Approximating Edit Distance

Mahdi Safarnejad

Joint work with Saeed Seddighin and Soheil Ehsani
Under the supervision of Dr. Ghodsi and Dr. HajiAghai

Winter Seminar Series / Sharif University of Technology
29 December 2016
Edit Distance
(a.k.a Levenshtein distance)

The smallest number of insertions, deletions, and substitutions that need to be made on one string ($S_1$) to transform it to another one ($S_2$).
15-5 Edit distance

In order to transform one source string of text $x[1 \ldots m]$ to a target string $y[1 \ldots n]$, we can perform various transformation operations. Our goal is, given $x$ and $y$, to produce a series of transformations that change $x$ to $y$. We use an array $z$—assumed to be large enough to hold all the characters it will need—to hold the intermediate results. Initially, $z$ is empty, and at termination, we should have $z[j] = y[j]$ for $j = 1, 2, \ldots, n$. We maintain current indices $i$ into $x$ and $j$ into $z$, and the operations are allowed to alter $z$ and these indices. Initially, $i = j = 1$. We are required to examine every character in $x$ during the transformation, which means that at the end of the sequence of transformation operations, we must have $i = m + 1$.
Edit Distance: bioinformatics
Edit Distance: an example
Dynamic Programming - ~1970 - $O(n^2)$

$$d_{i,j} = \begin{cases} 
    d_{i-1,j-1}, & \text{if } s_1[i] = s_2[j] \\
    1 + \min\{d_{i-1,j-1}, d_{i,j-1}, d_{i-1,j}\}, & \text{if } s_1[i] \neq s_2[j]. 
\end{cases}$$
Edit Distance: dynamic programming

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>E</th>
<th>X</th>
<th>E</th>
<th>C</th>
<th>U</th>
<th>T</th>
<th>I</th>
<th>O</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>N</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>N</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>T</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>I</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>O</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>N</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>
Edit Distance: classic algorithms

* Dynamic Programming - ~1970 - O(n^2)

* Best algorithm - ~1980 - O(n^2/log^2 n)

* No better algorithm unless SETH fails

  * [Backurs & Indyk STOC’15]: no truly subquadratic is possible - O(n^{2-\varepsilon})
  
  * [Abboud et al. STOC’16]: not many log shaving are possible - O(n^2/log^{1000} n)
Edit Distance: approximation algorithms (in truly subquadratic time)

* $\sqrt{n}$ approximation algorithm (SIAM Journal on Comp. 1998)
* $n^{3/7}$ approximation algorithm (FOCS’04)
* $n^{1/3 + o(1)}$ approximation algorithm (SODA’06)
* $2^{O(\sqrt{\log n})}$ approximation algorithm (STOC’09)
* $O(\text{polylog } n)$ approximation algorithm (FOCS’10)
* No constant factor approximation algorithm!
Quantum Algorithms
Quantum Algorithms

* Substantially improve the running time of many algorithmic problems such as

  * Prime factorization and discrete logarithm problems
  * Many graph problems (connectivity, single source shortest paths, minimum spanning tree)
  * Pattern matching (finding a substring in a large string)
  * ...

Quantum Algorithms

* But NO improvement for many classic problems such as
  * Many fundamental problems: sorting / counting
  * All problem with a dynamic programming solution!
  * Edit distance

  * No exact or constant approximation algorithm in truly subquadratic time!
Main Quantum Technique: The Grover’s Search (1996)

* Element Listing

* Input: an array $f$ of length $m$, which has $k$ ones, and $m-k$ zeroes. $f(i)$ is available via an oracle access.

* Output: a list of $k$ indices for which $f(i) = 1$.

* The element listing problems can be solved with $O(\sqrt{mk})$ quantum oracle queries (vs. $O(m)$ classical queries).
Our Results and Techniques
Our Results

* We give the first quantum algorithm for approximating the edit distance within a constant factor in subquadratic time.

* An $O(n^{1.857})$ quantum algorithm with an approximation factor of 7.

