Noisy Substring Problem

Implementation and Applications

Seth Banuach
Computer Science
University of Kentucky
Lexington, KY USA
Seth.Banuach@uky.edu

Jason Tran
Computer Science
University of Kentucky
Lexington, KY USA
Jason.Tran@uky.edu

ABSTRACT

The goal of this paper is to describe the process of implementing the Noisy Substring Algorithm presented by R. L. Kashyap and B. J. Oommen in 1983 [1]. The problem description is as follows: given a dictionary *H*, let *T(U)* be the set of words which contains *U* as a substring. The goal is to estimate *T(U)* when *U* is not known, but a noisy version of *U* is given as *Y*. The set estimate proposed by the authors *S*(Y)* of *T(U)* is a proper subset of *H* such that every element contains at least one substring which resembles Y most according to the Levenshtein metric. *S*(Y)* can be computed in cubic time. Another estimate, $S^M(Y)$, is also suggested by the authors, which runs in drastically less time with only slightly less accuracy. We confirmed the results of the original authors and tested the algorithm under non-ideal scenarios.

1 Introduction

The problem of noisy substring matching can be compared to the problem of exact substring matching. Many solutions already exist for exact substring matching, with most algorithms running in sub-linear time. However, in many cases, the string is presented in a noisy way. Running an exact substring matching algorithm on a noisy substring may return entirely useless information or no information at all about the substring searched. Therefore, we would like to use the noisy substring searching algorithm.

This algorithm has many potential applications. For example, with file editing, the user may want to search the file in a particular way that involves some close matching of a substring. Noisy substring matching could also be extended to searching for occurrences of sensitive or otherwise important information in large data. String searching algorithms have also generally found many uses in computational biology, so it is not unlikely that noisy substring matching could be used there.

1.1 Definition

Let *T(U)* be the set of words which contains *U* as a substring. The goal is to estimate *T(U)* when *U* is not known, but a noisy version of *U* is given as *Y*. Substitution, insertion, and deletion errors transform *U* into *Y*.

For this problem, ideally, the set estimate *S(Y)* should contain *T(U)*. Therefore, the quality of the estimate *S(Y)* is measured by the frequency of *S(Y)* containing *T(U)*. However, the set estimate *S(Y)* should also be as small as possible, such that it is actually a meaningful estimate of *T(U)* (p. 141).

Two set estimates will be considered. *S*(Y)*, is computed via comparing all contiguous substrings of X with the noisy substring Y using an appropriate measure of dissimilarity. $S^M(Y)$ is a slight modification to the same process. For the algorithms presented in this paper, the Levenshtein Distance (LD) is used. Other measures of dissimilarity could easily be substituted, according to the authors (p. 141).

The advantage of using LD as a measure of dissimilarity is that, depending on the application of the algorithm, it is easy to modify the weights of each kind of edit in order to make the algorithm much more accurate. In many cases, if a probability distribution is known for the transformations which transform U into Y, a set estimate can be obtained which is exactly $T(U)$ (p. 142).

To compute $S^*(Y)$, a function $D^{|X|}(X, Y)$ is defined. This function takes strings $X \in H$ and Y and returns the minimum LD between all contiguous substrings of X and the noisy substring Y. This result can be computed in time proportional to $O(|X|^2 \cdot |Y|)$, so the set estimate $S^*(Y)$ can be computed in cubic time (p. 142).

2 Implementation

In order to compute the quantity $D^{|X|}(X, Y)$, the Wagner-Fischer (WF) algorithm is used. The WF algorithm runs in time proportional to $O(|X| \cdot |Y|)$. The algorithm computes the LD between X and Y, but as a byproduct also computes the *LD* between the prefixes of X and the prefixes of Y. These results are stored in a WF matrix. An example of a WF matrix is as follows:

		s	l	u	m	b	e	r
	0	1	2	3	4	5	6	7
n	1	1	2	3	4	5	6	7
u	2	2	2	2	3	4	5	6
m	3	3	3	3	2	3	4	5
b	4	4	4	4	3	2	3	4
e	5	5	5	5	4	3	2	3
r	6	6	6	6	5	4	3	2
s	7	6	7	7	6	5	4	3

Figure 1: Example WF matrix between X="slumber" and Y="numbers".

The LD between "slumber" and "numbers" is the bottom right cell, which is 3 in this case. However, as a byproduct of the dynamic method used in computing this matrix, each cell $WF_{(i,j)}$ in the matrix actually represents the LD between the prefix of X ending at the column i and the prefix of Y ending at the row j.

