Simpler Constant-Factor Approximation to Edit Distance Problems

Alex Andoni

October 6, 2018

1 Introduction

Computing edit distance between two strings of length n is a classic dynamic programming problem, with a quadratic run-time solution. It has proven to be a great challenge to one of the central themes in TCS: to improve the run-time from polynomial to something closer to linear. Despite significant research over many decades, the running time has so far been improved only slightly, to $O(n^2/\log^2 n)$ [MP80], which remains the fastest algorithm known to date. See also the surveys of [Nav01] and [Sah08]. With the emergence of the fine-grained complexity field, researchers crystallized the reason why beating quadratic-time is hard by connecting to the SETH conjecture [BI15] (and even more plausible conjectures [AHVW16]).

Even before the above hardness results, researchers started considering faster algorithms that *approximate* edit distance. A linear-time $\sqrt{n}$-approximation algorithm immediately follows from the exact algorithm of [Ukk85, Mye86], which runs in time $O(n + d^2)$, where d is the edit distance between the input strings. Subsequent research improved the approximation factor, first to $n^{3/7}$ [BJKK04], then to $n^{1/3+o(1)}$ [BES06], and to $2^{\tilde{O}(\sqrt{\log n})}$ [AO12]. In the regime of $O(n^{1+\epsilon})$-time algorithms, the best approximation is $(\log n)^{O(\epsilon)}$ [AKO10]. Predating some of this work was the *sublinear-time* algorithm of [BEK$^+$03] achieving n^ϵ approximation when d is large.

In a recent breakthrough, [DGKS18] showed that one can obtain constant approximation in time $O(n^{1.618})$. The authors suggested that their techniques may be improved to get $5+\epsilon$ approximation in $O(n^{2-2/7})$ time, as well as constant approximation in time close to $n^{3/2}$.

Here we provide an alternative implementation of the core idea from [DGKS18] with the main goal of developing a simpler constant-factor approximation algorithm with sub-quadratic run-time, while achieving the best possible bounds. To obtain our results, we need to generalize the problem to the well-known *text searching* problem (in fact, a further generalization of it, discussed later).

Definition 1.1. *Fix $n \geq m$, some length $\lambda \geq 1$, and approximation $\alpha > 1$. For $x \in \Sigma^m$ and $y \in \Sigma^n$, the problem $\mathsf{TextSearch}_\alpha(x, y, \lambda)$ is to α-approximate the edit distance between x and every substring of y of length λ.*

The main results are summarized by the following two theorems.

Theorem 1.2 (Main, $n^{3/2+\delta}$ time)**.** *For any $x \in \Sigma^m$ and $y \in \Sigma^n$, and $\delta > 0$, we can solve the $\mathsf{TextSearch}_\alpha$ problem in time $O(nm^{1/2+\delta})$ for approximation α that depends on $1/\delta$ only.*

Theorem 1.3 (Main $3+\epsilon$ approximation)**.** *For any $x, y \in \Sigma^n$, and $\epsilon > 0$, we can estimate the edit distance up to $3+\epsilon$ factor approximation and an $O(m/\beta)$ additive error, for $\beta \in [m]$, in time $\tilde{O}_\epsilon(n^{1.6}\beta^{0.6})$. When combined with $O(n+d^2)$-time algorithm [Ukk85, Mye86], edit distance may be $(3+\epsilon)$-factor approximated in $O_\epsilon(n^{1.693})$ time.*

To obtain Theorem 1.2, we generalize the text searching problem a bit further, to multi-string text searching. We describe this problem, and its connection to our main theorems on text searching in the next section. In Section 3, we describe and analyze our algorithm for the multi-string text searching.

2 Multi-String Text Searching

For simplicity, we index everything from zero. Hence, $[n] = \{0, 1, \ldots n-1\}$. Also $x[i : j]$ is the string starting at position i until position $j-1$. For index i and set S, we use $i + S$ to denote the set $\{i + s : s \in S\}$.