* An $O(n^{1.781})$ quantum algorithm with a (large) constant approximation factor.
Our Approach

to find a good transformation of S1 into S2
Our Approach: outline

* I. Define some windows (substrings) on S₁ and S₂ and restrict ourselves to window-compatible transformations

* II. Find an approximate of edit distance between windows by using our metric estimation algorithm [the only step using quantum computing]

* III. Use these distances to find the best windows-compatible transformation, using dynamic programming

* IV. Show that the best windows-compatible transformation is not far from the best (general) transformation
emphasize that in order for a transformation to be window-compatible, the corresponding windows
in a window $S$. Intuitively, a window-compatible transformation with respects to two sequences of windows
the transformation. We call a transformation window-compatible, if it is window-compatible with
and (ii) every old character of window-compatible with respect to
non-overlapping windows from
Definition 2.3
notion of a character is not inserted during a transformation, it is called old. Based on this, we define the
substituted by a character of that turns
Figure 1:
Let $l_g$ be the window size
of the windows overlap.

Let $s_0$ be a character is not inserted during a transformation, it is called old. Based on this, we define the
substituted by a character of that turns
define a window of

$W_i$ as follows: for every $0 \leq i < n$

For clarity, we define two parameters $0 < \varepsilon, \delta < 1$. We use
analysis, we show which values for
real number between 0 and 1. We use
conquer and dynamic programming, married with the quantum results mentioned earlier for metric

We begin by defining the notion of a
transformation
of windows for

We construct a collection

We define a window-compatible transformation

$i_g$ be the window size

we call a character of

of windows for

...
What is a window-compatible transformation?

(a) An example of a window-compatible transformation.

(b) The transformation is not window-compatible since character 5 of the second string is old but doesn’t lie in any windows.
What is a window-compatible transformation?

(a) An example of a window-compatible transformation.

(b) The transformation is not window-compatible since character 5 of the second string is old but doesn't lie in any windows.

(c) The transformation is not window-compatible since character 1 of the second string is old but prior to the transformation, it was not placed in any windows.

(d) The transformation is not window-compatible since character 3 of the second string is old but prior to the transformation, it was not placed in the corresponding window.

Figure 2: Figures 2a, 2b, 2c, and 2d show a few examples of window-compatible and window-incompatible transformations. Solid arrows show substitutions, dashed arrows show the characters that remain in the string, and other characters are either inserted or deleted.

As we show in the following, window-compatible transformations are well-structured. In fact, we show in Section 4 that if the edit distances of the windows are accessible in time $O(1)$, a dynamic program can find an optimal window-compatible transformation of $s_1$ into $s_2$ in time $O(n + |W_1| |W_2|)$.

Lemma 4.1 [restated]. Given a matrix of edit distances between the substrings corresponding to every pair of windows of $W_1$ and $W_2$, one can compute an optimal window-compatible transformation of $s_1$ into $s_2$ in time $O(n + |W_1| |W_2|)$. This makes the connection of edit distance and metric estimation more clear.
How good is the best window-compatible transformation?

* If \( \text{edit}(S_1, S_2) \leq \delta n \), there exists a window-compatible transformation of \( S_1 \) into \( S_2 \) with at most \( 3\delta n + n/\gamma + 2l \) operations.

* Proof idea: substitution and intact links / construct windows / shift constructed windows to actual windows
How do we find the best windows-compatible transformation?

