An Efficient Autocorrect Algorithm Using Genetic Algorithm

Sol Kim

Farragut High School

ABSTRACT

Autocorrect is used widely in digital typing to, as the name suggests, automatically correct mistyped words. The purpose of this paper is to investigate what factors should contribute to how a mistyped word is corrected and how significant each factor is in the correction process. First, the aspects that could contribute to a more efficient autocorrect algorithm include how different a mistyped word is from the intended word, how far away a mistyped character is from the correct one, and how the typo is classified. These factors were then quantified and integrated into the autocorrect algorithm to see how accurately a corpus of words could be corrected. Finally, these quantities that were assigned were altered and experimented with to inspect how much they contributed to the accuracy of the algorithm.

1 INTRODUCTION

As the digitalization of typing has become more common, there has been more of an opportunity to use keyboards, and inevitably, typos have also become more common as a result. However, there are many instances in the modern world of devices being able to correct these mistakes, as essentially any electronic device is capable of doing so. The algorithm that is responsible for this autocorrection most likely calculates the minimum edit distance (MED) between the typo and a known word and corrects the typo .

However, clearly the MED between two strings should not be the only factor to consider. Some additional factors that should be considered are the distance between two characters on a keyboard and common types of typos. Through the combination of these three factors, a more accurate autocorrect program can be made.

Design of the Paper

Section 2 discusses some traditional methods of calculating a metric for the MED between two strings, known as Levenshtein-Distance and its successor Damerau-Levenshtein Distance.

Section 3 is about improving these metrics in a practical sense by considering keyboard distance, types of typos, and how these factors contribute to the MED between two strings.

Section 4 analyzes the accuracy of these factors and which ones should be given more weight when deciding the MED and thus which word the typo should be correct.

2 TRADITIONAL METHODS OF CALCULATING THE MINIMUM EDIT DISTANCE

Minimum edit distance (MED) is defined as the minimum numbers of operations that are needed to transform one string into another. These operations include insertion, which is adding a character; deletion, which is deleting a character; and substitution, which is replacing a character with another. In some cases, substitution is counted as two separate operations, as a substitution is technically a combination of a deletion and insertion. The number of edits it takes to get from one string to another can also be called the cost; in other words, finding the MED can be thought of as finding the lowest cost to transform one string into another. There are two main approaches to finding the MED between two strings

2.1 The Naive Solution

As mentioned, there are three main operations when transforming one string into another. This so-called naive solution involves performing each operation on every single character within the string until the desired string is obtained. For example, suppose the string “aaa” is to be transformed into the string “abc”. A potential first step could be to insert, delete or substitute the first “a” in the initial string. Afterwards, the second step could involve inserting, deleting, or substituting a character from the three new strings. For shorter strings, this may not seem like a problematic approach; however, if the string was ten, twenty, or more than twenty characters long, then considering every single operation that could be done would take an extensive amount of time. For the aforementioned example, there are only 3^3=27 different chains of operations to consider, which would not take long for a computer to find the MED. The time it would take for a computer to compute the MED via this method exponentially increases by a factor of three per additional character, which means that if a string was ten characters long, 3^{10}= 59,049 different strings and their edit distance from the original string would have to be compared to each other. For this reason, an algorithm has been developed to effectively calculate the MED between two strings.

2.2 The Effective Solution

In order to find the MED between two strings effectively, the recursive function, , the edit distance between two strings of length and has been defined as the following:

$$D(i,j) = \min \begin{cases} D(i-1,j-1) + \begin{cases} 2 & \text{if } X(i) \neq Y(j) \\ 0 & \text{if } X(i) = Y(j) \end{cases} \\ D(i-1,j)+1 \\ D(i,j-1)+1 \end{cases}$$

Figure 1: Minimum edit distance between strings X and Y where is the th character of string and is the character of string

It should be noted that this recursive function defines substitutions as operations of cost 2. This recursive function dramatically decreases the time needed to calculate the MED between two strings, and For shorter strings, can be computed easily; between the two strings “aaa” and “abc”, it is clear to see that the minimum edit distance is 4. However, if the strings were longer, say “equation” and “inequality”, then the answer is not so obvious. To compute the MED between two strings, a table can be used and the aforementioned recursive function can be applied to this table.

