consecutiveWithoutImprovement = 1;
                                        firstPool = false;
                                } else {
```

```
                                        if (pool.getFitnessScore() > lastFitnessScore + 0.00001)
{
                                                lastFitnessScore = pool.getFitnessScore();
                                                consecutiveWithoutImprovement = 1;
                                        } else {
                                                lastFitnessScore = pool.getFitnessScore();
                                                consecutiveWithoutImprovement++;
                                                if (consecutiveWithoutImprovement >=
MAXIMUM_AMOUNT_OF_GENERATIONS_WITHOUT_IMPROVEMENT) {
                                                        maxGenMessage = false;
                                                        if (!automatic) {
                                                                System.out.println("Finished due
to MAXIMUM_AMOUNT_OF_GENERATIONS_WITHOUT_IMPROVEMENT = " +
MAXIMUM_AMOUNT_OF_GENERATIONS_WITHOUT_IMPROVEMENT);
                                                        }
                                                        break;
                                                }
                                        }
                                }
                                pool = pool.getNextGeneration(MAXIMUM_POOL_SIZE, automatic);
                        } while (pool.getGenerationNumber() < MAXIMUM_AMOUNT_OF_GENERATIONS);
                        if (maxGenMessage) {
                                if (!automatic) {
                                        System.out.println("GenerationNumber = " +
pool.getGenerationNumber() + ", MAXIMUM_AMOUNT_OF_GENERATIONS = " +
MAXIMUM_AMOUNT_OF_GENERATIONS);
                                }
                        }
                        return pool.getWinners();
                }
        }
        private static abstract class MorphemeDataGetter {
                public abstract ArrayList<String> getCandidateMorphemes();
        }
        /*private static class TesterMorphemeDataGetter extends MorphemeDataGetter {
                public ArrayList<String> getCandidateMorphemes() {
                        return new ArrayList<String>(Arrays.asList("super", "ultra", "gat",
"it", "ot", "o", "a", "es", "s", "ch", "iquit", "ic"));
                }
        }*/
        private static class PlainLearningMorphemeDataGetter extends MorphemeDataGetter {
                private TreeSet<String> morphemes = new TreeSet<String>();
                public ArrayList<String> getCandidateMorphemes() {
                        return new ArrayList<String>(morphemes);
                }
                public void addCandidateMorpheme(String morpheme) {
                        morphemes.add(morpheme);
                }
                public boolean knowsCandidateMorpheme(String morpheme) {
                        return morphemes.contains(morpheme);
                }
        }
        private static class FullLearningMorphemeDataGetter extends MorphemeDataGetter {
                private TreeMap<String, TreeMap<Float, Integer>> morphemes = new
TreeMap<String, TreeMap<Float, Integer>>();
                public ArrayList<String> getCandidateMorphemes() {
                        return new ArrayList<String>(morphemes.keySet());
                }
                public void addCandidateMorphemeData(String morpheme, float position) {
                        TreeMap<Float, Integer> positionData;
                        if (morphemes.containsKey(morpheme)) {
                                positionData = morphemes.get(morpheme);
                        } else {
```

```
                        positionData = new TreeMap<Float, Integer>();
                        morphemes.put(morpheme, positionData);
                }
                int newValue;
                if (positionData.containsKey(position)) {
                        newValue = positionData.get(position) + 1;
                } else {
                        newValue = 1;
                }
                positionData.put(position, newValue);
        }
        public boolean knowsCandidateMorpheme(String morpheme) {
                return morphemes.containsKey(morpheme);
        }
        public TreeMap<Float, Integer> getPositionData(String morpheme) {
                if (morphemes.containsKey(morpheme)) {
                        return morphemes.get(morpheme);
                } else {
                        TreeMap<Float, Integer> positionData = new TreeMap<Float,
Integer>();
                        morphemes.put(morpheme, positionData);
                        return positionData;
                }
        }
    }
    private static boolean isVowel(char character) {
        char lowercaseCharacter = Character.toLowerCase(character);
        return (lowercaseCharacter == 'a' || lowercaseCharacter == 'á' ||
                        lowercaseCharacter == 'e' || lowercaseCharacter == 'é' ||
                        lowercaseCharacter == 'i' || lowercaseCharacter == 'í' ||
                        lowercaseCharacter == 'o' || lowercaseCharacter == 'ó' ||
                        lowercaseCharacter == 'u' || lowercaseCharacter == 'ú' ||
                        lowercaseCharacter == 'ü');
    }
    private static boolean isConsonant(char character) {
        char lowercaseCharacter  = Character.toLowerCase(character);
        if ((lowercaseCharacter >= 'a' && lowercaseCharacter <= 'z') ||
lowercaseCharacter == 'ñ') {
                return (!(isVowel(lowercaseCharacter)));
        }
        return false;
    }
    private static boolean isSpanishLetter(char character) {
        return isVowel(character) || isConsonant(character);
    }
    private static boolean isSpanishLetterOrDash(char character) {
        return character == '-' || isSpanishLetter(character);
    }
    private static boolean areCharactersSimilar(char character1, char character2) {
        char lowercaseCharacter1 = Character.toLowerCase(character1);
        char lowercaseCharacter2 = Character.toLowerCase(character2);
        if (lowercaseCharacter1 == lowercaseCharacter2) {
                return true;
        }
        return ((lowercaseCharacter1 == 'a' && lowercaseCharacter2 == 'á')
                        || (lowercaseCharacter1 == 'á' && lowercaseCharacter2 == 'a')
                        || (lowercaseCharacter1 == 'e' && lowercaseCharacter2 == 'é')
                        || (lowercaseCharacter1 == 'é' && lowercaseCharacter2 == 'e')
                        || (lowercaseCharacter1 == 'i' && lowercaseCharacter2 == 'í')
                        || (lowercaseCharacter1 == 'í' && lowercaseCharacter2 == 'i')
                        || (lowercaseCharacter1 == 'o' && lowercaseCharacter2 == 'ó')
                        || (lowercaseCharacter1 == 'ó' && lowercaseCharacter2 == 'o')
                        || (lowercaseCharacter1 == 'u' && lowercaseCharacter2 == 'ú')
```

```
                        || (lowercaseCharacter1 == 'ú' && lowercaseCharacter2 == 'u')
                        || (lowercaseCharacter1 == 'u' && lowercaseCharacter2 == 'ü')
                        || (lowercaseCharacter1 == 'ü' && lowercaseCharacter2 == 'u')
                        || (lowercaseCharacter1 == 'ú' && lowercaseCharacter2 == 'ü')
                        || (lowercaseCharacter1 == 'ü' && lowercaseCharacter2 == 'ú')
                        || (lowercaseCharacter1 == 'n' && lowercaseCharacter2 == 'ñ')
                        || (lowercaseCharacter1 == 'ñ' && lowercaseCharacter2 == 'n'));
        }
        private static boolean areSegmentsSimilar(String segment1, String segment2) {
                int segmentLength = segment1.length();
                if (segment2.length() != segmentLength) {
                        return false;
                }
                if (segmentLength == 0) {
                        return true;
                }
                boolean[] used = new boolean[segmentLength];
                for (int index = 0; index < segmentLength; index++) {
                        used[index] = false;
                }
                int usedCount = 0;
                for (int index1 = 0; index1 < segmentLength; index1++) {
                        char character1 = segment1.charAt(index1);
                        for (int index2 = 0; index2 < segmentLength; index2++) {
                                if (!used[index2]) {
                                        char character2 = segment2.charAt(index2);
                                        if (areCharactersSimilar(character1, character2)) {
                                                used[index2] = true;
                                                usedCount++;
                                                break;
                                        }
                                }
                        }
                }
                return usedCount == segmentLength;
        }
        private static class SegmentPair {
                private int word1SegmentIndex;
                private int word2SegmentIndex;
                private int wordSegmentLength;
                public SegmentPair(int word1SegmentIndex, int word2SegmentIndex, int
wordSegmentLength) {
                        this.word1SegmentIndex = word1SegmentIndex;
                        this.word2SegmentIndex = word2SegmentIndex;
                        this.wordSegmentLength = wordSegmentLength;
                }
                public int getWord1SegmentIndex() {
                        return word1SegmentIndex;
                }
                public int getWord2SegmentIndex() {
                        return word2SegmentIndex;
                }
                public int getWordSegmentLength() {
                        return wordSegmentLength;
                }
        }
        private static ArrayList<SegmentPair> getSegmentPairs(String word1, String word2) {
                ArrayList<SegmentPair> segmentPairs = new ArrayList<SegmentPair>();
                String largerWord;
                int largerWordLength;
                String shorterWord;
                int shorterWordLength;
                int word1Length = word1.length();
```

```
                int word2Length = word2.length();
                boolean word1Greater = word1Length >= word2Length;
                if (word1Greater) {
                        largerWord = word1;
                        largerWordLength = word1Length;
                        shorterWord = word2;
                        shorterWordLength = word2Length;
                } else {
                        largerWord = word2;
                        largerWordLength = word2Length;
                        shorterWord = word1;
                        shorterWordLength = word1Length;
                }
                for (int segmentLength = shorterWordLength; segmentLength >= 1;
segmentLength--) {
                        int shorterLimit = shorterWordLength - segmentLength;
                        for (int shorterStartIndex = 0; shorterStartIndex <= shorterLimit;
shorterStartIndex++) {
                                int largerLimit = largerWordLength - segmentLength;
                                for (int largerStartIndex = 0; largerStartIndex <= largerLimit;
largerStartIndex++) {
                                        int shorterEndIndex = shorterStartIndex + segmentLength;
                                        int largerEndIndex = largerStartIndex + segmentLength;
                                        int word1SegmentIndex;
                                        int word2SegmentIndex;
                                        if (word1Greater) {
                                                word1SegmentIndex = largerStartIndex;
                                                word2SegmentIndex = shorterStartIndex;
                                        } else {
                                                word1SegmentIndex = shorterStartIndex;
                                                word2SegmentIndex = largerStartIndex;
                                        }
                                        boolean potentiallyAddPair = true;
                                        for (SegmentPair pair: segmentPairs) {
                                                int pairWord1SegmentIndex =
pair.getWord1SegmentIndex();
                                                int pairWord2SegmentIndex =
pair.getWord2SegmentIndex();
                                                int pairWordSegmentLength =
pair.getWordSegmentLength();
                                                if (word1SegmentIndex >= pairWord1SegmentIndex)
{
                                                        if (word1SegmentIndex + segmentLength <=
pairWord1SegmentIndex + pairWordSegmentLength) {
                                                                if (word2SegmentIndex >=
pairWord2SegmentIndex) {
                                                                        if (word2SegmentIndex +
segmentLength <= pairWord2SegmentIndex + pairWordSegmentLength) {
                                                                                potentiallyAddPair
= false;
                                                                                break;
                                                                        }
                                                                } else {
                                                                        if (pairWord2SegmentIndex
+ pairWordSegmentLength <= word2SegmentIndex + segmentLength) {
                                                                                potentiallyAddPair
= false;
                                                                                break;
                                                                        }
                                                                }
                                                        }
                                                } else {
                                                        if (pairWord1SegmentIndex +
```

```
pairWordSegmentLength <= word1SegmentIndex + segmentLength) {
                                                                        if (word2SegmentIndex >=
pairWord2SegmentIndex) {
                                                                                if (word2SegmentIndex +
segmentLength <= pairWord2SegmentIndex + pairWordSegmentLength) {
                                                                                        potentiallyAddPair
= false;
                                                                                        break;
                                                                                }
                                                                        } else {
                                                                                if (pairWord2SegmentIndex
+ pairWordSegmentLength <= word2SegmentIndex + segmentLength) {
                                                                                        potentiallyAddPair
= false;
                                                                                        break;
                                                                                }
                                                                        }
                                                                }
                                                        }
                                        }
                                        if (potentiallyAddPair) {
                                                String segment1 =
shorterWord.substring(shorterStartIndex, shorterEndIndex);
                                                String segment2 =
largerWord.substring(largerStartIndex, largerEndIndex);
                                                if (areSegmentsSimilar(segment1, segment2)) {
                                                        segmentPairs.add(new
SegmentPair(word1SegmentIndex, word2SegmentIndex, segmentLength));
                                                }
                                        }
                                }
                        }
                }
                return segmentPairs;
        }
        /*private static float getPositionScore(String word1, String word2, SegmentPair pair)
{
                int word1SegmentIndex = pair.getWord1SegmentIndex();
                int word2SegmentIndex = pair.getWord2SegmentIndex();
                int wordSegmentLength = pair.getWordSegmentLength();
                int word1DistanceFromLeft = word1SegmentIndex;
                int word1DistanceFromRight = word1.length() - word1SegmentIndex -
wordSegmentLength;
                int word2DistanceFromLeft = word2SegmentIndex;
                int word2DistanceFromRight = word2.length() - word2SegmentIndex -
wordSegmentLength;
                return 1 - (Math.abs(word1DistanceFromLeft * word2DistanceFromRight -
word1DistanceFromRight * word2DistanceFromLeft) / (float)(((word1DistanceFromLeft +
word1DistanceFromRight) * (word2DistanceFromLeft + word2DistanceFromRight)) + 1));
        }*/
        private static float getOrderScore(String word1, String word2, SegmentPair pair) {
                int word1SegmentIndex = pair.getWord1SegmentIndex();
                int word2SegmentIndex = pair.getWord2SegmentIndex();
                int wordSegmentLength = pair.getWordSegmentLength();
                String segment1 = word1.substring(word1SegmentIndex, word1SegmentIndex +
wordSegmentLength);
                String segment2 = word2.substring(word2SegmentIndex, word2SegmentIndex +
wordSegmentLength);
                boolean[] used1 = new boolean[wordSegmentLength];
                boolean[] used2 = new boolean[wordSegmentLength];
                for (int index = 0; index < wordSegmentLength; index++) {
                        used1[index] = false;
                        used2[index] = false;
```