Definition 2.1. *Fix $n \geq m$, and approximation $\alpha > 1$. For $x \in \Sigma^m$, $y \in \Sigma^n$, and a "start-set" $S_y \subset [n]$, the problem $\mathsf{TextSearchStart}_\alpha(x, y, S_y)$ is to output, for every $t \in [n]$, an α-approximation to $\min_{s \in S_y} \mathrm{ed}(x, y[s : t])$.*

Lemma 2.2 (Reduction from $\mathsf{TextSearchStart}$ to $\mathsf{TextSearch}$)**.** *We can reduce the problem $\mathsf{TextSearch}_{3\alpha}$ to the problem $\mathsf{TextSearchStart}_\alpha$, up to an extra time of $O(n)$.*

Proof. To solve $\mathsf{TextSearch}_{3\alpha}(x, y, \lambda)$, run the $\mathsf{TextSearchStart}_\alpha(x, y, S)$ instance with the set $S = [n]$. For c_t the output for index $t \in [n]$, we output c_t as the approximation to $\mathrm{ed}(x, y[t - \lambda : t])$.

Note that $c_t \leq \alpha \,\mathrm{ed}(x, y[t-l : t])$. Suppose $s \in [n]$ is such that $\mathrm{ed}(x, y[s : t]) \leq c_t$. Then $\mathrm{ed}(x, y[t-l : t]) \leq |m-l| + \mathrm{ed}(x, y[t-m : t]) \leq |m-l| + |m-s| + \mathrm{ed}(x, y[s : t]) \leq |m-l| + 2c_t$. Since $|m-l| \geq \mathrm{ed}(x, y[t-l : t])$, we have that $|m-l| + 2c_t \leq 3\alpha \cdot \mathrm{ed}(x, y[t-l])$ and hence a 3α approximation. $\square$

Definition 2.3. *Fix $n \geq m$, some lengths $\lambda_x, \lambda_y \geq 1$, start positions into x termed S_x, and approximation $\alpha > 1$. For $x \in \Sigma^m$ and $y \in \Sigma^n$, the problem $\mathsf{MultiTextSearch}_\alpha(x, y, S_x, \lambda_x, \lambda_y)$ is to solve the problem $\mathsf{TextSearch}_\alpha(x[s : s + \lambda_x], y, \lambda_y)$ for all $s \in S_x$ (i.e., compute the edit distance between every substring of x of length λ_x starting at $s \in S_x$, and every substring of y of length λ_y).*

Similarly, define $\mathsf{MultiTextSearchStart}_\alpha(x, y, S_x, \lambda_x, S_y)$ to be solving all the problems $\mathsf{TextSearchStart}_\alpha(x[s : s + \lambda_x], y, S_y)$ for $s \in S_x$.

Note that the above reduction also holds for reducing $\mathsf{MultiTextSearch}$ to $\mathsf{MultiTextSearchStart}$.

2.1 Main technical contributions

The main technical statement is the following theorem, which shows how to solve $\mathsf{MultiTextSearchStart}$ problem using an algorithm for $\mathsf{MultiTextSearch}$ on smaller strings. We then state how it implies the main theorems.

Theorem 2.4. *Fix $n \geq m \geq w, \lambda_x, \lambda_y \geq 1$. Suppose there's an algorithm $\mathcal{A}$ to solve $\mathsf{MultiTextSearch}_\alpha(\cdot, \cdot, S_x, O(w), O(w))$, for strings of length $O(w)$ in time $t(w, |S_x|)$. We assume that $t(w, |S_x|)$ is monotonic, $t(cw, |S_x|) \in [c, c^2] \cdot t(w, |S_x|)$ for any $c \geq 1/w$, and $\sum_{i=1}^k t(w, s_i) \leq O(k \cdot t(w, 1) + t(w, \sum s_i))$ for any $s_i \geq 1$.*

Then, for any $\epsilon > 0$, there exists a randomized algorithm to solve $\mathsf{MultiTextSearchStart}_{\alpha'}(x, y, S_x, \lambda_x, \lambda_y)$, for $x \in \Sigma^n, y \in \Sigma^m$, in time

$$O(\tfrac{n\sqrt{m}}{w^{1.5}} \cdot t(w, 1) + \tfrac{n}{w} \cdot t(w, \tfrac{m}{w}) + s(\tfrac{n}{w} \cdot t(w, 1) + n \cdot \tfrac{m^2}{w^2})) \cdot (\tfrac{\log n}{\epsilon})^{O(1)}.$$

where the approximation is $\alpha' = (1 + \epsilon)(2\alpha + \alpha^2)$.