Therefore, when computing the LD between each substring of X and Y in $D^{|X|}(X, Y)$, we can use WF without needing to compute WF for every contiguous substring of X. Instead, we can compute WF for each suffix of X. This will compute the LD between each prefix of each suffix of X and the noisy substring Y. The minimum of these is what returned by $D^{|X|}(X, Y)$.

In order to analyze the complexity of this algorithm, let $|V|$ be a substring of X, and let $q=|V|$ and $M=|Y|$. $WF(V, Y)$ can be computed in exactly $q \cdot M$ symbol comparisons. Hence, the number of symbol comparisons needed to compute $D^{|X|}(X, Y)$ is:

$$\sum_{q=1}^{N} q \cdot M = M \cdot \left[\frac{N \cdot (N+1)}{2} \right]$$

Note that, for an average case where Y is relatively large, it is highly unlikely that any substring of X of very small size will have a minimal LD to Y. This is the assumption that the set $S^M(Y)$ is constructed from. Instead of computing the WF matrix between each suffix of X and Y, the WF matrix is only computed for suffixes of X where $|X| \geq |Y|$. Therefore, instead of q ranging from 1 to |X|, it will range from $N - k + 1$, where $k = max\,[|X| - M + 1, 1]$. Let this algorithm be $D^k(X, Y)$. The number of symbol comparisons needed to compute $D^k(X, Y)$ is:

$$\sum_{q=|X|-k+1}^{N} qM = M \cdot \left[\frac{|X|\cdot(|X|+1)}{2} - \frac{(|X|-k)\cdot(|X|-k+1)}{2}\right]$$

$$= M \cdot \left[|X|\cdot k - \frac{k\cdot(k-1)}{2}\right] \text{ (p. 143)}$$

$$= \frac{M}{2} \cdot [|X|\cdot(|X| + 1) - M\cdot(M - 1)] \text{ if } M < |X|$$

$$= M\cdot|X| \text{ if } M \geq |X|.$$

While $D^k(X, Y)$ also requires cubic computation time, due to the negative quadratic term dependent on M, as M grows toward $|X|$, the number of symbol comparison trends toward quadratic (p. 144). Therefore, $S^M(Y)$ is, in the limit, one order computationally less expensive than $S^*(Y)$.

2.1 Algorithm

Given functions d_delet(), d_insrt(), and d_subst(char a, char b) which return the cost of each respective edit operation, $D^k(X, Y)$ is implemented as follows (C++):

```
int k_dist(string x, string y, size_t k) {
   int minDist = INF;
   size_t n = x.size();
   size_t m = y.size();

   vector<vector<int>> p(
         n+1, vector<int>(m+1, 0));

   for (size_t i = 1; i <= n; i++)
      p[i][0] = p[i - 1][0] + d_delet();

   for (size_t j = 1; j <= m; j++)
      p[0][j] = p[0][j - 1] + d_insrt();

   for (size_t q = n - k + 1; q <= n; q++) {
      string v = x.substr(n - q, q);

      for (size_t i = 1; i <= q; i++) {
         for (size_t j = 1; j <= m; j++) {
            int r1 = p[i - 1][j - 1]
                  + d_subst(v[i - 1], y[j - 1]);
            int r2 = p[i][j - 1] + d_insrt();
            int r3 = p[i - 1][j] + d_delet();
            p[i][j] = min(r1, min(r2, r3));
         }
         minDist = min(minDist, p[i][m]);
      }
   }
   return minDist;
}
```

3 Experimental Results

The first goal in obtaining experimental results was to reproduce the results obtained by the authors. In order to match our test cases as closely as possible, the same methods were used for each experiment. Dictionaries were obtained from an online list of the 100,000 most common English words provided by h3xx in 2015 [2].

The original authors had a slightly different method for turning U into Y. First, they used a geometric distribution to determine the number of insertions to perform on U, and then deleted or inserted symbols using a predefined probability distribution. We used a geometric distribution, but instead used it to determine the total number of edits. The geometric distribution parameter was obtained from the authors' reported average number of errors. The type of error was assumed to be equally likely among all three types. The probability of a specific character occurring in a substitution or insertion was assumed to be equally likely among all lowercase characters.