* Using dynamic programming
  * Time complexity: $O(n + |W_1| |W_2|)$
  * If we have all edit distances between windows
  * How can we find all edit distances between windows?
Metric Estimation
Metric Estimation

* For any set of $m$ strings like $M$, $\langle M, \text{edit} \rangle$ forms a metric system

* We give two approximation algorithms:

<table>
<thead>
<tr>
<th></th>
<th>Approximation Factor</th>
<th>Query Complexity</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm I</td>
<td>$3 + \epsilon$</td>
<td>$\tilde{O}(m^{5/3} \text{poly}(1/\epsilon))$</td>
<td>$O(m^2 \text{poly}(1/\epsilon))$</td>
</tr>
<tr>
<td>Algorithm II</td>
<td>$O(1/\epsilon)$</td>
<td>$\tilde{O}(m^{3/2+\epsilon} \text{poly}(1/\epsilon))$</td>
<td>$O(m^2 \text{poly}(1/\epsilon))$</td>
</tr>
</tbody>
</table>
Metric Estimation: Algorithm I

* The first idea: discretize the problem
* The second idea: solving for a single threshold as a graph
* The third idea: low degree and high degree vertices
The first idea: discretize the problem

Intervals of the form $[x, (1 + \epsilon/3)x]$

$[(1 + \epsilon/3)^{k-1}, (1 + \epsilon/3)^k]$

$\log_{1+\epsilon/3} U = \tilde{O}(\text{poly}(1/\epsilon))$ disjoint intervals

Add a $(1+\epsilon/3)$ term to the approximation factor
The second idea: threshold

- Given a threshold $t$, find all pairs $(p_i, p_j)$ such that $d(p_i, p_j) \leq t$

- With some false positives $d(p_i, p_j) \leq 3t$

- A term of 3 to approximation factor

- No false negative
Metric Estimation: Algorithm I (cont.)

* The third idea

* For low degree (<\(m^{1/3}\)) vertices, find all neighbors via the Grover’s search, then remove it

* For high degree (\(\geq m^{1/3}\)) vertices, find all neighbors \(N(v,t)\), then find all neighbors with threshold \(2t\), \(N(v,2t)\)

  * Connect the two sets, all of the neighbors of \(N(v,t)\) is in \(N(v,2t)\), and they’re not far away.

  * Remove \(v\) and \(N(v,t)\)
Metric Estimation: Algorithm I Analysis

* Amortized analysis (query complexity)

  * For low degree (<$m^{1/3}$) vertices, we spend $O(\sqrt{m} \cdot m^{1/3}) = O(m^{2/3})$ for 1 vertex

  * For high degree ($\geq m^{1/3}$) vertices, we spend $O(m)$ for at least $O(m^{1/3})$ vertices, or $O(m^{2/3})$ for 1 vertex

* $O(m^{5/3})$ for a fixed threshold

* $O(m^{5/3} \text{poly}(1/\epsilon))$ for Algorithm 1 (which is truly subquadratic)
Edit Distance
**Edit Distance**

* Compute the edit distance of $S_1$ and $S_2$

* We use Algorithm I and Algorithm II as a subroutine

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Approximation</th>
<th>Time and Query Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm III</td>
<td>$7 + \epsilon$</td>
<td>$\tilde{O}(n^{2-1/7} \text{poly}(1/\epsilon))$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$= O(n^{1.857})$</td>
</tr>
<tr>
<td>Algorithm IV</td>
<td>$O(1/\epsilon)^{O(\log 1/\epsilon)}$</td>
<td>$\tilde{O}(n^{2-(5-\sqrt{17})/4+\epsilon} \text{poly}(1/\epsilon))$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$= O(n^{1.781})$</td>
</tr>
</tbody>
</table>
Edit Distance: $\delta$-bounded edit distance

* Solve the $\delta$-bounded edit distance

* Guarantee: $\text{edit}(S_1, S_2) \leq \delta(|S_1| + |S_2|)$

* For $\delta \leq n^{-1/14}$ we run an exact algorithm of $O(n+d^2)=O(n^{2-1/7})$

* So we can assume $\delta > n^{-1/14}$

* Note: $\delta$ does not affect the approximation factor, it affects the time, so we tune the parameters with $\delta$!
Edit Distance: Algorithm III

* For a $\delta > n^{-1/14}$

* Construct some windows of $S_1$ and $S_2$ where the number of windows and the size of them are $o(n)$

* Solve the metric estimation problem for this windows using Algorithm I and the classical dynamic programming as the distance oracle

* Find the best windows-compatible transformation
With the classic algorithm for this problem to approximate the edit distance of the strings in time \( O \), but the running time improves. In this section, we show how we combine these ideas to achieve time. Of course, this again comes at the expense of deteriorating the approximation guarantee.