```
		}
		TreeMap<Integer, Integer> subsegmentMatchMap1 = new TreeMap<Integer,
Integer>();
		for (int subsegmentLength = wordSegmentLength; subsegmentLength >= 1;
subsegmentLength--) {
			for (int index1 = 0; index1 <= wordSegmentLength - subsegmentLength;
index1++) {
				boolean unused1 = true;
				for (int index = index1; index < index1 + subsegmentLength;
index++) {
					if (used1[index]) {
						unused1 = false;
						break;
					}
				}
				if (unused1) {
					String subsegment1 = segment1.substring(index1, index1 +
subsegmentLength);
					for (int index2 = 0; index2 <= wordSegmentLength -
subsegmentLength; index2++) {
						boolean unused2 = true;
						for (int index = index2; index < index2 +
subsegmentLength; index++) {
							if (used2[index]) {
								unused2 = false;
								break;
							}
						}
						if (unused2) {
							String subsegment2 =
segment2.substring(index2, index2 + subsegmentLength);
							boolean subsegmentMatch = true;
							for (int index = 0; index <
subsegmentLength; index++) {
								char character1 =
subsegment1.charAt(index);
								char character2 =
subsegment2.charAt(index);
								if (!
areCharactersSimilar(character1, character2)) {
									subsegmentMatch = false;
									break;
								}
							}
							if (subsegmentMatch) {
								for (int index = index1; index <
index1 + subsegmentLength; index++) {
									used1[index] = true;
								}
								for (int index = index2; index <
index2 + subsegmentLength; index++) {
									used2[index] = true;
								}
								subsegmentMatchMap1.put(index1,
index2);
							}
						}
					}
				}
			}
		}
		TreeMap<Integer, Integer> subsegmentMatchMap2 = new TreeMap<Integer,
Integer>();
```

```
                int newIndex = 0;
                while (!(subsegmentMatchMap1.isEmpty())) {
                        int smallestIndex = 0x7fffffff;
                        for (int index: subsegmentMatchMap1.keySet()) {
                                if (index < smallestIndex) {
                                        smallestIndex = index;
                                }
                        }
                        subsegmentMatchMap2.put(newIndex++,
subsegmentMatchMap1.get(smallestIndex));
                        subsegmentMatchMap1.remove(smallestIndex);
                }
                TreeMap<Integer, Integer> subsegmentMatchMap3 = new TreeMap<Integer,
Integer>();
                newIndex = 0;
                while (!(subsegmentMatchMap2.isEmpty())) {
                        int indexOfSmallestValue = -1;
                        int smallestValue = 0x7fffffff;
                        for (int index: subsegmentMatchMap2.keySet()) {
                                int value = subsegmentMatchMap2.get(index);
                                if (value < smallestValue) {
                                        indexOfSmallestValue = index;
                                        smallestValue = value;
                                }
                        }
                        subsegmentMatchMap3.put(indexOfSmallestValue, newIndex++);
                        subsegmentMatchMap2.remove(indexOfSmallestValue);
                }
                int subsegmentCount = subsegmentMatchMap3.size();
                boolean[] usedIndex = new boolean[subsegmentCount];
                boolean[] usedValue = new boolean[subsegmentCount];
                for (int index = 0; index < subsegmentCount; index++) {
                        usedIndex[index] = false;
                        usedValue[index] = false;
                }
                int orderScore = 0;
                for (int index = 0; index < subsegmentCount; index++) {
                        int value = subsegmentMatchMap3.get(index);
                        if (index == value) {
                                usedIndex[index] = true;
                                usedValue[index] = true;
                        }
                }
                for (int index = 0; index < subsegmentCount; index++) {
                        if (!usedIndex[index]) {
                                for (int valueIndex = 0; valueIndex < subsegmentCount;
valueIndex++) {
                                        if (!usedValue[valueIndex]) {
                                                int value = subsegmentMatchMap3.get(valueIndex);
                                                if (index == value) {
                                                        usedIndex[index] = true;
                                                        usedValue[valueIndex] = true;
                                                        orderScore += Math.abs(index -
valueIndex);
                                                        break;
                                                }
                                        }
                                }
                        }
                }
                switch (formulaKind) {
                        case AVERAGE:
                                if (subsegmentCount % 2 == 0) {
```

```
                                        return 1 - orderScore * 2 / (float)(subsegmentCount *
subsegmentCount);
                                }
                                if (subsegmentCount == 1) {
                                        return 1;
                                } else {
                                        return 1 - orderScore * 2 / (float)(subsegmentCount *
subsegmentCount - 1);
                                }
                        case PRODUCT:
                                if (subsegmentCount % 2 == 0) {
                                        return 1 - orderScore * 2 / (float)(subsegmentCount *
subsegmentCount + 2);
                                }
                                return 1 - orderScore * 2 / (float)(subsegmentCount *
subsegmentCount + 1);
                        default:
                                return -1;
                }
        }
        private static float getLetterSimilarityScore(String word1, String word2, SegmentPair
pair) {
                int word1SegmentIndex = pair.getWord1SegmentIndex();
                int word2SegmentIndex = pair.getWord2SegmentIndex();
                int wordSegmentLength = pair.getWordSegmentLength();
                int word1SegmentEndIndex = word1SegmentIndex + wordSegmentLength;
                int word2SegmentEndIndex = word2SegmentIndex + wordSegmentLength;
                boolean[] used = new boolean[wordSegmentLength];
                for (int index = 0; index < wordSegmentLength; index++) {
                        used[index] = false;
                }
                int usedCount = 0;
                for (int index1 = word1SegmentIndex; index1 < word1SegmentEndIndex; index1++)
{
                        char character1 = word1.charAt(index1);
                        for (int index2 = word2SegmentIndex; index2 < word2SegmentEndIndex;
index2++) {
                                int usedIndex = index2 - word2SegmentIndex;
                                if (!used[usedIndex]) {
                                        char character2 = word2.charAt(index2);
                                        if (character1 == character2) {
                                                used[usedIndex] = true;
                                                usedCount++;
                                                break;
                                        }
                                }
                        }
                }
                switch (formulaKind) {
                        case AVERAGE:
                                return usedCount / (float)wordSegmentLength;
                        case PRODUCT:
                                return (usedCount + 1) / (float)(wordSegmentLength + 1);
                        default:
                                return -1;
                }
        }
        private static float getSizeScore(String word1, String word2, SegmentPair pair) {
                return pair.getWordSegmentLength() * 2 / (float)(word1.length() +
word2.length());
        }
        private static float getSegmentPairScore(String word1, String word2, SegmentPair
pair) {
```

```
		switch (formulaKind) {
			case AVERAGE:
				return (getOrderScore(word1, word2, pair) +
getLetterSimilarityScore(word1, word2, pair) + getSizeScore(word1, word2, pair)) / (float)3;
			case PRODUCT:
				return getOrderScore(word1, word2, pair) *
getLetterSimilarityScore(word1, word2, pair) * getSizeScore(word1, word2, pair);
			default:
				return -1;
		}
	}
	private static float getSegmentPairsScore(String word1, String word2,
ArrayList<SegmentPair> segmentPairs) {
		float highestSegmentPairScore = 0;
		for (SegmentPair pair: segmentPairs) {
			highestSegmentPairScore = Math.max(highestSegmentPairScore,
getSegmentPairScore(word1, word2, pair));
		}
		return highestSegmentPairScore;
	}
	/*private static float getCoverageScore(String word1, String word2,
ArrayList<SegmentPair> segmentPairs) {
		int word1Length = word1.length();
		int word2Length = word2.length();
		boolean[] used1 = new boolean[word1Length];
		boolean[] used2 = new boolean[word2Length];
		for (int index = 0; index < word1Length; index++) {
			used1[index] = false;
		}
		for (int index = 0; index < word2Length; index++) {
			used2[index] = false;
		}
		for (SegmentPair pair: segmentPairs) {
			int word1SegmentIndex = pair.getWord1SegmentIndex();
			int word2SegmentIndex = pair.getWord2SegmentIndex();
			int wordSegmentLength = pair.getWordSegmentLength();
			for (int index = word1SegmentIndex; index < word1SegmentIndex +
wordSegmentLength; index++) {
				used1[index] = true;
			}
			for (int index = word2SegmentIndex; index < word2SegmentIndex +
wordSegmentLength; index++) {
				used2[index] = true;
			}
		}
		int usedCount = 0;
		for (int index = 0; index < word1Length; index++) {
			if (used1[index]) {
				usedCount++;
			}
		}
		for (int index = 0; index < word2Length; index++) {
			if (used2[index]) {
				usedCount++;
			}
		}
		return usedCount / (float)(word1Length + word2Length);
	}
	private static float getSegmentPairPairOrderScore(String word1, String word2,
SegmentPair pair1, SegmentPair pair2) {
		int wordSegmentLength1 = pair1.getWordSegmentLength();
		int topStartIndex1 = pair1.getWord1SegmentIndex();
		int topEndIndex1 = topStartIndex1 + wordSegmentLength1 - 1;
```

```
                int bottomStartIndex1 = pair1.getWord2SegmentIndex();
                int bottomEndIndex1 = bottomStartIndex1 + wordSegmentLength1 - 1;
                int wordSegmentLength2 = pair2.getWordSegmentLength();
                int topStartIndex2 = pair2.getWord1SegmentIndex();
                int topEndIndex2 = topStartIndex2 + wordSegmentLength2 - 1;
                int bottomStartIndex2 = pair2.getWord2SegmentIndex();
                int bottomEndIndex2 = bottomStartIndex2 + wordSegmentLength2 - 1;
                float crossCheck1;
                if (bottomStartIndex1 == bottomStartIndex2) {
                        crossCheck1 = 0;
                } else {
                        crossCheck1 = (topStartIndex1 - topStartIndex2) / (float)
(bottomStartIndex1 - bottomStartIndex2);
                }
                float crossCheck2;
                if (bottomStartIndex1 == bottomEndIndex2) {
                        crossCheck2 = 0;
                } else {
                        crossCheck2 = (topStartIndex1 - topEndIndex2) / (float)
(bottomStartIndex1 - bottomEndIndex2);
                }
                float crossCheck3;
                if (bottomEndIndex1 == bottomStartIndex2) {
                        crossCheck3 = 0;
                } else {
                        crossCheck3 = (topEndIndex1 - topStartIndex2) / (float)(bottomEndIndex1
- bottomStartIndex2);
                }
                float crossCheck4;
                if (bottomEndIndex1 == bottomEndIndex2) {
                        crossCheck4 = 0;
                } else {
                        crossCheck4 = (topEndIndex1 - topEndIndex2) / (float)(bottomEndIndex1 -
bottomEndIndex2);
                }
                if (crossCheck1 > 0 && crossCheck2 > 0 && crossCheck3 > 0 && crossCheck4 > 0)
{
                        return 1;
                }
                int topOverlapScore = Math.max(topEndIndex1 - topStartIndex2, topEndIndex2 -
topStartIndex1) + 1;
                if (topOverlapScore < 0) {
                        topOverlapScore = 0;
                }
                int bottomOverlapScore = Math.max(bottomEndIndex1 - bottomStartIndex2,
bottomEndIndex2 - bottomStartIndex1) + 1;
                if (bottomOverlapScore < 0) {
                        bottomOverlapScore = 0;
                }
                return 1 - (topOverlapScore + bottomOverlapScore) / (float)((word1.length() +
word2.length()) * (wordSegmentLength1 + wordSegmentLength2));
        }
        private static float getSegmentPairsOrderScore(String word1, String word2,
ArrayList<SegmentPair> segmentPairs) {
                int segmentPairsSize = segmentPairs.size();
                if (segmentPairsSize <= 1) {
                        return 1;
                }
                float totalSegmentPairPairOrderScore = 0;
                for (int index1 = 0; index1 < segmentPairsSize - 1; index1++) {
                        for (int index2 = index1 + 1; index2 < segmentPairsSize; index2++) {
                                SegmentPair pair1 = segmentPairs.get(index1);
                                SegmentPair pair2 = segmentPairs.get(index2);
```