Corollary 2.5. *Fix $\delta \in (0, 1/5)$. Then, for any $i \geq 1$, we can solve* $\mathsf{MultiTextSearch}_\alpha(x, y, S_x, \lambda_x, \lambda_y)$*, where $|x|, |y|, \lambda_x, \lambda_y \leq n$, and $s = |S_x|$, in time:*

$$t_i(n, s) \leq (\log n)^{O(i)} \cdot n^{1.5} \cdot n^{f(i)} \cdot (1 + \tfrac{s}{n^{\delta/2}}),$$

where $f(i) \leq c(1-\delta)^i + \delta(1-(1-\delta)^i)$ for a constant $c > 0$, and approximation α only dependent on i.

Proof. First of all, the algorithm is the obvious recursive one: we use Theorem 2.4 recursively to solve MultiTextSearchStart with extra approximation factor $3 + \epsilon \leq 4$, and use reduction from Lemma 2.2 to reduce to solve MultiTextSearch with an extra factor 3. After i iterations, the approximation is only a funtion of i (specifically, double-exponential in i).

We now prove the runtime. Base case: for $i = 0$, the standard dynamic programming for the exact problem would run in time $O(sn^2)$, and hence the statement is true for high enough $c > 0$.

Next, we prove the inductive step. We use Theorem 2.4 with $n = m$, $\beta, \lambda_x, \lambda_y \leq n$, and $w = n^{1-\delta}$. For $t_i(n, s)$ denoting the runtime after i recursive steps, the inductive hypothesis can be written as:

$$\tfrac{t_i(w,s)}{w\sqrt{w}} \leq (\log w)^{O(i)} \cdot w^{f(i)} \cdot (1 + s/w^{\delta/2}).$$

Plugging this into the theorem, we obtain:

$$\tfrac{t_{i+1}(n,s)}{n\sqrt{n}} \leq (\log n)^{O(i+1)} \cdot \left(w^{f(i)} + n^{-\delta/2} \cdot w^{f(i)} \cdot (1 + n^{\delta}/w^{\delta/2}) + s \cdot (n^{-\delta/2} \cdot w^{f(i)} + \tfrac{n^{1+2\delta}}{n\sqrt{n}})\right)$$

$$\leq (\log n)^{O(i+1)} \cdot \left(2n^{(1-\delta)\cdot f(i)} + n^{-\delta/2} \cdot n^{(1-\delta)f(i)} \cdot n^{\delta}/n^{-(1-\delta)\delta/2} + s \cdot (n^{-\delta/2} \cdot n^{(1-\delta)f(i)} + n^{2\delta - 1/2})\right)$$

$$\leq (\log n)^{O(i+1)} \cdot n^{(1-\delta)f(i)} \cdot \left(1 + n^{\delta^2/2} + s/n^{-\delta/2}\right).$$

Comparing above versus the hypothesis for $i+1$, it only remains to prove that $f(i+1) \leq (1-\delta)f(i) + \delta^2/2$. Indeed, we have that:

$$f(i+1) = c(1-\delta)^{i+1} + \delta(1-\delta)(1-(1-\delta)^i) + \delta^2/2 \leq c(1-\delta)^{i+1} + \delta(1-(1-\delta)^{i+1}).$$

□

The above corollary immediately implies Theorem 1.2 by using $\delta \leftarrow \delta/2$ and $i = O(1/\delta \cdot \log 1/\delta)$.

The Theorem 1.3 also follows from Theorem 2.4 when using the exact algorithm for TextSearch as a primitive [LV89]. While Theorem 1.3 statement does not include the dependence on β (of the statement of Theorem 1.3), the proof does. See Remark 3.6, and set $w = (m\beta)^{1/5}$.

3 Main algorithm for MultiTextSearchStart: proof of Theorem 2.4

We setup some further notation. Assume that n, m are divisible by w, as otherwise we can extend x, y so that their lengths are multiple of w. Also suppose we settle for an additive m/β approximation, for $\beta \leq m$.[1] Let $E \in [n]$ be the set of the powers of $1+\epsilon$, namely $E = \{\lfloor (1+\epsilon)^i \rfloor \mid i = 0, \ldots \log_{1+\epsilon} n\}$.