The process for choosing noisy strings was therefore as follows: a random word was chosen from *H*. A random substring *U* was chosen from it. *U* was transformed into *Y* using the process described above. LD edit distances were 0 or 1.

We reproduced each of the tables provided by the author, except we replaced their data with our own experimental data, as follows:

Table I
Some Results From Table II, Experiment 3. In All Cases, $S^*(Y)$ Contains $T(U)$.

Word	*U*	*Y*	$\lvert S^*(Y)\rvert$
promised	romis	romiss	2
government	nment	nmenz	1
suddenly	sudde	sugde	1
provided	vided	videdh	1
greatest	reatest	retest	1
original	origina	otrgina	2
existence	stence	stenceb	1
american	merican	meican	1
themselves	hemse	hemset	1
original	origi	jrigi	1
although	lthoug	lthodqg	1

Table II
Results of the Five Experiments

Exp	Nstr	Dict	Min $\lvert U\rvert$	Est	Acc	Av. Size
I	200	H1	5	$S^*(Y)$	0.970	2.60
I	200	H1	5	$S^M(Y)$	0.965	2.56
II	200	H1	6	$S^*(Y)$	0.985	1.550
II	200	H1	6	$S^M(Y)$	0.980	1.575
III	500	H2	5	$S^*(Y)$	0.986	1.612
III	500	H2	5	$S^M(Y)$	0.984	1.566
IV	500	H2	6	$S^*(Y)$	0.988	1.318
IV	500	H2	6	$S^M(Y)$	0.986	1.352
V	500	H2	7	$S^*(Y)$	0.994	1.230
V	500	H2	7	$S^M(Y)$	0.992	1.216

For Table 2:

- Nstr: number of noisy strings used
- H1: dictionary of 292 words, length ≥ 7
- H2: dictionary of 166 words, length ≥ 6
- Min $|U|$: minimum length of substring *U*
- Acc: frequency of est containing $T(U)$
- Av. Size: average size of $S^*(Y)$ or $S^M(Y)$

For each experiment, the geometric parameter was derived from the author's reported average number of errors in their strings. The results are consistent with the original author's results. The singular exception is the average size of $T(U)$; oddly enough, the average size in our experiments was notably lower than the average size from the authors' experiments. This is most likely due to the convention in which the authors transformed *U* into *Y*. Because we don't know how exactly they chose whether to do an insertion or substitution or deletion, the average $|Y|$ could have varied considerably from ours. A smaller $|Y|$ would result in a much larger $T(U)$, since for smaller substrings there are many more potential matches in the noisy substring algorithm.

We also reproduced the data comparing the number of symbol comparisons between $S^*(Y)$ and $S^M(Y)$, using a dictionary of size 500 with words of length ≥ 7.

Table III
Comparison of Number of Symbol Comparisons Required In $S^*(Y)$ vs $S^M(Y)$

$\lvert Y\rvert$	$S^*(Y)$	$S^M(Y)$	$\lvert Y\rvert$	$S^*(Y)$	$S^M(Y)$
5	92125	67075	10	184250	43450
6	110550	65460	11	202675	45705
7	128975	55328	12	221100	48540
8	147400	48392	13	239525	52117
9	165825	44289	14	257950	56126

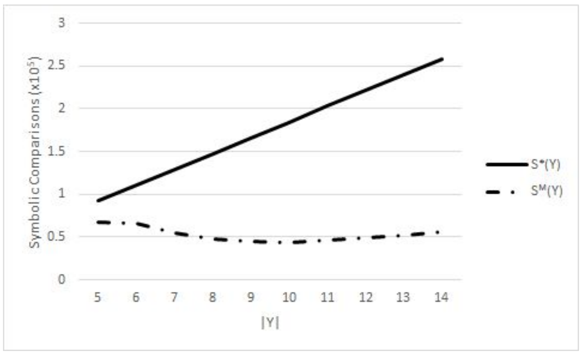

Figure 2: Graph of the number of symbol comparisons required to compute $S^*(Y)$ and $S^M(Y)$ versus $|Y|$. Dictionary size of 500, words of length ≥ 7.

Our data is consistent with their data and the equations from section 2. However, it is worth noting that the "dip" in our data is less steep than theirs. This is perhaps due to a greater number of longer words in our data compared to their data, since we sampled more words for our test, and the most common words tend to be shorter than less common words.

The other goal for our experiments was to see how the algorithm performs in non-ideal circumstances. This would specifically be when there are small strings in the dictionary and when the dictionary is much larger. The conditions of experiments I-V were repeated, with a few changes.