```
                                totalSegmentPairPairOrderScore +=
getSegmentPairPairOrderScore(word1, word2, pair1, pair2);
                        }
                }
                return totalSegmentPairPairOrderScore * 2 / (float)(segmentPairsSize *
(segmentPairsSize - 1));
        }*/
        private static float getSimpleCheckWordSimilarityScore(String word1, String word2,
ArrayList<SegmentPair> segmentPairs) {
                if (word1.equals(word2)) {
                        return 1;
                }
                return getSegmentPairsScore(word1, word2, segmentPairs);
        }
        private static TreeSet<String> getNewPronunciationAlternatives(String word, int
startIndex, String newPronunciationSegment, int lengthOfReplaced) {
                int wordLength = word.length();
                String newWord = word.substring(0, startIndex) + newPronunciationSegment +
word.substring(startIndex + lengthOfReplaced, wordLength);
                TreeSet<String> result = getPronunciationAlternatives(word, startIndex + 1);
         result.addAll(getPronunciationAlternatives(newWord, startIndex + 1));
                return result;
        }
        private static TreeSet<String> getPronunciationAlternatives(String word, int
startIndex) {
                int wordLength = word.length();
                if (startIndex >= wordLength) {
                        TreeSet<String> result = new TreeSet<String>();
                        result.add(word);
                        return result;
                }
                String substring1 = word.substring(startIndex, startIndex + 1);
                String substring2;
                if (startIndex + 2 <= wordLength) {
                        substring2 = word.substring(startIndex, startIndex + 2);
                } else {
                        substring2 = word.substring(startIndex, wordLength);
                }
                String substring3;
                if (startIndex + 3 <= wordLength) {
                        substring3 = word.substring(startIndex, startIndex + 3);
                } else {
                        substring3 = word.substring(startIndex, wordLength);
                }
                if (substring1.equals("b")) {
                        return getNewPronunciationAlternatives(word, startIndex, "v", 1);
                }
                if (substring1.equals("v")) {
                        return getNewPronunciationAlternatives(word, startIndex, "b", 1);
                }
                if (substring1.equals("h")) {
                        boolean canChange;
                        if (startIndex > 0) {
                                canChange = word.charAt(startIndex - 1) != 'c';
                        } else {
                                canChange = true;
                        }
                        if (canChange) {
                                return getNewPronunciationAlternatives(word, startIndex, "",
1);
                        }
                }
                if (substring1.equals("s")) {
```

```
                        return getNewPronunciationAlternatives(word, startIndex, "z", 1);
                }
                if (substring1.equals("z")) {
                        return getNewPronunciationAlternatives(word, startIndex, "s", 1);
                }
                if (substring1.equals("u") || substring1.equals("ú") ||
substring1.equals("ü")) {
                        boolean canChange;
                        if (substring1.equals("u")) {
                                if (startIndex > 0) {
                                        char previousCharacter = word.charAt(startIndex - 1);
                                        canChange = previousCharacter != 'g' &&
previousCharacter != 'q';
                                } else {
                                        canChange = true;
                                }
                        } else {
                                canChange = true;
                        }
                        if (canChange) {
                                return getNewPronunciationAlternatives(word, startIndex, "w",
1);
                        }
                }
                if (substring1.equals("w")) {
                        return getNewPronunciationAlternatives(word, startIndex, "u", 1);
                }
                if (substring2.startsWith("i") || substring2.startsWith("í")) {
                        boolean possiblyCanChange;
                        if (substring2.equals("i") || substring2.equals("í")) {
                                possiblyCanChange = true;
                        } else {
                                possiblyCanChange = isConsonant(substring2.charAt(1));
                        }
                        if (possiblyCanChange) {
                                boolean canChange;
                                if (startIndex > 1) {
                                        String previousTwoCharacters = word.substring(startIndex
- 2, startIndex);
                                        canChange = !(previousTwoCharacters.equals("gu")) && !
(previousTwoCharacters.equals("qu"));
                                } else {
                                        canChange = true;
                                }
                                if (canChange) {
                                        return getNewPronunciationAlternatives(word, startIndex,
"y", 1);
                                }
                        }
                }
                if (substring2.startsWith("y")) {
                        boolean canChange;
                        if (substring2.equals("y")) {
                                canChange = true;
                        } else {
                                canChange = isConsonant(substring2.charAt(1));
                        }
                        if (canChange) {
                                return getNewPronunciationAlternatives(word, startIndex, "i",
1);
                        }
                }
                if (substring2.equals("ll")) {
```

```
                        return getNewPronunciationAlternatives(word, startIndex, "y", 2);
                }
                if (substring2.startsWith("y")) {
                        boolean canChange;
                        if (substring2.equals("y")) {
                                canChange = false;
                        } else {
                                canChange = isVowel(substring2.charAt(1));
                        }
                        if (canChange) {
                                return getNewPronunciationAlternatives(word, startIndex, "ll",
1);
                        }
                }
                if (substring1.equals("x")) {
                        return getNewPronunciationAlternatives(word, startIndex, "cc", 1);
                }
                if (substring2.equals("cc")) {
                        return getNewPronunciationAlternatives(word, startIndex, "x", 2);
                }
                if (substring2.equals("ca")) {
                        return getNewPronunciationAlternatives(word, startIndex, "ka", 2);
                }
                if (substring2.equals("cá")) {
                        return getNewPronunciationAlternatives(word, startIndex, "ká", 2);
                }
                if (substring2.equals("ka")) {
                        return getNewPronunciationAlternatives(word, startIndex, "ca", 2);
                }
                if (substring2.equals("ká")) {
                        return getNewPronunciationAlternatives(word, startIndex, "cá", 2);
                }
                if (substring2.equals("ce")) {
                        return getNewPronunciationAlternatives(word, startIndex, "se", 2);
                }
                if (substring2.equals("cé")) {
                        return getNewPronunciationAlternatives(word, startIndex, "sé", 2);
                }
                if (substring2.equals("se")) {
                        return getNewPronunciationAlternatives(word, startIndex, "ce", 2);
                }
                if (substring2.equals("sé")) {
                        return getNewPronunciationAlternatives(word, startIndex, "cé", 2);
                }
                if (substring2.equals("ci")) {
                        return getNewPronunciationAlternatives(word, startIndex, "si", 2);
                }
                if (substring2.equals("cí")) {
                        return getNewPronunciationAlternatives(word, startIndex, "sí", 2);
                }
                if (substring2.equals("si")) {
                        return getNewPronunciationAlternatives(word, startIndex, "ci", 2);
                }
                if (substring2.equals("sí")) {
                        return getNewPronunciationAlternatives(word, startIndex, "cí", 2);
                }
                if (substring2.equals("co")) {
                        return getNewPronunciationAlternatives(word, startIndex, "ko", 2);
                }
                if (substring2.equals("có")) {
                        return getNewPronunciationAlternatives(word, startIndex, "kó", 2);
                }
                if (substring2.equals("ko")) {
```

```
                        return getNewPronunciationAlternatives(word, startIndex, "co", 2);
                }
                if (substring2.equals("kó")) {
                        return getNewPronunciationAlternatives(word, startIndex, "có", 2);
                }
                if (substring2.equals("cu")) {
                        return getNewPronunciationAlternatives(word, startIndex, "ku", 2);
                }
                if (substring2.equals("cú")) {
                        return getNewPronunciationAlternatives(word, startIndex, "kú", 2);
                }
                if (substring2.equals("ku")) {
                        return getNewPronunciationAlternatives(word, startIndex, "cu", 2);
                }
                if (substring2.equals("kú")) {
                        return getNewPronunciationAlternatives(word, startIndex, "cú", 2);
                }
                if (substring3.equals("que")) {
                        TreeSet<String> result = getNewPronunciationAlternatives(word,
startIndex, "ke", 3);
                        result.addAll(getNewPronunciationAlternatives(word, startIndex, "qe",
3));
                        return result;
                }
                if (substring3.equals("qué")) {
                        TreeSet<String> result = getNewPronunciationAlternatives(word,
startIndex, "ké", 3);
                        result.addAll(getNewPronunciationAlternatives(word, startIndex, "qé",
3));
                        return result;
                }
                if (substring2.equals("ke")) {
                        TreeSet<String> result = getNewPronunciationAlternatives(word,
startIndex, "que", 2);
                        result.addAll(getNewPronunciationAlternatives(word, startIndex, "qe",
2));
                        return result;
                }
                if (substring2.equals("ké")) {
                        TreeSet<String> result = getNewPronunciationAlternatives(word,
startIndex, "qué", 2);
                        result.addAll(getNewPronunciationAlternatives(word, startIndex, "qé",
2));
                        return result;
                }
                if (substring3.equals("qui")) {
                        TreeSet<String> result = getNewPronunciationAlternatives(word,
startIndex, "ki", 3);
                        result.addAll(getNewPronunciationAlternatives(word, startIndex, "qi",
3));
                        return result;
                }
                if (substring3.equals("quí")) {
                        TreeSet<String> result = getNewPronunciationAlternatives(word,
startIndex, "kí", 3);
                        result.addAll(getNewPronunciationAlternatives(word, startIndex, "qí",
3));
                        return result;
                }
                if (substring2.equals("ki")) {
                        return getNewPronunciationAlternatives(word, startIndex, "qui", 2);
                }
                if (substring2.equals("kí")) {
```

```
                        return getNewPronunciationAlternatives(word, startIndex, "quí", 2);
                }
                if (substring2.equals("ge")) {
                        return getNewPronunciationAlternatives(word, startIndex, "je", 2);
                }
                if (substring2.equals("gé")) {
                        return getNewPronunciationAlternatives(word, startIndex, "jé", 2);
                }
                if (substring2.equals("je")) {
                        return getNewPronunciationAlternatives(word, startIndex, "ge", 2);
                }
                if (substring2.equals("jé")) {
                        return getNewPronunciationAlternatives(word, startIndex, "gé", 2);
                }
                if (substring2.equals("gi")) {
                        return getNewPronunciationAlternatives(word, startIndex, "ji", 2);
                }
                if (substring2.equals("gí")) {
                        return getNewPronunciationAlternatives(word, startIndex, "jí", 2);
                }
                if (substring2.equals("ji")) {
                        return getNewPronunciationAlternatives(word, startIndex, "gi", 2);
                }
                if (substring2.equals("jí")) {
                        return getNewPronunciationAlternatives(word, startIndex, "gí", 2);
                }
                if (substring2.equals("qe")) {
                        TreeSet<String> result = getNewPronunciationAlternatives(word,
startIndex, "que", 2);
                        result.addAll(getNewPronunciationAlternatives(word, startIndex, "ke",
3));
                        return result;
                }
                if (substring2.equals("qé")) {
                        TreeSet<String> result = getNewPronunciationAlternatives(word,
startIndex, "qué", 2);
                        result.addAll(getNewPronunciationAlternatives(word, startIndex, "ké",
3));
                        return result;
                }
                if (substring2.equals("qi")) {
                        return getNewPronunciationAlternatives(word, startIndex, "qui", 2);
                }
                if (substring2.equals("qí")) {
                        return getNewPronunciationAlternatives(word, startIndex, "quí", 2);
                }
                return getPronunciationAlternatives(word, startIndex + 1);
        }
        private static TreeSet<String> getPronunciationAlternatives(String word) {
                return getPronunciationAlternatives(word, 0);
        }
        private static class WordSimilarityCacheInfo {
                private float wordSimilarityScore;
                public String previous;
                public String next;
                public WordSimilarityCacheInfo(float wordSimilarityScore, String previous,
String next) {
                        this.wordSimilarityScore = wordSimilarityScore;
                        this.previous = previous;
                        this.next = next;
                }
                public float getWordSimilarityScore() {
                        return wordSimilarityScore;
```