2.3 Example Using the Recursive Function

For this example, the strings “equation” and “inequality” will be used. The characters will be put into a table as follows:

n	8										
o	7										
i	6										
t	5										
a	4										
u	3										
q	2										
e	1										
#	0	1	2	3	4	5	6	7	8	9	10
	#	**i**	**n**	**e**	**q**	**u**	**a**	**l**	**i**	**t**	**y**

In this table, “#” denotes an empty string, and the numbers represent the edit distance between the intersecting strings. For example, the intersection of the row with “#” and the column with “y” represents the MED between an empty string and the word “inequality”, which would be 10. The table can then be filled out according to the formula, filling in each box at a time.

n	8	7	6	7	8	7	6	7	6	7	8
o	7	6	7	8	7	6	5	6	5	6	7
i	6	5	6	7	6	5	4	5	4	5	6
t	5	6	7	6	5	4	3	4	5	4	5
a	4	5	6	5	4	3	2	3	4	5	6
u	3	4	5	4	3	2	3	4	5	6	7
q	2	3	4	3	2	3	4	5	6	7	8
e	1	2	3	2	3	4	5	6	7	8	9
#	0	1	2	3	4	5	6	7	8	9	10
	#	**i**	**n**	**e**	**q**	**u**	**a**	**l**	**i**	**t**	**y**

As shown, if substitutions were to have a cost of two, the MED between the words “equation” and “inequality” is 8. This method is much more efficient than considering 3^8 (not 3^{10} because 2 insertions/deletions are necessary to transform one into another) different edit distances and comparing them to find the smallest one.

2.4 Levenshtein Distance (LD)

One of the primary ways to calculate the MED between two strings is via the Levenshtein Distance. This method to calculate the MED was named after a mathematician named Vladimir Levenshtein. The edits that can be made on a string are the ones mentioned in the previous section: insertions, deletions, and substitutions. However, substitutions are considered to have a cost of 1 with the Levenshtein Distance metric. The LD of two strings, *a* and *b*, with lengths *i* and *j* respectively can be defined by $\mathrm{lev}_{a,b}(i,j)$, which is defined as the following:

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j)+1 \\ \mathrm{lev}_{a,b}(i,j-1)+1 \\ \mathrm{lev}_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} \end{cases}$$

Figure 2: Recursive formula of Levenshtein Distance

It should be noted that this recursive function is very similar to the previously defined $D(i, j)$, but instead of $D(i, j)$ describing substitutions as having a cost of 0 or 2, this function describes substitutions as having cost of 0 or 1. The subscript next to the 1 in the scenario describing substitutions is a condition that states that the cost of a substitution is 1 if the *i*-th character of string *a* is not equal to the *j*-th character of string *b*.

2.5 Damerau-Levenshtein Distance (DLD)

Although the LD metric of MED between two strings is effective, there is a slight flaw with the algorithm. Consider the word “background”. A possible misspelling of this word could be “backrgound” if the person typing were to make a mistake. However, the minimum edit distance between the strings “background” and “backrgound” would be 2, as two substitutions would have to be made, despite the minor mistake of switching two letters of an otherwise correctly typed word. This is the main advantage that Damerau-Levenshtein Distance has over the classic Levenshtein Distance. The DLD metric of MED takes into consideration another type of operation that can be performed between two strings: transposition, or swapping two adjacent characters with each other. Thus, to calculate the DLD between two strings, the recursive function $d_{a,\,b}(i, j)$, where strings a and b have lengths i and j respectively, is defined as follows:

$$\begin{array}{ll} & i = j = 0 \\ j) + 1 & i > 0 \\ 1) + 1 & j > 0 \\ j - 1) + 1_{(a_i \neq b_j)} & i, j > 0 \\ j - 2) + 1 & i, j > 0 \text{ and } a[i] = b[j-1] \text{ and } a[i-1] = \end{array}$$

Figure 3: Recursive formula of Damerau-Levenshtein Distance

The fifth scenario of the function represents transposition, as $a[i]$ denotes the i-th character of the string a. This is clearly the same as the previously defined $\text{lev}_{a,\,b}(i, j)$, except for the fact that there is an additional function that represents transposition. This allows computers to calculate the MED between two strings in a more realistic matter, as people are prone to making the mistake switching two characters within a word while typing.

3. OUR METHODS

In order to create an efficient autocorrect algorithm, the smallest DLD should not be the only factor that contributes to how a word is corrected. A typo may be close to a word in terms of DLD, but it may not be the intended word of the person typing. In other words, simply considering DLD is unrealistic, as there are other factors to take into consideration, which will be discussed in the following sections.