We can use our algorithm for approximating edit distance to implement the oracle in subquadratic edit distances of the windows; a constant estimation to the distances suffices. To this, as an oracle function for metric estimation, we do not really need to compute the exact approximation and estimate the distances in time.

Figure 3 depicts the components of the algorithm.

As discussed before, such a solution approximates the edit distance within a constant factor. The dynamic programming algorithm to find an almost optimal window-compatible transformation. Next, we use metric estimation to estimate the edit distances of the windows and finally we use a dynamic programming algorithm to find an almost optimal window-compatible transformation.

So far, we discussed how to use divide and conquer and metric estimation to approximate edit distances of the windows; a constant estimation to the distances suffices. To this, as an oracle function for metric estimation, we do not really need to compute the exact approximation and estimate the distances in time.

Thus, to formalize the above ideas, suppose we are given two strings and , respectively. We inductively show that a \( \tilde{O}(n^5/3 \text{poly}(1/\epsilon)) \) algorithm and the approximation factor converge to as a black box.

For \( \epsilon \) small enough, the approximation algorithm for metric estimation, we can lose a factor of . Notice that if \( \tilde{O}(n/\epsilon) \) and \( \tilde{O}(1/\epsilon) \), respectively. We inductively show that a \( \tilde{O}(n^5/3 \text{poly}(1/\epsilon)) \) algorithm and the approximation factor converge to as a black box.

We refer the reader to a discussion in Section 5.

We call our approximation algorithm for metric estimation, we can lose a factor of . Notice that if \( \tilde{O}(n/\epsilon) \) and \( \tilde{O}(1/\epsilon) \), respectively. We inductively show that a \( \tilde{O}(n^5/3 \text{poly}(1/\epsilon)) \) algorithm and the approximation factor converge to as a black box.

We refer the reader to a discussion in Section 5.
Metric Estimation: Algorithm II

* Approximation factor $O(1/\epsilon)$

* Query Complexity $O^\sim(m^{3/2+\epsilon} \text{poly}(1/\epsilon))$

* Time Complexity $O^\sim(m^2\text{poly}(1/\epsilon))$

* Idea: handle high degree vertices with a hitting set (random sampling) and solve them recursively
Algorithm IV: Bootstraping

\[ \tilde{O}(n^{1.781+\epsilon}\text{poly}(1/\epsilon)) \]
\[ \tilde{O}((1/\delta)^2 n^{1.562+\epsilon}\text{poly}(1/\epsilon)) \]
\[ \tilde{O}(n^{3/2+\epsilon}\text{poly}(1/\epsilon)) \]

\[ A(\epsilon) \]
\[ e_\epsilon(\epsilon) \text{ Edit Distance} \]
\[ \delta \text{-bounded edit distance} \]
\[ O(1/\epsilon) \text{ Metric Estimation} \]
\[ O(n + d^2) \]

Landau et al.

\[ - \]
\[ \tilde{O}(n^2) \]

\[ \text{DP} \]
## Edit Distance (recall)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Approximation</th>
<th>Time and Query Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm III</td>
<td>$7 + \epsilon$</td>
<td>$\tilde{O}(n^{2-1/7} \text{poly}(1/\epsilon))$ \hspace{1cm} $= O(n^{1.857})$</td>
</tr>
<tr>
<td>Algorithm IV</td>
<td>$O(1/\epsilon)^O(\log 1/\epsilon)$</td>
<td>$\tilde{O}(n^{2-(5-\sqrt{17})/4+\epsilon} \text{poly}(1/\epsilon))$ \hspace{1cm} $= O(n^{1.781})$</td>
</tr>
</tbody>
</table>
Any question?
Thank you for your time