```
		}
	}
	private static TreeMap<String, WordSimilarityCacheInfo> wordSimilarityCache = new
TreeMap<String, WordSimilarityCacheInfo>();
	private static String firstCacheEntry = null;
	private static String lastCacheEntry = null;
	private static void memorizeWordSimilarityScore(String word1, String word2, float
wordSimilarityScore) {
		final int CACHE_SIZE = 100000;
		String wordToBeMemorized;
		if (word1.compareTo(word2) < 0) {
			wordToBeMemorized = word1 + "/" + word2;
		} else {
			wordToBeMemorized = word2 + "/" + word1;
		}
		if (wordSimilarityCache.containsKey(wordToBeMemorized)) {
			WordSimilarityCacheInfo wordSimilarityCacheInfo =
wordSimilarityCache.get(wordToBeMemorized);
			if (wordSimilarityCacheInfo.previous != null) {
				wordSimilarityCache.get(wordSimilarityCacheInfo.previous).next
= wordSimilarityCacheInfo.next;
			} else {
				firstCacheEntry = wordSimilarityCacheInfo.next;
			}
			if (wordSimilarityCacheInfo.next != null) {
				wordSimilarityCache.get(wordSimilarityCacheInfo.next).previous
= wordSimilarityCacheInfo.previous;
			} else {
				lastCacheEntry = wordSimilarityCacheInfo.previous;
			}
			wordSimilarityCache.get(lastCacheEntry).next = wordToBeMemorized;
			wordSimilarityCacheInfo.previous = lastCacheEntry;
			wordSimilarityCacheInfo.next = null;
			lastCacheEntry = wordToBeMemorized;
			return;
		}
		if (wordSimilarityCache.size() >= CACHE_SIZE) {
			WordSimilarityCacheInfo wordSimilarityCacheInfo =
wordSimilarityCache.get(firstCacheEntry);
			wordSimilarityCache.remove(firstCacheEntry);
			firstCacheEntry = wordSimilarityCacheInfo.next;
			wordSimilarityCacheInfo.next = null;
		}
		WordSimilarityCacheInfo wordSimilarityCacheInfo = new
WordSimilarityCacheInfo(wordSimilarityScore, lastCacheEntry, null);
		wordSimilarityCache.put(wordToBeMemorized, wordSimilarityCacheInfo);
		if (lastCacheEntry != null) {
			wordSimilarityCache.get(lastCacheEntry).next = wordToBeMemorized;
		}
		lastCacheEntry = wordToBeMemorized;
		if (firstCacheEntry == null) {
			firstCacheEntry = lastCacheEntry;
		}
	}
	private static float getWordSimilarityScore(String word1, String word2) {
		if (word1.equals(word2)) {
			return 1;
		}
		String wordToBeMemorized;
		if (word1.compareTo(word2) < 0) {
			wordToBeMemorized = word1 + "/" + word2;
		} else {
			wordToBeMemorized = word2 + "/" + word1;
```

```
                }
                if (wordSimilarityCache.containsKey(wordToBeMemorized)) {
                        return
wordSimilarityCache.get(wordToBeMemorized).getWordSimilarityScore();
                }
                TreeSet<String> word1Alternatives = getPronunciationAlternatives(word1);
                TreeSet<String> word2Alternatives = getPronunciationAlternatives(word2);
                float baseScore = 0;
                float highestScore = 0;
                for (String word1Alternative: word1Alternatives) {
                        for (String word2Alternative: word2Alternatives) {
                                ArrayList<SegmentPair> segmentPairs =
getSegmentPairs(word1Alternative, word2Alternative);
                                float simpleCheckWordSimilarityScore =
getSimpleCheckWordSimilarityScore(word1Alternative, word2Alternative, segmentPairs);
                                if (word1Alternative.equals(word1) &&
word2Alternative.equals(word2)) {
                                        baseScore = simpleCheckWordSimilarityScore;
                                }
                                if (simpleCheckWordSimilarityScore > highestScore) {
                                        highestScore = simpleCheckWordSimilarityScore;
                                }
                        }
                }
                float wordSimilarityScore = (baseScore + highestScore) / (float)2;
                memorizeWordSimilarityScore(word1, word2, wordSimilarityScore);
                return wordSimilarityScore;
        }
        private static float getMorphemeSimilarityScore(String checkMorpheme, float
checkPosition, String candidateMorpheme, TreeMap<Float, Integer>
candidateMorphemePositionData) {
                float wordSimilarityScore = getWordSimilarityScore(checkMorpheme,
candidateMorpheme);
                if (candidateMorphemePositionData.isEmpty()) {
                        return wordSimilarityScore;
                }
                if (candidateMorphemePositionData.containsKey(checkPosition)) {
                        return wordSimilarityScore;
                }
                Float ceilingPosition =
candidateMorphemePositionData.ceilingKey(checkPosition);
                Float floorPosition = candidateMorphemePositionData.floorKey(checkPosition);
                float ceilingFrequency;
                float floorFrequency;
                if (ceilingPosition == null) {
                        return wordSimilarityScore * ((checkPosition + floorPosition - 2) /
(float)((floorPosition - 1) * 2));
                } else {
                        ceilingFrequency = candidateMorphemePositionData.get(ceilingPosition);
                }
                if (floorPosition == null) {
                        return wordSimilarityScore * (((checkPosition / ceilingPosition) + 1) /
(float)2);
                } else {
                        floorFrequency = candidateMorphemePositionData.get(floorPosition);
                }
                if (checkPosition <= floorPosition + ((floorFrequency / (float)(floorFrequency
+ ceilingFrequency)) * (ceilingPosition - floorPosition))) {
                        return wordSimilarityScore * (
                                (
                                        (float)0.5 * (floorPosition - checkPosition) / (float)(
                                                (floorFrequency / (float)(floorFrequency +
ceilingFrequency)) * (ceilingPosition - floorPosition)
```

```
                                        )
                                ) + 1
                        );
                } else {
                        return wordSimilarityScore * (
                                (
                                        (float)0.5 * (checkPosition - ceilingPosition) / (float)
(
                                                (ceilingFrequency / (float)(floorFrequency +
ceilingFrequency)) * (ceilingPosition - floorPosition)
                                        )
                                ) + 1
                        );
                }
        }
        private static String lastChosenFilename = "";
        private static ArrayList<String> getChooserFileLines() {
                return getChooserFileLines("");
        }
        private static ArrayList<String> getChooserFileLines(String predefinedFilename) {
                String filename;
                if (predefinedFilename.equals("")) {
                        JFileChooser chooser = new JFileChooser();
                        FileNameExtensionFilter filter = new FileNameExtensionFilter("MGC
Corpus Files", "mgc");
                        chooser.setFileFilter(filter);
                        int result = chooser.showOpenDialog(null);
                        if (result != JFileChooser.APPROVE_OPTION) {
                                return null;
                        }
                        File file = chooser.getSelectedFile();
                        try {
                                filename = file.getCanonicalPath();
                        } catch (IOException e) {
                                System.out.println("I/O exception while trying to get canonical
path: \"" + e.getMessage() + "\"");
                                return null;
                        }
                } else {
                        filename = predefinedFilename;
                }
                lastChosenFilename = filename;
                FileReader reader;
                try {
                        reader = new FileReader(filename);
                } catch (FileNotFoundException e) {
                        System.out.println("File not found while attempting to open the chosen
file: \"" + e.getMessage() + "\"");
                        return null;
                }
                BufferedReader bufferedReader = new BufferedReader(reader);
                ArrayList<String> readLines = new ArrayList<String>();
                String line;
                do {
                        try {
                                line = bufferedReader.readLine().trim();
                        } catch (IOException e) {
                                System.out.println("I/O exception while trying to read the
chosen file: \"" + e.getMessage() + "\"");
                                try {
                                        bufferedReader.close();
                                } catch (IOException e2) {
                                        System.out.println("...aaand another I/O exception while
```

```
trying to close the buffered reader: \"" + e2.getMessage() + "\"");
                                }
                                try {
                                        reader.close();
                                } catch (IOException e2) {
                                        System.out.println("...aaand another I/O exception while
trying to close the file: \"" + e2.getMessage() + "\"");
                                }
                                return null;
                        } catch (NullPointerException e) {
                                line = "";
                                int character;
                                do {
                                        try {
                                                character = bufferedReader.read();
                                        } catch (IOException e2) {
                                                System.out.println("I/O exception while trying
to read the chosen file: \"" + e2.getMessage() + "\"");
                                                try {
                                                        bufferedReader.close();
                                                } catch (IOException e3) {
                                                        System.out.println("...aaand another I/O
exception while trying to close the buffered reader: \"" + e3.getMessage() + "\"");
                                                }
                                                try {
                                                        reader.close();
                                                } catch (IOException e3) {
                                                        System.out.println("...aaand another I/O
exception while trying to close the file: \"" + e3.getMessage() + "\"");
                                                }
                                                return null;
                                        }
                                        if (character >= 0) {
                                                line += String.valueOf((char)character);
                                        }
                                } while (character >= 0);
                        }
                        if (line != null) {
                                int semicolon = line.indexOf(';');
                                if (semicolon >= 0) {
                                        line = line.substring(0, semicolon).trim();
                                }
                                if (!(line.equals(""))) {
                                        readLines.add(line);
                                }
                        }
                        if (line.equals("")) {
                                line = null;
                        }
                } while (line != null);
                boolean closeOK = true;
                try {
                        bufferedReader.close();
                } catch (IOException e) {
                        System.out.println("I/O exception while trying to close the buffered
reader: \"" + e.getMessage() + "\"");
                        closeOK = false;
                }
                try {
                        reader.close();
                } catch (IOException e) {
                        System.out.println((closeOK ? "" : "...aaand another ") + "I/O
exception while trying to close the chosen file: \"" + e.getMessage() + "\"");
```

```
                        closeOK = false;
                }
                if (!closeOK) {
                        return null;
                }
                if (predefinedFilename.equals("")) {
                        System.out.println("Read file \"" + filename + "\", " +
readLines.size() + " lines.");
                }
                return readLines;
        }
        private static String[] wordList50            = {"gatotes",   "ultrafino",
"ultraconservador",    "taxis",  "beben",   "cosedora",    "repartirse",   "recibido",
"recibís",   "cuéntale",   "silloncito",    "ininterrumpido",    "reinscribir",   "inactivo",
"sobrepuesta",   "preacuerdo",   "exnovia",   "vicepresidente",   "vicedecanato",
"extracurricular",   "anticonstitucionales",     "observarán",    "extensionalidad",
"principado",  "tolerancia",  "bienaventuranza",   "incertidumbre",   "torcedura",
"cabezota",   "panezote",   "villorrio",   "colombiano",   "vizcaína",   "tepozteco",
"comías",    "comiste",  "cantaremos",    "viviré",   "vivirá",   "viviríamos",
"vivirían",    "cantásemos",    "cantáremos",    "cantareis",    "vivieren",    "librería",
"caserío",  "jugaré",   "superestadistas",    "escogería"  };
        private static String[] morphemyzedWordList50 = {"gat-ot-es", "ultra-fin-o", "ultra-
conserv-a-dor", "taxi-s", "beb-e-n", "cos-e-dor-a", "repart-ir-se", "recib-id-o", "recib-í-
s", "cuént-a-le", "sill-on-cit-o", "in-interrump-id-o", "re-inscrib-ir", "in-act-iv-o",
"sobre-puest-a", "pre-acuerd-o", "ex-novi-a", "vice-presid-ente", "vice-decan-ato", "extra-
curricul-ar", "anti-constitu-cion-al-es", "observ-a-rá-n", "extension-al-i-dad", "princip-
ado", "toler-ancia", "bien-aventur-anza", "in-certi-dumbre", "torc-edura", "cabez-ot-a",
"pan-ezot-e", "vill-orri-o", "colombi-an-o", "vizca-ín-a", "tepoz-tec-o", "com-í-a-s", "com-
iste", "cant-a-re-mos", "viv-i-ré", "viv-i-rá", "viv-i-ría-mos", "viv-i-ría-n", "cant-á-se-
mos", "cant-á-re-mos", "cant-a-re-is", "viv-ie-re-n", "libr-ería", "cas-erío", "jug-a-ré",
"super-estad-ista-s", "escog-e-ría"};
        private static void LearningSystem1(Scanner keyboard, PlainLearningMorphemeDataGetter
getter, boolean startingFromScratch, boolean automatic) {
                int automaticCounter = -1;   //100 <<< To begin at "with position data"
                if (automatic) {
                        System.out.println("*** Without position data ***");
                } else {
                        System.out.println("Learning system without learning position data.");
                        System.out.println("Enter a word to analyze it.");
                        System.out.println("Type an equals sign and a morphemyzed word to learn
it.");
                        if (startingFromScratch) {
                                System.out.println("Note: Starting from scratch.");
                        }
                }
                int phase = 1;
                int dnl = 0;
                int imnt = 0;
                int imwt = 0;
                int rtnt = 0;
                int rtwt = 0;
                while (true) {
                        boolean retry = false;
                        String input = "";
                        String automaticWord = "";
                        String automaticMorphemyzedWord = "";
                        if (automatic) {
                                if (phase == 1) {
                                        automaticCounter++;
                                        if (automaticCounter >= wordList50.length ||
automaticCounter >= morphemyzedWordList50.length) {
                                                System.out.println("=== Summary of results
for \"without position data\" modality ===");
```