First, the distribution of word lengths obtained from the dictionary was obtained from a normal distribution with mode ≈ 7 and a left skew parameter which varied for each experiment. The equation for a skewed normal distribution is

$$\frac{2}{\omega\sqrt{2\pi}}e^{-\frac{(x-\xi)^2}{2\omega^2}}\int_{-\infty}^{\alpha\left(\frac{x-\xi}{\omega}\right)}\frac{1}{\sqrt{2\pi}}e^{-\frac{t^2}{2}}\,dt$$

where ω is the scale parameter, α is the shape parameter ($\alpha < 0$ for left skew, $\alpha > 0$ for right skew), and ξ is the location parameter. [3]

Also, the geometric distribution parameter was set to be $\frac{1}{1.4}$ (the authors' average number of errors across all experiments). The number of noisy substrings was set to be 1500 for all experiments. Only words between length 3 and 16 were allowed in the dictionary. Instead of setting $\min(|U|)$, $|U|$ was set. For each test in table IV, the average size of the estimate set was ≤ 5. The results are as follows:

Table IV
Results of Experiments with Lower Length Strings in the Dictionary (skewed normal)

Dict	|U|	Est	Acc
H1	5	$S^*(Y)$	0.954
H1	5	$S^M(Y)$	0.920
H1	6	$S^*(Y)$	0.983
H1	6	$S^M(Y)$	0.957
H1	7	$S^*(Y)$	0.992
H1	7	$S^M(Y)$	0.980
H2	5	$S^*(Y)$	0.952
H2	5	$S^M(Y)$	0.926
H2	6	$S^*(Y)$	0.986
H2	6	$S^M(Y)$	0.963
H2	7	$S^*(Y)$	0.991
H2	7	$S^M(Y)$	0.977
H3	5	$S^*(Y)$	0.969
H3	5	$S^M(Y)$	0.941
H3	6	$S^*(Y)$	0.981
H3	6	$S^M(Y)$	0.962
H3	7	$S^*(Y)$	0.993
H3	7	$S^M(Y)$	0.977

H1: $\omega = 6$, $\alpha = -3$, $\xi = 10.8$
H2: $\omega = 8$, $\alpha = -4$, $\xi = 11.3$
H3: $\omega = 10$, $\alpha = -5$, $\xi = 11.8$

The skew equation for each dictionary was multiplied by a constant factor of 500.

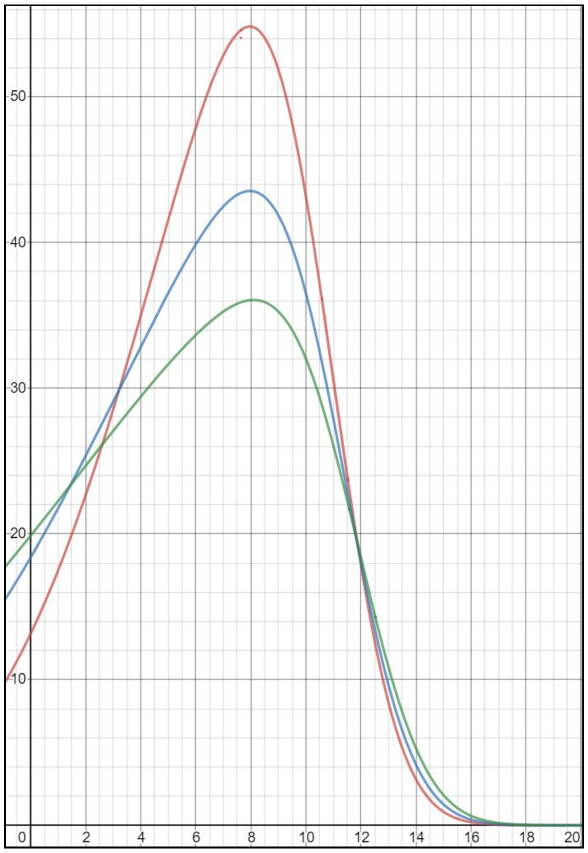

Figure 3: Graph of H1, H2, and H3 skewed normal distribution of word lengths. H1 has the highest peak; H3 has the lowest peak. [4]

From the table, it does not appear that there is a very significant difference between the mean frequencies when testing *H1, H2,* and *H3*. This is quite a surprise, because one would intuitively expect that a greater ratio of smaller words to larger words would lead to a higher inaccuracy. This is because it would greatly increase the number of contiguous substrings in *H* while only slightly increasing the size of the answer set *T(U)*. However, it only slightly impacted the accuracy of the algorithms, despite the significantly different dictionaries. Interestingly, when $|U| = 5$, the algorithm performed better on *H3* than on *H2* and *H1*. Similarly, when $|U| = 6$, the algorithm performed better on *H2* than on *H3* or *H1*. For all other cases, the algorithm more or less performed within expectations.