[1]For simplicity, the reader can just assume that $\beta = m$—this parameter merely allows for possible extensions.

We partition x into $b = m/w$ blocks of length w with starting positions in the set $I = w \cdot [m/w]$, termed $X_i = x[i : i+w]$ for $i \in I$. We consider two types of blocks of y. Let $J_w = w \cdot [n/w] \subset J$ to be the start of w-length non-overlapping blocks of y, and $J = \Delta \cdot [n/\Delta]$, where $\Delta = \max\{1, \epsilon w/\beta\}$, be the starting positions of possibly overlapping blocks. Let $L = (w \pm \Delta \cdot E) \cap [w/\epsilon]$ be the set of lengths. Note that $|L| \le O(\frac{\log n}{\epsilon})$. We denote y-blocks as $Y_{j,l} = y[j : j+l]$ where $j \in J$ and $l > 0$.

By assumption, there exists an algorithm $\mathcal{A}$ for solving $\mathsf{MultiTextSearch}_\alpha(x, y, S, \lambda_x, \lambda_y)$ for $x, y \in \Sigma^{O(w/\epsilon)}$, and $\lambda_x, \lambda_y \le O(w/\epsilon)$, in time $O(1/\epsilon^2) \cdot t(w, |S_x|)$. Note that, when the length of y is $n > w$, we can solve $\mathsf{MultiTextSearch}_\alpha(x, y, S, \lambda_x, \lambda_y)$ in time $O(n/w \cdot t(w, |S_x|))$ by the standard reduction — just partition y into $2n/w = O(n/w)$ overlapping blocks of length $2w$ and solve $\mathsf{MultiTextSearch}$ for each block of y separately.

Note that we can use $\mathcal{A}$ to solve $\mathsf{TextSearch}(x, y, \lambda)$ (using $\lambda_x = |x|$ and $S_x = \{0\}$), as well as to estimate $\mathrm{ed}(x, y)$—for clarity, we call the algorithms $\mathcal{A}_{TS}$ and $\mathcal{A}_{\mathrm{ed}}$. Both runtimes are $t(w, 1)$. For simplicity, we denote $t(w) = t(w, 1)$. While the algorithms $\mathcal{A}, \mathcal{A}_{TS}, \mathcal{A}_{\mathrm{ed}}$ may be randomized, we assume that, when run on the same input, it will produce the exact same output (by, say, memoization). Specifically, we often use $\mathcal{A}_{\mathrm{ed}}(X_i, Y_{j+j',l})$, for $j \in J_w, j' \in \Delta\mathbb{Z} \cap [w]$, which are extracted from running $\mathcal{A}_{TS}(X_i, Y_{j,w+l}, l)$ and hence considered to be a deterministic quantity.

3.1 Algorithm description

At the high level, the algorithm constructs a graph corresponding the dynamic programming table in two phases, and then runs the relevant shortest path computations in the third phase.

We build a grid graph on vertices $\bar{I} \times \bar{J}$, which are just respectively I, J with one extra index at the end. We have edges (i, j) to $(i+w, j)$ of cost w, as well as edges (i, j) to $(i, j+\Delta)$ of cost Δ. Most importantly, for each triple $(i, j, l) \in I \times J \times L$, we will add the edge[2] $(i, j) \to (i+w, j+l)$ with some cost which upper bounds the distance $\mathrm{ed}(X_i, Y_{j,l})$. In the first phase, the algorithm produces upper bounds for all the triples (i, j, l). The second phase will update *some* of these edges with a more accurate upper bound on the cost.

In the third, estimation phase, we compute the shortest path in the graph for each required substring of x.

Phase 1: dense blocks. Fix $k = \Theta(\sqrt{b})$ to be the number of y-centers. Pick k random indeces $j \in J_w$ (the centers), forming the set $C \subset J_w$.