3.1. Different Types of Common Typos

According to Roger Mitton’s *English Spelling and the Computer*, many of the typos made by humans today can be characterized by one or more of the four types of errors that the Damerau-Levenshtein Distance describes. The frequency of each error can be seen in the table below. By taking these statistics into consideration, an autocorrect algorithm can be made more realistic. The cost of an edit can be adjusted based on how frequently it is made based on this table. For example, if a letter was omitted, which translates to an insertion, the cost may be reduced to 0.5 because of how common it is, as opposed to transposition, which may be given an increased cost of 1.5 because of how rare it is compared to an insertion. Even within these four types of errors, there are specific cases of each that have to be taken into consideration as well.

3.1.1 Omissions

For omissions (typos that require an insertion), Mitton presents the observation that “keys struck by the little fingers are more likely to be omitted”, so some edits on the QWERTY layout such as an insertion of the letter A or Q should be given an even further reduced cost because of how frequently it happens within insertion errors. Other observations regarding omission errors include the fact that the first letter of words is rarely omitted and that letters that are repeated within a word are also likely to be omitted.

3.1.2 Substitutions

Similar patterns have also been observed and presented by Mitton within substitutions. The figure below shows the keys that have been frequently typed in the place of the letter D. As stated by Mitton, “The amount of shading in a box represents the number of times that that key was typed in place of “*d*.” It can be seen that it is very likely that someone types S, E, F or C instead of D because of how close they are on the keyboard. The reason why K is also frequently typed in the place of D is because it is typed by the same finger on a different hand. Therefore, while letters that are far away on a keyboard should be given a higher cost because of how unlikely it is that they will be typed, letters that are typed by the same finger on a different hand should not be penalized as heavily.

3.1.3 Transpositions

The final type of error is transpositions. Mitton states that “eighty per cent or more [errors] involve keys typed with different hands.” It is also stated that shorter words such as “the” and “that” are especially affected, since the letters T and H are typed with different hands. The reason being is because these shorter words are typed rather quickly, and as a result, the person typing may be tempted to type both letters at the same time, which commonly translates to the two letters being transposed. Additionally, Mitton states that the videos of transposition errors demonstrate that the second finger usually gets to the key faster than the first finger can get to its key.

3.2 Our model

Using this information, a realistic autocorrect algorithm can be formulated by making uncommon mistakes cost more than their common counterparts. It can be recalled that the traditional Damerau-Levenshtein Distance calls for every edit - insertion, deletion, substitution, and transposition - to have a cost of one. However, based on statistics given by Mitton and the proposition before, the edits that need to be penalized most heavily from greatest to least are transposition, substitution, deletion, and insertion. Within these edits, however, the cost of each edit should increase based on how far away the intended key is on a traditional QWERTY keyboard. For example, the string "hpt" is more likely to be intended as "hot" rather than "hat". If the distance between characters on a keyboard were not considered, then the editing cost between "hpt" and "hat" or "hot" would be the same, which would therefore make the algorithm inaccurate. Of course, this cost would be adjusted according to the information presented by Mitton above. An instance of this is presented in *English Spelling and the Computer*, where the letter K should not be penalized as heavily as typing the letter P when the intention was to type the letter D.

3.3 Genetic Algorithm

Finding the optimal cost of each edit through brute force would be ridiculously time-consuming, as there are simply too many combinations of values to test. This is where the genetic algorithm comes into play. A genetic algorithm is essentially a search heuristic that is inspired by, as its name suggests, genetics and the theory of evolution. There are five main phases that determine a genetic algorithm.

The first of these phases is the initialization of a population. Each member of the population has a specific set of parameters or genes that constitute its genotype. For the purpose of this project, a member of the population would be a set of costs for each edit, with each cost corresponding to a gene.

After the population has been initialized, the next step is to assess the fitness of the population. This is essentially where each member of the population is evaluated to see how well the parameters solve the problem at hand. The fitness is then quantified using a function of some sort. To determine the fitness of a set of four costs, the percentage of words of a corpus that have been properly corrected can be used to assess the fitness of each member of the population.

Once the fitness of each member has been calculated, the two members of the population that have the highest fitness score are selected. This would correspond to selecting the two sets of costs that would result in the highest percentage of words corrected.

Afterwards, the most important phase, the crossover phase, is when the two fittest members' genes are swapped. This occurs by picking a random "crossover point" within the genes and switching the genes past or before that crossover point, making two "children" with characteristics of the "parents." In the case of this project, the genes that would be exchanged between two sets would be their costs. For example, if there were two sets of costs, {1, 2, 3, 4} and {5, 6, 7, 8}, and the crossover point were to be between the second and third costs, then the two new members would be {1, 2, 7, 8} and {5, 6, 3, 4}.