```
                                                System.out.println("Did not learn: " + dnl);
                                                System.out.println("Immediate result (no ties):
" + imnt);
                                                System.out.println("Immediate result (with
ties): " + imwt);
                                                System.out.println("Requires teaching (no ties):
" + rtnt);
                                                System.out.println("Requires teaching (with
ties): " + rtwt);
                                                return;
                                        }
                                }
                                automaticWord = wordList50[automaticCounter];
                                automaticMorphemyzedWord =
morphemyzedWordList50[automaticCounter];
                                if (phase == 1) {
                                        System.out.print(automaticWord + " ... " +
automaticMorphemyzedWord + " ... ");
                                        input = automaticWord;
                                } else {
                                        if (phase == 2) {
                                                input = "=" + automaticMorphemyzedWord;
                                        } else {
                                                input = automaticWord;
                                        }
                                }
                        } else {
                                System.out.print(">");
                                input = keyboard.nextLine().trim();
                                if (input.equals("")) {
                                        System.out.print("Terminate system? (y/N)>");
                                        input = keyboard.nextLine().trim().toLowerCase();
                                        System.out.println();
                                        if (input.equals("y") || input.equals("yes")) {
                                                break;
                                        } else {
                                                retry = true;
                                        }
                                }
                        }
                        if (!retry) {
                                int inputLength = input.length();
                                if (input.charAt(0) == '=') {
                                        input = "=" + input.substring(1, inputLength).trim();
                                        inputLength = input.length();
                                        if (inputLength > 1) {
                                                boolean morphemyzedSpanish = true;
                                                for (int index = 1; index < inputLength; index+
+) {
                                                        if (!
isSpanishLetterOrDash(input.charAt(index))) {
                                                                morphemyzedSpanish = false;
                                                                break;
                                                        }
                                                }
                                                if (morphemyzedSpanish) {
                                                        MorphemyzedWord morphemyzedWord = null;
                                                        try {
                                                                morphemyzedWord = new
MorphemyzedWord(getter, input.substring(1, inputLength));
                                                        } catch (Exception e) {

        System.out.println(e.getMessage());
```

```
                                                }
                                                if (morphemyzedWord != null) {
                                                        ArrayList<String> morphemes =
morphemyzedWord.getMorphemes();
                                                        for (String morpheme: morphemes)
{

        getter.addCandidateMorpheme(morpheme);
                                                        }
                                                        if (automatic) {
                                                                phase = 3;
                                                        } else {
                                                                System.out.println("OK.");
                                                        }
                                                }
                                        } else {
                                                System.out.println("Please only use
dashes or Spanish characters.");
                                        }
                                }
                        } else {
                                boolean spanish = true;
                                for (int index = 0; index < inputLength; index++) {
                                        if (!isSpanishLetter(input.charAt(index))) {
                                                spanish = false;
                                                break;
                                        }
                                }
                                if (spanish) {
                                        WordMorphemyzerGeneticAlgorithm geneticAlgorithm
= new WordMorphemyzerGeneticAlgorithm(input, getter);
                                        TreeSet<MorphemyzedWord> morphemyzedWords =
geneticAlgorithm.run(automatic);
                                        int morphemyzedWordsSize =
morphemyzedWords.size();
                                        if (automatic) {
                                                if (phase == 1) {
                                                        phase = 2;
                                                        for (MorphemyzedWord
morphemyzedWord: morphemyzedWords) {
                                                                if
(automaticMorphemyzedWord.equals(morphemyzedWord.getDisplayString())) {

        System.out.println("Immediate result" + (morphemyzedWordsSize == 1 ? "" : ", ties
with " + (morphemyzedWordsSize - 1) + " other" + (morphemyzedWordsSize == 2 ? "" : "s")));
                                                                        if
(morphemyzedWordsSize == 1) {
                                                                                imnt++;
                                                                        } else {
                                                                                imwt++;
                                                                        }
                                                                        phase = 1;
                                                                        break;
                                                                }
                                                        }
                                                } else {
                                                        boolean didNotLearn = true;
                                                        for (MorphemyzedWord
morphemyzedWord: morphemyzedWords) {
                                                                if
(automaticMorphemyzedWord.equals(morphemyzedWord.getDisplayString())) {

        System.out.println("Requires teaching" + (morphemyzedWordsSize == 1 ? "" : ", ties
```

```
with " + (morphemyzedWordsSize - 1) + " other" + (morphemyzedWordsSize == 2 ? "" : "s")));
                                                                        if
(morphemyzedWordsSize == 1) {
                                                                                rtnt++;
                                                                        } else {
                                                                                rtwt++;
                                                                        }
                                                                        didNotLearn =
false;
                                                                        break;
                                                                }
                                                        }
                                                        if (didNotLearn) {
                                                                System.out.println("Did
not learn");
                                                                dnl++;
                                                        }
                                                        phase = 1;
                                                }
                                        } else {
                                                System.out.print("Result" +
(morphemyzedWordsSize == 1 ? "" : "s") + " (" + morphemyzedWordsSize + "): ");
                                                boolean firstWord = true;
                                                for (MorphemyzedWord morphemyzedWord:
morphemyzedWords) {
                                                        if (firstWord) {
                                                                firstWord = false;
                                                        } else {
                                                                System.out.print(", ");
                                                        }

        System.out.print(morphemyzedWord.getDisplayString());
                                                }
                                                System.out.println();
                                                if (morphemyzedWords.size() == 1) {
                                                        MorphemyzedWord morphemyzedWord =
(new ArrayList<MorphemyzedWord>(morphemyzedWords)).get(0);
                                                        ArrayList<String> morphemes =
morphemyzedWord.getMorphemes();
                                                        boolean fullyKnown = true;
                                                        for (String morpheme: morphemes)
{
                                                                if (!
(getter.knowsCandidateMorpheme(morpheme))) {
                                                                        fullyKnown =
false;
                                                                        break;
                                                                }
                                                        }
                                                        if (!fullyKnown) {
                                                                System.out.print("Correct?
(y/N)>");
                                                                input =
keyboard.nextLine().trim().toLowerCase();
                                                                if (input.equals("y") ||
input.equals("yes")) {
                                                                        for (String
morpheme: morphemes) {

        getter.addCandidateMorpheme(morpheme);
                                                                        }
                                                                }
                                                        }
```

```
                                                        }
                                                }
                                        } else {
                                                System.out.println("Please only use Spanish
characters.");
                                        }
                                }
                                if (!automatic) {
                                        System.out.println();
                                }
                        }
                }
        }
        private static void LearningSystem2(Scanner keyboard, FullLearningMorphemeDataGetter
getter, boolean startingFromScratch, boolean automatic) {
                int automaticCounter = -1;
                if (automatic) {
                        for (String line: morphemyzedWordList50) {
                                String input = line;
                                int inputLength = input.length();
                                if (input.charAt(0) == '-' || input.charAt(inputLength - 1) ==
'-') {
                                        System.out.println("Should not begin or end with a
dash.");
                                        System.exit(-1);
                                } else {
                                        int wordLength = inputLength;
                                        for (int index = 0; index < inputLength; index++) {
                                                char character = input.charAt(index);
                                                if (character == '-') {
                                                        wordLength--;
                                                }
                                        }
                                        input += "-";
                                        inputLength++;
                                        int startIndex = 0;
                                        int dashes = 0;
                                        for (int index = 0; index < inputLength; index++) {
                                                char character = input.charAt(index);
                                                if (character == '-') {
                                                        String morpheme =
input.substring(startIndex, index);
                                                        int morphemeLength = morpheme.length();
                                                        float position = (startIndex - dashes) /
(float)(wordLength - morphemeLength);
                                                        startIndex = index + 1;
                                                        for (int index2 = 0; index2 <
LEARN_REPETITIONS - 1; index2++) {

        getter.addCandidateMorphemeData(morpheme, position);
                                                        }
                                                        dashes++;
                                                }
                                        }
                                }
                        }
                        System.out.println("*** With position data ***");
                } else {
                        System.out.println("Learning system while learning position data.");
                        System.out.println("Enter a word to analyze it.");
                        System.out.println("Type an equals sign and a morphemyzed word to learn
it.");
                        if (startingFromScratch) {
```

```
                        System.out.println("Note: Starting from scratch.");
                }
        }
        int phase = 1;
        int dnl = 0;
        int imnt = 0;
        int imwt = 0;
        int rtnt = 0;
        int rtwt = 0;
        do {
                while (true) {
                        boolean retry = false;
                        String input = "";
                        String automaticWord = "";
                        String automaticMorphemyzedWord = "";
                        if (automatic) {
                                if (phase == 1) {
                                        automaticCounter++;
                                        if (automaticCounter >= wordList50.length ||
automaticCounter >= morphemyzedWordList50.length) {
                                                System.out.println("=== Summary of
results for \"with position data\" modality ===");
                                                System.out.println("Did not learn: " +
dnl);
                                                System.out.println("Immediate result (no
ties): " + imnt);
                                                System.out.println("Immediate result
(with ties): " + imwt);
                                                System.out.println("Requires teaching (no
ties): " + rtnt);
                                                System.out.println("Requires teaching
(with ties): " + rtwt);
                                                System.out.println();
                                                return;
                                        }
                                }
                                automaticWord = wordList50[automaticCounter];
                                automaticMorphemyzedWord =
morphemyzedWordList50[automaticCounter];
                                if (phase == 1) {
                                        System.out.print(automaticWord + " ... " +
automaticMorphemyzedWord + " ... ");
                                        input = automaticWord;
                                } else {
                                        if (phase == 2) {
                                                input = "=" + automaticMorphemyzedWord;
                                        } else {
                                                input = automaticWord;
                                        }
                                }
                        } else {
                                System.out.print(">");
                                input = keyboard.nextLine().trim();
                                if (input.equals("")) {
                                        System.out.print("Terminate system? (y/N)>");
                                        input =
keyboard.nextLine().trim().toLowerCase();
                                        System.out.println();
                                        if (input.equals("y") || input.equals("yes")) {
                                                break;
                                        } else {
                                                retry = true;
                                        }
```

```
                                }
                        }
                        if (!retry) {
                                int inputLength = input.length();
                                if (input.charAt(0) == '=') {
                                        input = "=" + input.substring(1,
inputLength).trim();
                                        inputLength = input.length();
                                        if (inputLength > 1) {
                                                boolean morphemyzedSpanish = true;
                                                for (int index = 1; index < inputLength;
index++) {
                                                        if (!
isSpanishLetterOrDash(input.charAt(index))) {
                                                                morphemyzedSpanish =
false;
                                                                break;
                                                        }
                                                }
                                                if (morphemyzedSpanish) {
                                                        input = input.substring(1,
inputLength);
                                                        inputLength--;
                                                        if (input.charAt(0) == '-' ||
input.charAt(inputLength - 1) == '-') {
                                                                System.out.println("Should
not begin or end with a dash.");
                                                        } else {
                                                                int wordLength =
inputLength;
                                                                for (int index = 0; index
< inputLength; index++) {
                                                                        char character =
input.charAt(index);
                                                                        if (character ==
'-') {

        wordLength--;
                                                                        }
                                                                }
                                                                input += "-";
                                                                inputLength++;
                                                                int startIndex = 0;
                                                                int dashes = 0;
                                                                for (int index = 0; index
< inputLength; index++) {
                                                                        char character =
input.charAt(index);
                                                                        if (character ==
'-') {
                                                                                String
morpheme = input.substring(startIndex, index);
                                                                                int
morphemeLength = morpheme.length();
                                                                                float
position = (startIndex - dashes) / (float)(wordLength - morphemeLength);
                                                                                startIndex
= index + 1;

        getter.addCandidateMorphemeData(morpheme, position);
                                                                                dashes++;
                                                                        }
                                                                }
```

```
                                                                        if (automatic) {
                                                                                phase = 3;
                                                                        } else {

        System.out.println("OK.");
                                                                        }
                                                                }
                                                        } else {
                                                                System.out.println("Please only
use dashes or Spanish characters.");
                                                        }
                                                }
                                        } else {
                                                boolean spanish = true;
                                                for (int index = 0; index < inputLength; index+
+) {
                                                        if (!
isSpanishLetter(input.charAt(index))) {
                                                                spanish = false;
                                                                break;
                                                        }
                                                }
                                                if (spanish) {
                                                        WordMorphemyzerGeneticAlgorithm
geneticAlgorithm = new WordMorphemyzerGeneticAlgorithm(input, getter);
                                                        TreeSet<MorphemyzedWord> morphemyzedWords
= geneticAlgorithm.run(automatic);
                                                        int morphemyzedWordsSize =
morphemyzedWords.size();
                                                        if (automatic) {
                                                                if (phase == 1) {
                                                                        phase = 2;
                                                                        for (MorphemyzedWord
morphemyzedWord: morphemyzedWords) {
                                                                                if
(automaticMorphemyzedWord.equals(morphemyzedWord.getDisplayString())) {