From this, we can infer that the ratio of lengths of words is actually more complicated than simply stating that the presence of small words in the dictionary leads to a greater inaccuracy.

The experiments with *H1* were repeated, except with three times the number of words in the dictionary (and three times the number of noisy substrings.) The results are as follows:

Table V

Results of Experiment H1 with 3x Words

Dict	**|U|**	**Est**	**Acc**
H1_3	5	S*(Y)	0.929
H1_3	5	$S^M(Y)$	0.891
H1_3	6	S*(Y)	0.968
H1_3	6	$S^M(Y)$	0.937
H1_3	7	S*(Y)	0.982
H1_3	7	$S^M(Y)$	0.961

The accuracy dropped considerably. We can therefore see that as the dictionary gets larger, the accuracy of the algorithm will decrease considerably. This is justified because there will be more chances for a “red herring” substring that looks like a better match to *Y* than all the other substrings in *H*, even to those which belonged to the original word from which *U* was derived. It is also worth noting that H1 was the most favorable dictionary for the algorithm (with a much larger concentration of larger words compared to smaller words).

In order to test a more non-ideal scenario, in Table V, a normal distribution was used to determine the distribution of word lengths in *H*. For the parameters, $\mu=4$ and $\sigma=3$, were selected with the equation multiplied by a constant factor of 3000. The number of words of length ≤ 5 was 1153, while the number of words of length ≥ 6 was 921. 3000 noisy substrings were used in each test. The results are as follows:

Table VI
Results of Experiments with More Lower Length Strings in the Dictionary (normal)

|U|	Est	Acc	Av. Size
5	S*(Y)	0.932	7.847
5	S^M(Y)	0.886	7.262
6	S*(Y)	0.968	2.893
6	S^M(Y)	0.940	2.712
7	S*(Y)	0.988	1.714
7	S^M(Y)	0.976	1.661

Interestingly, the accuracy did not drop that much, despite the very large number of words, most of which were small. The reason for this is probably because the small words most likely only rarely match the noisy substring *Y*. Even for $|U|=5$, it is still much more likely to match with some substring of a larger string than any of the strings of length ≤ 5.

Despite the very non-ideal situation, when $|U|$ was ≥ 7, the accuracy of the algorithm was still decent. It is only for small $|U|$ where the small words in the dictionary and the large size of the dictionary have a greater impact.

4 Remarks / Future Work

The algorithm performed surprisingly better than expected. Our tests confirmed the original authors' tests and also demonstrated that, as long as the length of the noisy substring is sufficiently large, the noisy substring algorithm will most likely succeed. The presence of small words in the dictionary has a decent impact on the accuracy of the algorithm, but only when $|U|$ is small. Similarly, a large number of words in the dictionary impacts the accuracy of the algorithm, but when $|U|$ is large this is somewhat mitigated.

With more time / space for experiments, we would have liked to have shown to what effect the size of the dictionary and the proportion of small words diminishes the accuracy of the algorithm, as well as what $|U|$ will lead to an optimal accuracy based on these factors. We would also like to further examine how the average size of the estimate set is affected by these factors.

ACKNOWLEDGMENTS

We would like to acknowledge Dr. Jerzy Jaromczyk for guiding us through this problem and providing us with many educational opportunities, as well as our colleagues for providing excellent feedback and great ideas.

REFERENCES

[1] R. L. Kashyap and B. J. Oommen, "The Noisy Substring Matching Problem," in *IEEE Transactions on Software Engineering*, vol. SE-9, no. 3, pp. 365-370, May 1983, doi: 10.1109/TSE.1983.237018.

[2] h3xx, "Wictionary top 100,000 most frequently-used English words [for john the ripper]." https://gist.github.com/h3xx/1976236

[3] Wikipedia, "Skew normal distribution." https://en.wikipedia.org /wiki/Skew_normal_distribution

[4] Desmos. https://www.desmos.com/