Finally, there is the mutation phase. This is when the genes of a child are given a chance of developing a random gene that is not inherited from either of the parents. Going back to the example given above, a mutation of one of the children may be {1, 2, 9, 8} instead of {1, 2, 7, 8}. This essentially offers a constant variation as the population grows, since eventually the majority of the expanded population would be the fittest of the initial population.

4. EXPERIMENTS

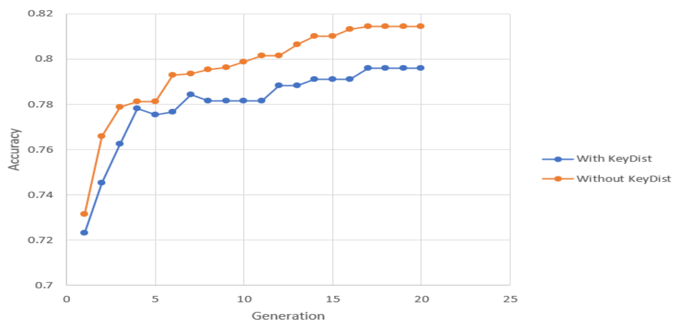

Figure 4: Accuracy of the autocorrect algorithm using genetic algorithms

The above graph demonstrates how the accuracy of the autocorrect algorithm evolved with each generation, where "KeyDist" refers to the algorithm that calculated the physical distance between two keys on a keyboard. Surprisingly, accounting for the distance between two keys harmed the accuracy of the autocorrect algorithm, as the difference between the two accuracies by the twentieth generation is about 1.8 percent.

Generation(s)	a	b	c	d	Accuracy (%)
12-13	0.4	1	0.9	2.2	78.8254
14-16	0.5	1	0.8	2.5	79.1156
17-20	0.5	1	0.8	2	79.6105

Table 1: Last three distinct accuracies that the autocorrect algorithm produced with KeyDist

This table shows the last three distinct accuracies that the autocorrect algorithm produced with the KeyDist algorithm included. “a” corresponds to the cost of transposition, “b” to insertion, “c” to deletion, and “d” to substitution. It can be seen that the cost transposition is kept the lowest; the costs of insertion and deletion are relatively similar, but deletion is slightly lower; and the cost of substitution is much higher than the rest of the edits.

Generation(s)	a	b	c	d	Accuracy (%)
14-15	0.8	1.2	1	1.2	80.9985
16	0.8	1	0.8	1	81.3105
17-20	0.5	1	0.8	1	81.4289

Table 2: Last three distinct accuracies that the autocorrect algorithm produced without KeyDist

This table shows the same statistics as the previous table but without the use of the KeyDist algorithm. Although similar trends can be seen among the majority of the costs, the cost of substitution is seen to be substantially lower without the use of KeyDist.

From the data above that depicts the accuracy of the autocorrect algorithm with and without the consideration of the physical distance between keys, several conclusions can be drawn.

Firstly, it can be seen that the most frequent edit is surprisingly transposition; then deletion; and finally, depending on the inclusion of the KeyDist algorithm or not, insertion then substitution. The deletions can be attributed to unintentionally pressing a key while typing a word. A possible reason that KeyDist may have resulted in lower accuracies is because of the lack of repeated typos in the corpus, which means that there were not a lot of typos in which KeyDist was relevant enough to consider.

5 CONCLUSION

Using what started as brute force and evolved into the genetic algorithm, it was found that the most efficient set of costs for transposition, insertion, deletion, and substitution were 0.5, 1, 0.8, and 1.2, respectively.

Blindly testing different combinations of costs for each edit took an absurd amount of time — about a month of running code if each trial took the same amount of time — to determine the most accurate ones. Thus, a genetic algorithm was implemented to not only look for a more accurate set of costs, but to look for that set in a shorter amount of time.

With these costs that were generated from the genetic algorithm, the accuracy of the autocorrect algorithm turned out to be about 81.4%.

It turns out that considering the physical distance between keys on a keyboard led to more inaccuracy than ignoring it. This may be attributed to the types of typos that were present on the corpus used to test this autocorrect algorithm.

It would be interesting to look into some different factors that contribute to inaccurate typing that were not discussed and try to implement those into the algorithm. Clearly, there is exists a much more efficient algorithm, as almost all mobile devices and computers are able to correct typos in an instant.

REFERENCES

1. Mitton, R. (1996). Slips and Typos. In *English spelling and the computer*. London: Longman.

2. Minimum Edit Distance. Stanford. URL https://web.stanford.edu/class/cs124/lec/med.pdf

3. Damerau–Levenshtein distance. (2020, September 13). URL https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance

4. Mallawaarachchi, V. (2020, March 01). Introduction to Genetic Algorithms URL https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3