        System.out.println("Immediate result" + (morphemyzedWordsSize == 1 ? "" : ", ties
with " + (morphemyzedWordsSize - 1) + " other" + (morphemyzedWordsSize == 2 ? "" : "s")));
                                                                                        if
(morphemyzedWordsSize == 1) {

        imnt++;
                                                                                        } else {

        imwt++;
                                                                                        }
                                                                                        phase = 1;
                                                                                        break;
                                                                                }
                                                                        }
                                                                } else {
                                                                        boolean didNotLearn =
true;
                                                                        for (MorphemyzedWord
morphemyzedWord: morphemyzedWords) {
                                                                                if
(automaticMorphemyzedWord.equals(morphemyzedWord.getDisplayString())) {

        System.out.println("Requires teaching" + (morphemyzedWordsSize == 1 ? "" : ", ties
with " + (morphemyzedWordsSize - 1) + " other" + (morphemyzedWordsSize == 2 ? "" : "s")));
                                                                                        if
(morphemyzedWordsSize == 1) {
```

```
        rtnt++;
                                                                                } else {
        rtwt++;
                                                                                }
                                                                                didNotLearn
= false;
                                                                                break;
                                                                        }
                                                                }
                                                                if (didNotLearn) {
        System.out.println("Did not learn");
                                                                        dnl++;
                                                                }
                                                                phase = 1;
                                                        }
                                                } else {
                                                        System.out.print("Result" +
(morphemyzedWordsSize == 1 ? "" : "s") + " (" + morphemyzedWordsSize + "): ");
                                                        boolean firstWord = true;
                                                        for (MorphemyzedWord
morphemyzedWord: morphemyzedWords) {
                                                                if (firstWord) {
                                                                        firstWord = false;
                                                                } else {
        System.out.print(", ");
                                                                }
        System.out.print(morphemyzedWord.getDisplayString());
                                                        }
                                                        System.out.println();
                                                        if (morphemyzedWords.size() == 1)
{
                                                                MorphemyzedWord
morphemyzedWord = (new ArrayList<MorphemyzedWord>(morphemyzedWords)).get(0);
                                                                ArrayList<String>
morphemes = morphemyzedWord.getMorphemes();
                                                                boolean fullyKnown = true;
                                                                for (String morpheme:
morphemes) {
                                                                        if (!
(getter.knowsCandidateMorpheme(morpheme))) {
                                                                                fullyKnown
= false;
                                                                                break;
                                                                        }
                                                                }
                                                                if (!fullyKnown) {
        System.out.print("Correct? (y/N)>");
                                                                        input =
keyboard.nextLine().trim().toLowerCase();
                                                                        if
(input.equals("y") || input.equals("yes")) {
                                                                                input =
morphemyzedWord.getDisplayString();
                                                                                inputLength
= input.length();
                                                                                int
wordLength = inputLength;
```

```
                                                                        for (int
index = 0; index < inputLength; index++) {
        char character = input.charAt(index);
                                                                                if
(character == '-') {
        wordLength--;
                                                                                }
                                                                        }
                                                                        input +=
"-";
        inputLength++;
                                                                        int
startIndex = 0;
                                                                        int dashes
= 0;
                                                                        for (int
index = 0; index < inputLength; index++) {
        char character = input.charAt(index);
                                                                                if
(character == '-') {
        String morpheme = input.substring(startIndex, index);
        int morphemeLength = morpheme.length();
        float position = (startIndex - dashes) / (float)(wordLength - morphemeLength);
        startIndex = index + 1;
        getter.addCandidateMorphemeData(morpheme, position);
        dashes++;
                                                                                }
                                                                        }
                                                                                }
                                                                        }
                                                                }
                                                        }
                                                        } else {
                                                                System.out.println("Please only use
Spanish characters.");
                                                        }
                                                }
                                                if (!automatic) {
                                                        System.out.println();
                                                }
                                        }
                                }
                        } while (false);
        }
        public static void main(String[] args) {
                Scanner keyboard = new Scanner(System.in);
                boolean quit = false;
                do {
                        boolean retry;
                        int selection = -1;
                        do {
                                System.out.println("Morphological analysis of Spanish words
using genetic algorithms");
```

```
                                switch (formulaKind) {
                                        case AVERAGE:
                                                System.out.println("Using average formula");
                                                break;
                                        case PRODUCT:
                                                System.out.println("Using product formula");
                                                break;
                                }
                                System.out.println();
                                System.out.println("1. Detailed comparison of two words");
                                System.out.println("2. Consecutive string of words");
                                System.out.println("3. Learning system without learning
position data, from scratch");
                                System.out.println("4. Learning system while learning position
data, from scratch");
                                System.out.println("5. Learning system without learning
position data, from corpus");
                                System.out.println("6. Learning system while learning position
data, from corpus");
                                System.out.println("7. Automatic pre-defined hundred test
run");
                                System.out.println("8. Quit");
                                String selectionAsString = keyboard.nextLine().trim();
                                retry = false;
                                try {
                                        selection = Integer.valueOf(selectionAsString);
                                } catch (NumberFormatException e) {
                                        retry = true;
                                }
                        } while (retry);
                        switch (selection) {
                                case 1:
                                        while (true) {
                                                System.out.print("Word #1>");
                                                String word1 = keyboard.nextLine().trim();
                                                if (word1.equals("")) {
                                                        break;
                                                }
                                                System.out.print("Word #2>");
                                                String word2 = keyboard.nextLine().trim();
                                                if (word2.equals("")) {
                                                        break;
                                                }
                                                ArrayList<SegmentPair> segmentPairs =
getSegmentPairs(word1, word2);
                                                if (segmentPairs.size() == 0) {
                                                        System.out.println("There are no segment
pairs.");
                                                        System.out.println("Word similarity
score: " + (getWordSimilarityScore(word1, word2) * 100) + "%");
                                                        System.out.println();
                                                } else {
                                                        System.out.println("Found " +
segmentPairs.size() + " segment pair" + (segmentPairs.size() == 1 ? "" : "s") + ".");
                                                        for (SegmentPair pair: segmentPairs) {
                                                                int word1SegmentIndex =
pair.getWord1SegmentIndex();
                                                                int word2SegmentIndex =
pair.getWord2SegmentIndex();
                                                                int wordSegmentLength =
pair.getWordSegmentLength();

        System.out.println(word1.substring(0, word1SegmentIndex) + "[" +
```

```
word1.substring(word1SegmentIndex, word1SegmentIndex + wordSegmentLength) + "]" +
word1.substring(word1SegmentIndex + wordSegmentLength, word1.length()));

       System.out.println(word2.substring(0, word2SegmentIndex) + "[" +
word2.substring(word2SegmentIndex, word2SegmentIndex + wordSegmentLength) + "]" +
word2.substring(word2SegmentIndex + wordSegmentLength, word2.length()));
                                                              float orderScore =
getOrderScore(word1, word2, pair);
                                                              System.out.println("Order score:
" + (orderScore * 100) + "%");
                                                              float letterSimilarityScore =
getLetterSimilarityScore(word1, word2, pair);
                                                              System.out.println("Letter
similarity score: " + (letterSimilarityScore * 100) + "%");
                                                              float sizeScore =
getSizeScore(word1, word2, pair);
                                                              System.out.println("Size score: "
+ (sizeScore * 100) + "%");
                                                              float segmentPairScore =
getSegmentPairScore(word1, word2, pair);
                                                              System.out.println("Segment pair
score: " + (segmentPairScore * 100) + "%");
                                                              System.out.println();
                                                      }
                                                      float segmentPairsScore =
getSegmentPairsScore(word1, word2, segmentPairs);
                                                      System.out.println("Segment pairs score:
" + (segmentPairsScore * 100) + "%");
                                                      System.out.println();
                                                      float simpleCheckWordSimilarityScore =
getSimpleCheckWordSimilarityScore(word1, word2, segmentPairs);
                                                      System.out.println("Simple check word
similarity score (without pronunciation checks): " + (simpleCheckWordSimilarityScore * 100) +
"%");
                                                      float wordSimilarityScore =
getWordSimilarityScore(word1, word2);
                                                      System.out.println("Final word similarity
score: " + (wordSimilarityScore * 100) + "%");
                                                      System.out.println();
                                              }
                                      }
                                      break;
                              case 2:
                                      ArrayList<String> words2 = new ArrayList<String>();
                                      while (true) {
                                              words2.clear();
                                              int number = 1;
                                              while (true) {
                                                      System.out.print("Word #" + (number++) +
">");
                                                      String word = keyboard.nextLine().trim();
                                                      if (word.equals("")) {
                                                              break;
                                                      } else {
                                                              words2.add(word);
                                                      }
                                              }
                                              if (words2.isEmpty()) {
                                                      break;
                                              }
                                              int wordsSize = words2.size();
                                              float[][] scoresMatrix = new float[wordsSize]
[wordsSize];
```

```
{
                                                for (int index = 0; index < wordsSize; index++)
                                                        scoresMatrix[index][index] = 1;
                                                }
                                                for (int outerWordIndex = 0; outerWordIndex <=
wordsSize - 2; outerWordIndex++) {
                                                        String outerWord =
words2.get(outerWordIndex);
                                                        for (int innerWordIndex = wordsSize - 1;
innerWordIndex > outerWordIndex; innerWordIndex--) {
                                                                String innerWord =
words2.get(innerWordIndex);
                                                                System.out.println("Comparing \""
+ outerWord + "\" and \"" + innerWord + "\".");
                                                                float wordSimilarityScore =
getWordSimilarityScore(outerWord, innerWord);
                                                                scoresMatrix[outerWordIndex]
[innerWordIndex] = wordSimilarityScore;
                                                                scoresMatrix[innerWordIndex]
[outerWordIndex] = wordSimilarityScore;
                                                        }
                                                }
                                                float highestWordSimilarityScore = -1;
                                                int highestIndex = -1;
                                                for (int outerIndex = 0; outerIndex < wordsSize;
outerIndex++) {
                                                        float totalSimilarityScore = 0;
                                                        for (int innerIndex = 0; innerIndex <
wordsSize; innerIndex++) {
                                                                totalSimilarityScore +=
scoresMatrix[innerIndex][outerIndex];
                                                        }
                                                        if (totalSimilarityScore >
highestWordSimilarityScore) {
                                                                highestWordSimilarityScore =
totalSimilarityScore;
                                                                highestIndex = outerIndex;
                                                        }
                                                }
                                                boolean[] used = new boolean[wordsSize];
                                                for (int index = 0; index < wordsSize; index++)
{
                                                        used[index] = false;
                                                }
                                                while (true) {
                                                        String highestWord =
words2.get(highestIndex);
                                                        System.out.println(highestWord);
                                                        used[highestIndex] = true;
                                                        boolean done = true;
                                                        for (int index = 0; index < wordsSize;
index++) {
                                                                if (!used[index]) {
                                                                        done = false;
                                                                        break;
                                                                }
                                                        }
                                                        if (done) {
                                                                System.out.println();
                                                                break;
                                                        }
                                                        float highestSimilarityScore = -1;
                                                        int previousHighestIndex = highestIndex;
```

```
                                                        highestIndex = -1;
                                                        for (int index = 0; index < wordsSize;
index++) {
                                                               if (!used[index]) {
                                                                      float similarityScore =
scoresMatrix[index][previousHighestIndex];
                                                                      if (similarityScore >
highestSimilarityScore) {

       highestSimilarityScore = similarityScore;
                                                                             highestIndex =
index;
                                                                      }
                                                               }
                                                        }
                                                        used[highestIndex] = true;
                                                 }
                                          }
                                          break;
                                   case 3:
                                          do {
                                                 PlainLearningMorphemeDataGetter getter = new
PlainLearningMorphemeDataGetter();
                                                 LearningSystem1(keyboard, getter, true, false);
                                          } while (false);
                                          break;
                                   case 4:
                                          do {
                                                 FullLearningMorphemeDataGetter getter = new
FullLearningMorphemeDataGetter();
                                                 LearningSystem2(keyboard, getter, true, false);
                                          } while (false);
                                          break;
                                   case 5:
                                          PrefixLearningSystem1(keyboard, false);
                                          break;
                                   case 6:
                                          PrefixLearningSystem2(keyboard, "", false);
                                          break;
                                   case 7:
                                          lastChosenFilename = "";
                                          PrefixLearningSystem1(keyboard, true);
                                          if (!(lastChosenFilename.equals(""))) {
                                                 PrefixLearningSystem2(keyboard,
lastChosenFilename, true);
                                          }
                                          break;
                                   case 8:
                                          quit = true;
                                          break;
                                   default:
                                          break;
                            }
                     } while (!quit);
                     keyboard.close();
              }
              private static void PrefixLearningSystem1(Scanner keyboard, boolean automatic) {
                     do {
                            ArrayList<String> readLines = getChooserFileLines();
                            if (readLines == null) {
                                   break;
                            }
                            boolean breakAgain = false;
```

```
                        boolean breakYetAgain = false;
                        for (String line: readLines) {
                                int lineLength = line.length();
                                for (int i = 0; i < lineLength; i++) {
                                        char character = line.charAt(i);
                                        if (!isSpanishLetterOrDash(character)) {
                                                System.out.println("Error: Data must only
contain Spanish characters or dashes.");
                                                System.out.println("Offending entry: \"" + line
+ "\".");
                                                breakAgain = true;
                                                break;
                                        }
                                }
                                if (breakAgain) {
                                        breakYetAgain = true;
                                        break;
                                }
                        }
                        if (breakYetAgain) {
                                break;
                        }
                        PlainLearningMorphemeDataGetter getter = new
PlainLearningMorphemeDataGetter();
                        for (String line: readLines) {
                                String input = "=" + line;
                                int inputLength = input.length();
                                MorphemyzedWord morphemyzedWord = null;
                                try {
                                        morphemyzedWord = new MorphemyzedWord(getter,
input.substring(1, inputLength));
                                } catch (Exception e) {
                                        System.out.println(e.getMessage());
                                        breakAgain = true;
                                }
                                ArrayList<String> morphemes = morphemyzedWord.getMorphemes();
                                for (String morpheme: morphemes) {
                                        getter.addCandidateMorpheme(morpheme);
                                }
                        }
                        if (breakAgain) {
                                break;
                        }
                        LearningSystem1(keyboard, getter, false, automatic);
                } while (false);
        }
        private static void PrefixLearningSystem2(Scanner keyboard, String
predefinedFilename, boolean automatic) {
                do {
                        ArrayList<String> readLines = getChooserFileLines(predefinedFilename);
                        if (readLines == null) {
                                break;
                        }
                        boolean breakAgain = false;
                        boolean breakYetAgain = false;
                        for (String line: readLines) {
                                int lineLength = line.length();
                                for (int i = 0; i < lineLength; i++) {
                                        char character = line.charAt(i);
                                        if (!isSpanishLetterOrDash(character)) {
                                                System.out.println("Error: Data must only
contain Spanish characters or dashes.");
                                                System.out.println("Offending entry: \"" + line
```

```
+ "\".");
                                                breakAgain = true;
                                                break;
                                        }
                                }
                                if (breakAgain) {
                                        breakYetAgain = true;
                                        break;
                                }
                        }
                        if (breakYetAgain) {
                                break;
                        }
                        FullLearningMorphemeDataGetter getter = new
FullLearningMorphemeDataGetter();
                        for (String line: readLines) {
                                String input = line;
                                int inputLength = input.length();
                                if (input.charAt(0) == '-' || input.charAt(inputLength - 1) ==
'-') {
                                        System.out.println("Should not begin or end with a
dash.");
                                        breakAgain = true;
                                        break;
                                } else {
                                        int wordLength = inputLength;
                                        for (int index = 0; index < inputLength; index++) {
                                                char character = input.charAt(index);
                                                if (character == '-') {
                                                        wordLength--;
                                                }
                                        }
                                        input += "-";
                                        inputLength++;
                                        int startIndex = 0;
                                        int dashes = 0;
                                        for (int index = 0; index < inputLength; index++) {
                                                char character = input.charAt(index);
                                                if (character == '-') {
                                                        String morpheme =
input.substring(startIndex, index);
                                                        int morphemeLength = morpheme.length();
                                                        float position = (startIndex - dashes) /
(float)(wordLength - morphemeLength);
                                                        startIndex = index + 1;
                                                        for (int index2 = 0; index2 <
LEARN_REPETITIONS; index2++) {

        getter.addCandidateMorphemeData(morpheme, position);
                                                        }
                                                        dashes++;
                                                }
                                        }
                                }
                        }
                        if (breakAgain) {
                                break;
                        }
                        LearningSystem2(keyboard, getter, false, automatic);
                } while (false);
        }
}
```

APPENDIX B. CORPUS USED FOR EXPERIMENTATION

The corpus used for experimentation consists of over 640 spanish words that have been morphologically divided. Fifty words were chosen at random and used to perform the tests. They are listed at the end of this appendix. The remaining words, over 590, were used to train the system prior to running the tests. The list follows:

gat-it-o-s	cosmeto-logí-a	nutr-ición	com-e-ré
gat-it-a-s	sill-a	mal-dad	com-e-rás
gat-ot-a-s	sill-ón	seri-edad	com-e-rá
perr-it-o-s	in-ac-ción	rival-idad	com-e-re-mos
conej-it-o-s	inter-nacion-al	leal-tad	com-e-ré-is
elefant-it-o-s	sub-yac-e-nte	at-adura	com-e-rá-n
animal-it-o-s	sub-suel-o	viñ-edo	viv-i-rás
ch-ic-o	re-imprim-ir	arbol-eda	viv-i-re-mos
ch-iquit-ic-a	re-con-stru-ir	libr-ería	viv-i-ré-is
super-ch-iquit-it-o	re-ac-ción	flor-ero	viv-i-rá-n
super-ch-ic-a	des-ment-ir	guant-era	cant-a-ría-s
super-pájar-o-s	des-alent-ar	rigid-ez	cant-a-ría
super-pajar-it-o-s	des-mont-ar	bell-eza	cant-a-ría-mos
super-duper-tortug-a-s	des-dicha	cortes-ía	cant-a-ría-is
super-duper-caball-o-s	re-introduc-ir	atlet-ismo	cant-a-ría-n
super-ultra-yegu-a	introduc-ción	art-ista	com-e-ría
super-ultra-giraf-a	sub-conscien-te	lent-itud	com-e-ría-s
ultra-soni-d-o	sub-secuen-te	llama-miento	com-e-ría-mos
ultra-efect-iv-o	sub-sigu-ie-n-te	jura-mento	com-e-ría-is
casa-s	sub-estim-ar	profes-or	com-e-ría-n
café-s	in-decis-o	dulz-ura	viv-i-ría-s
pared-es	in-discut-id-o	barril-et-e	viv-i-ría
reloj-es	in-co-heren-te	histori-et-a	viv-i-ría-is
ley-es	im-bu-ír	maestr-ic-o	cant-e-s
rey-es	in-satisf-ech-o	duend-ecic-o	cant-e
rubí-es	in-saci-a-ble	chiqu-ill-o	cant-e-mos
tabú-es	imagin-a-ble	amor-cill-o	cant-é-is
análisis	im-pens-a-ble	red-ecill-a	cant-e-n
crisis	in-cont-a-ble	madr-in-a	com-a-s
déficit	contra-indic-ad-o	camp-iñ-a	com-a
salta-monte-s	contra-cep-tiv-o	cas-it-a	com-a-mos
superávit	contra-dic-ción	terron-cit-o	com-á-is
extend-er	contra-puest-a	luc-ecit-a	com-a-n
extens-ión	contra-posici-ón	chic-uel-o	viv-a
imprim-ir	inter-cambi-ar	ca-zuel-a	viv-a-s
impres-ión	inter-comunic-a-ción	perr-az-o	viv-a-mos
eléctric-o	inter-relacion-ad-o	mujer-on-a	viv-á-is
electric-idad	abol-i-cion-ista	mach-ot-e	viv-a-n
electric-ista	public-ista	pajarr-ac-o	cant-a-ra-s
permit-ir	cuchar-ón	popul-ach-o	cant-a-se-s
permis-iv-o	caj-ón	mig-aj-a	cant-a-ra
trans-atlántic-o-s	lagrim-ón	poet-astr-a	cant-a-se

rápida-mente
feliz-mente
lenta-mente
nueva-mente
actr-iz
emperatr-iz
habl-a-dor
habl-a-dor-a
habl-ar
habl-ar-se
habl-ar-le
habl-ad-o
habl-a-d-a
habl-a-ndo
habl-o
habl-a-s
habl-a
habl-a-mos
habl-á-is
habl-a-n
observ-a-dor
observ-a-dor-a
observ-ar
observ-ar-se
observ-ar-le
observ-ad-o
observ-a-d-a
observ-a-ndo
observ-o
observ-a-s
observ-a
observ-a-mos
observ-á-is
observ-a-n
beb-e-dor
beb-e-dor-a
beb-er
beb-er-se
beb-er-le
beb-id-o
beb-i-d-a
beb-ie-ndo
beb-o
beb-e-s
beb-e
beb-e-mos
beb-é-is
cos-e-dor
cos-er
cos-er-se
cos-er-le
cos-id-o
cos-i-d-a
cos-ie-ndo
cos-o
cos-e-s
cos-e
cos-e-mos
cos-é-is
cos-e-n
repart-i-dor
repart-i-dor-a
trans-géner-o
trans-vers-al
trans-sex-u-al
trans-curs-o
trans-port-ar
sobre-estimul-a-da
sobre-maner-a
sobre-mes-a
pre-dich-o
pre-conceb-id-o
pre-vist-o
semi-coc-id-o
semi-educ-a-tiv-o
semi-instruc-tiv-o
semi-curios-a
caprich-os-o
cautel-os-o
gelatin-os-o
respet-u-os-o
a-sintomá-t-ic-o
a-fónic-o
a-soci-al
a-ton-al
ex-presid-ente
ex-novi-o
ex-comulg-ar
vice-decan-o
vice-secretari-a
extra-pol-ar
extra-fin-o
extra-larg-o
retro-visor
retro-aliment-a-ción
retro-act-iv-o
retro-tra-er
america-n-iz-ar
ridicul-iz-ar
mone-t-iz-ar
memor-iz-ar
amen-iz-ar
nuestr-a-s
im-pre-vist-a
sub-mar-in-a
re-analiz-ad-o
des-leal-mente
im-posib-le
real
i-rreal
canción
cancion-cit-a
café
cafe-cit-o
flor-e-cit-a
buen-o
mejor
ten-í-a
tuv-e
agu-a
acu-os-o
duerm-o
dorm-i-mos
árbol
calent-orr-o
mujer-uc-a
cas-uch-a
gent-uz-a
vener-able
toler-able
acces-ible
color-ad-a
grup-al
edit-or-i-al
equival-ente
durm-iente
human-it-ari-a
eleccion-ari-o
perece-der-a
quebr-ad-iz-a
hu-idiz-a
pesqu-er-o
azucar-er-a
picar-esc-a
detectiv-esc-a
hambr-ient-o
harap-ient-o
varon-il
estudiant-il
estad-ista
box-ístic-o
patr-ístic-a
paj-iz-o
quebr-ad-iz-o
espum-os-o
honr-os-a
borr-os-o
pel-ud-o
panz-ud-o
austri-ac-o
pol-ac-a
eslov-ac-o
peru-an-a
bolivi-an-a
moneg-asc-o
chil-en-o
asunc-en-a
nacianc-en-o
cartagin-ense
bonaer-ense
flumin-ense
puertorriqu-eñ-o
madril-eñ-a
caraqu-eñ-o
franc-es-a
ingl-és
portugu-es-a
israel-í
marroqu-í
iran-í
argent-in-o
salmant-in-a
vietnam-ita
isreal-ita
selen-ita
yuca-tec-o
cant-á-ra-mos
cant-a-ra-is
cant-a-se-is
cant-a-ra-n
cant-a-se-n
com-ie-ra
com-ie-se
com-ie-ra-s
com-ie-se-s
com-ié-ra-mos
com-ié-se-mos
com-ie-ra-is
com-ie-se-is
com-ie-ra-n
com-ie-se-n
viv-ie-se
viv-ie-ra-s
viv-ie-se-s
viv-ie-ra
viv-ié-ra-mos
viv-ié-se-mos
viv-ie-ra-is
viv-ie-se-is
viv-ie-ra-n
viv-ie-se-n
cant-a-re-s
cant-a-re
cant-a-re-n
com-ie-re
com-ie-re-s
com-ié-re-mos
com-ie-re-is
com-ie-re-n
viv-ie-re
viv-ie-re-s
viv-ié-re-mos
viv-ie-re-is
cant-a
cant-ad
com-e
com-ed
viv-e
viv-id
usted
usted-es
super-mercad-o
deficit-ari-o
ajedrez
deficien-te
deficien-cia
gent-í-o
pond-ré
pond-ría
excelent-ísim-o
hermos-ísim-o
bell-ísim-o
alt-ísim-o
barat-ísim-o
pro-mov-er
pro-cre-ar
super-intendente
morf-o-logí-a

repart-ir
repart-ir-le
repart-id-o
repart-i-d-a
repart-ie-ndo
repart-o
repart-e-s
repart-e
repart-i-mos
repart-é-is
repart-e-n
recib-i-dor
recib-i-dor-a
recib-ir
recib-ir-se
recib-ir-le
recib-i-d-a
recib-ie-ndo
recib-o
recib-e-s
recib-e
recib-i-mos
recib-e-n
cuénta-se-lo
am-ar-se
viéndo-la
dí-se-lo
reloj-er-o
mes-er-a
anti-soci-al
univers-al-ista
univers-o
uni-cicl-o
uni-later-al
uni-celul-ar
uni-corni-o
partid-ista
anti-repet-itiv-o
institu-ción
parti-ción
institu-cion-al-es
anti-bió-tic-o-s
bio-logí-a
bio-físic-a
bio-ritm-o
arbór-e-o
hag-o
hic-e
tierr-a
terr-estre
fuerte
fort-ísim-o
v-oy
fu-i
mal-o
peor
jardin-er-a
am-á-ba-mos
contra-produc-ente
honr-os-o
encant-a-dor-a
traduc-ción-es
con-ti-go
pro-pus-e
re-form-a-dor-es
econom-ista-s
intencion-al-mente
re-produc-ción
inter-nacion-al-es
sub-atóm-ic-o
novat-ada
estoc-ada
sal-ida
decan-ato
lav-ad-o
encend-id-o
ronqu-id-o
aprend-iz-aje
a-terr-iz-aje
cafet-al
palm-ar
viol-encia
deline-ante
presid-ente
sirvi-ente
botic-ari-o
biblio-tec-ari-a
vist-azo
reduc-ción
abdic-ación
guatemal-tec-a
norueg-o
polinesi-o
uruguay-a
rus-a
amarill-it-o
bonit-ill-o
ric-ach-ón
bon-it-ísim-a-s
cant-a-ba
cant-a-ba-s
cant-á-ba-mos
cant-a-ba-is
cant-a-ba-n
com-í-a
com-í-a-mos
com-í-a-is
com-í-a-n
viv-í-a
viv-í-a-s
viv-í-a-mos
viv-í-a-is
viv-í-a-n
cant-é
cant-aste
cant-ó
cant-a-mos
cant-aste-is
cant-aro-n
com-í
com-ió
com-i-mos
com-iste-is
com-iero-n
viv-í
viv-iste
viv-ió
viv-i-mos
viv-iste-is
viv-iero-n
cant-a-ré
cant-a-rás
cant-a-rá
cant-a-réis
cant-a-rá-n
consider-a-ción
robust-o
marci-an-o
mar-in-o
ós-e-o
ultra-mar
higién-ic-o
serv-i-ci-al
sever-ísim-o
anti-cuerp-o
continent-al
inter-continent-al
arbol-ed-a
dent-al
dent-ríf-ic-o
artíst-ic-o
contest-a-ción
orto-grafí-a
carto-grafí-a
hidrául-ic-o
person-ific-a-ción
re-accion-ar
exhaust-iv-o
fosil-iz-a-ción
fosil-iz-ad-o
im-par
procrastin-ar
juici-os-o
bosc-os-ísim-a
dens-ísim-o
ultra-nacion-al-ista
pint-a-ría
estacion-ar-se
estacion-ó
termin-a-ría
termin-ó
matricul-ar-se
circum-naveg-ó
des-alent-a-ba
emocion-ad-o
control-a-dor
absurda-mente
aboli-cion-ista

The 50 words that were randomly selected in order to perform tests are listed below:

gat-ot-es
ultra-fin-o
ultra-conserv-a-dor
taxi-s
beb-e-n
cos-e-dor-a
repart-ir-se
recib-id-o
recib-í-s
cuént-a-le
sill-on-cit-o
in-interrump-id-o
re-inscrib-i-r
in-act-iv-o
sobre-puest-a
pre-acuerd-o
ex-novi-a
vice-presid-ente
vice-decan-ato
extra-curricul-ar
anti-constitu-cion-al-es
observ-a-rá-n
extension-al-i-dad
princip-ado
toler-ancia
bien-aventur-anza
in-certi-dumbre
torc-edura
cabez-ot-a
pan-ezot-e
vill-orri-o
colombi-an-o
vizca-ín-a
tepoz-tec-o
com-í-a-s
com-iste
cant-a-re-mos
viv-i-ré
viv-i-rá
viv-i-ría-mos
viv-i-ría-n
cant-á-se-mos
cant-á-re-mos
cant-a-re-is
viv-ie-re-n
libr-ería
cas-erío
jug-a-ré
super-estad-ista-s
escog-e-ría

APPENDIX C. MAXIMUM DISORDER SCORES

This program, written in Java, produces a table such that the amount of segment pairs are shown in the left column, starting at 1, and the maximum disorder score for the given amount of segment pairs is shown in the right column. The program stops due to excessive memory usage when it is going to compute the maximum disorder score of 11 segment pairs, because the program computes all possible permutations of length n, where n is the amount of segment pairs.

```
package maximumDisorderScores;

import java.util.ArrayList;

public class MaximumDisorderScores {
	public static String getOrderedString(int size) {
		//Assumes size >= 1
		String result = "";
		for (int index = 1; index <= size; index++) {
			result += Character.valueOf((char)(index + 64));
		}
		return result;
	}
	public static ArrayList<String> getAllStringPermutations(String source) {
		//Assumes source.length() >= 1
		ArrayList<String> result = new ArrayList<String>();
		if (source.length() == 1) {
			result.add(source);
		} else {
			for (int index = 0; index < source.length(); index++) {
				Character character = source.charAt(index);
				String left = source.substring(0, index);
```

```
                        String right = source.substring(index + 1, source.length());
                        String remainder = left + right;
                        ArrayList<String> remainderPermutations =
                                getAllStringPermutations(remainder);
                        for (String permutation: remainderPermutations) {
                                result.add(character + permutation);
                        }
                }
        }
        return result;
}
public static int getDisorderScore(String s1, String s2) {
        //Assumes s1.length() == s2.length()
        int Result = 0;
        for (int index1 = 0; index1 < s1.length(); index1++) {
                Character character = s1.charAt(index1);
                int index2 = s2.indexOf(character);
                Result += Math.abs(index1 - index2);
        }
        return Result;
}
public static int getMaximumDisorderScore(int subsegmentPairs) {
        //Assumes subsegmentPairs >= 1
        String source = getOrderedString(subsegmentPairs);
        ArrayList<String> permutations = getAllStringPermutations(source);
        int result = 0;
        for (String permutation: permutations) {
                int score = getDisorderScore(source, permutation);
                if (score > result) {
                        result = score;
                }
        }
        return result;
}
public static void main(String[] args) {
        int index = 1;
        while (true) {
                int score = getMaximumDisorderScore(index);
                System.out.println(index + " --> " + score);
```

```
                        index++;
                }
        }
}
```

REFERENCES

[1] O'Grady, William; Archibald, John; Aronoff, Mark; Rees-Miller Janie. **Contemporary Linguistics: An Introduction. 5th edition.** *Bedford/St. Martin's.* 2005.

[2] Tchoukalov, Tzvetan; Monson, Christian; Roark, Brian. **Morphological Analysis by Multiple Sequence Alignment.** *Stanford University; Center for Spoken Language Understanding, Oregon Health & Science University.* 2010.

[3] Alers Valentín, Hilton. **Morfología.** *Mayagüez Campus, Puerto Rico University, Puerto Rico.* http://ece.uprm.edu/docs/Morfolog%C3%ADa%20%28Dr.%20Hilton%20Alers%20Valent%C3%ADn%29.pdf

[4] Honrado, Asunción; Leon, Ruben; O'Donnel, Ruairi; Sinclair, Duncan. **A Word Stemming Algorithm for the Spanish Language.** *Universidad Autónoma de Madrid. University of Strathclyde.* 2000.

[5] Smirnov, Ilia. **Overview of Stemming Algorithms.** *DePaul University.* 2008.

[6] Willett, Peter. **The Porter Stemming Algorithm: Then and Now.** *University of Sheffield.* 2006.

[7] van den Bosch, Antal; Daelemans, Walter. **Memory-Based Morphological Analysis.** *ILK / Computational Linguistics, Tilburg University.* 1999.

[8] Ak, Koray; Taner Yıldız, Olcay. **Unsupervised Morphological Analysis Using Tries.** *Dept. of Computer Science and Engineering, Işık University, Turkey.* 2011.

[9] van den Bosch, Antal; Daelemans, Walter; Weijters, Ton. **Morphological Analysis as Classification: an Inductive-Learning Approach.** *Dept. of Computer Science, University of Limburg; Computational Linguistics, Tilburg University.* 1996.

[10] Internet Archive. **Google Help Center.** 2006. http://web.archive.org/web/20060322142522/http://google.com/support/bin/static.py?page=searchguides.html&ctx=basics

[11] de Gispert, A.; Mariño, J.B. **On the Impact of Morphology in English to Spanish Statistical MT.** *Universitat Politècnica de Catalunya. Spain.* 2008.

[12] Tang, Xuri. **English Morphological Analysis with Machine-learned Rules.** *Dept. Foreign Languages, Wuhan University of Science and Engineering, China.* 2006.

[13] Tzoukermann, Evelyne; Liberman, Mark Y. **A Finite-State Morphological Processor for Spanish.** *AT&T Bell Laboratories.* 1990.

[14] Velásquez, Francisco; Gelbukh, Alexander; Sidorov, Grigori. **AGME: Un Sistema de Análisis y Generación de la Morfología del Español.** *Centro de Investigación en Computación, Instituto Politécnico Nacional, México.* 2006.

[15] Moreno, Antonio; Goñi, José M. **GRAMPAL: A Morphological Processor for Spanish Implemented in Prolog.** *Universidad Autónoma de Madrid. Universidad Politécnica de Madrid.* 1995.

[16] Zapata, Carlos Mario; Edison Mesa, Jhon. **Una Propuesta para el Análisis Morfológico de Verbos del Español.** *Universidad Nacional de Colombia.* 2008.

[17] Al-Sughaiyer, Imad A.; Al-Kharashi, Ibrahim A. **Arabic Morphological Analysis Techniques: A Comprehensive Survey.** *Computer and Electronics Research Institute, King Abdul Aziz City for Science and Technology, Saudi Arabia.* 2003.

[18] Monson, Christian; Carbonell, Jaime; Lavie, Alon; Levin, Lori. **ParaMor: Minimally Supervised Induction of Paradigm Structure and Morphological Analysis.** *Carnegie Mellon University.* 2007.

[19] Haupt, Randy L.; Haupt, Sue Ellen. **Practical Genetic Algorithms.** *Wiley-Interscience.* 1998.

[20] Russell, Stuart; Norvig, Peter. **Artificial Intelligence: A Modern Approach.** *Prentice Hall.* 1995.

[21] Tiedemann, Jörg. **Automatic Construction of Weighted String Similarity Measures.** *Department of Linguistics, Uppsala University.* 1999.

[22] Lopresti, Daniel; Tomkins, Andrew. **Block Edit Models for Approximate String Matching.** *Panasonic Technologies, Inc.* 1996.

[23] Mackay, Wesley; Kondrak, Grzegorz. **Computing Word Similarity and Identifying Cognates with Pair Hidden Markov Models.** Page 40. *Department of Computing Science, University of Alberta, Canada.* 2005.

[24] Gelbukh, Alexander; Alexandrov, Mikhail; Han, SangYong. **Detecting Inflection Patterns in Natural Language by Minimization of Morphological Model.** *National Polytechnic Institute, Mexico; Chung-Ang University, Korea.* 2004.

[25] Kazakov, Dimitar. **Unsupervised Learning of Naïve Morphology with Genetic Algorithms.** *Czech Technical University, Prague.* 1997.

[26] Kazakov, Dimitar; Manandhar, Suresh. **A Hybrid Approach to Word Segmentation.** *University of York, Heslington, York.* 1998.

[27] Grice, J. Alicia; Hughey, Richard; Speck, Don. **Reduced Space Sequence Alignment.** *CABIOS, Oxford University Press.* 1997.

[28] Morgenstern, Burkhard; Dress, Andreas; Werner, Thomas. **Multiple DNA and Protein Sequence Alignment Based on Segment-to-Segment Comparison.** *GSF, National Research Center for Environment and Health, Institute of Mammalian Genetics; Fakultät für Mathematik, Universität Bielefeld. Germany.* 1996.

[29] Beasley, David; Bull, David R.; Martin, Ralph R. **An Overview of Genetic Algorithms: Part 2, Research Topics.** *University of Wales College of Cardiff; University of Bristol. United Kingdom.* 1993.