For each $i \in [b]$ and $(j, l) \in C \times L$, solve the problem $\mathsf{TextSearch}(X_i, Y_{j,w+l}, l)$ using $\mathcal{A}_{TS}$. Define d_i be the minimal distance found over all considered substrings of y: in particular (α-approximation of) $\min_{j \in C, l \in L, j' \in \Delta\mathbb{Z} \cap [w]} \mathrm{ed}(X_i, Y_{j+j',l})$, and π_i be the corresponding minimizing pair $(j+j', l)$. At the same time, form the sets $S_{j,l}$, where $j \in C, l \in L$, by adding to $S_{j,l}$ the index $j' \in \Delta\mathbb{Z} \cap [w]$ each time we have $\pi_i = (j+j', l)$. Note that $\sum_{j,l} |S_{j,l}| \le b$.

For each $(j, l) \in C \times L$, solve $\mathsf{MultiTextSearch}(Y_{j,w+l}, y, S_{j,l}, l, l')$ for all $l' \in L$. We store the result for fixed (j, l) as a (sparse) table $K_{\pi,\tau}$ where $\pi \in (j + S_{j,l}, l)$ and $\tau \in J \times L$.

Finally, add to the graph all edges corresponding to (i, j, l), where $i \in [b], j \in J, l \in L$, with cost $d_i + K_{\pi_i,(j,l)}$.

Phase 2: sparse blocks. We build a binary tree of depth $O(\log b)$ as follows. Each node corresponds to a sub-interval of I, where the root corresponds to the entire I. For each node,

[2] Some indeces go out of bounds — in the description below, we will just ignore such edges as being irrelevant.

starting from the root, we partition its interval into two and assign them to the two children. The tree has $|I|$ leafs corresponding to singletons. Now, for each node v, with the interval $U_v \subseteq I$, we sample $h = O(\frac{\log n}{\epsilon})$ anchors at random from U_v, with probabilities proportional to d_q (say, we sample with repetition): i.e., q is chosen with probability $d_q / \sum_{q' \in U_v} d_{q'}$. The resulting set of anchors is termed Q_v.

For the ensuing computations, we introduce a definition:

Definition 3.1. *Consider two triples $(q, j, l), (q', j', l') \in I \times J \times L$, where $q < q'$. Then (q, j, l) and (q', j', l') are* compatible *if $j \leq j'$ and $j' - j \leq \frac{1}{\epsilon} \cdot (q' - q)$.*

For each top-level anchor $q \in I$, we estimate the distance from X_q to each block Y_π for $\pi \in J \times L$, by solving $\mathsf{TextSearch}(X_q, y, l)$ using $\mathcal{A}_{TS}$ for each $l \in L$ separately. Let S_q be the set of pairs (j, l) such that the estimated distance is less than d_q. For each anchor q at non-top level, whose parent r has anchors $Q_r \subset I$, we compute the set $\breve{S}_q \subseteq J \times L$ to be the set of pairs (j, l) such that (i, j, l) is *compatible* with at least one triple from $\cup_{q' \in Q_r} q' \times S_{q'}$. Then, we estimate the distance from X_q to each Y_π for $\pi \in \breve{S}_q$, and store the set S_q of all pairs where the estimated distance is $< d_q$ (updating the corresponding costs in the graph).

We now estimate the distance from X_q to Y_π for $\pi \in \breve{S}_q$ by running $\mathsf{TextSearch}$ on relevant blocks of y. We decompose $[n]$ into blocks with starting positions that are multiples of w and consider only blocks where $\breve{S}$ is non-empty. Specifically, for each $j \in J_w$ and $l \in L$, if there's $j' \in [j : j + w]$ with $(j', l) \in S_q$, then we run $\mathsf{TextSearch}(X_q, Y_{j,w+l}, l)$. Note that this covers all relevant substrings encoded by $\breve{S}_q$.

Phase 3: Shortest paths in the graph. Now consider each starting position $s \in S_x$ into x (from the problem input). Let $t = s + \lambda_x$ be the ending position. If $s, t \in \bar{I}$, then we simply run the shortest path algorithm from the set of vertices (s, S_y) to each of terminal vertices $(t, \bar{J})$. Then for each $z \in [n]$, output the distance from the source to the node (t, j) where j is the closest $j \in J$ to z (note that this introduces additive error only $\Delta \leq \epsilon \lambda_x / \beta$.

Now suppose $s \notin I$ or $t \notin \bar{I}$. Let $s' \in I$ be the smallest $s' \geq s$ and similarly $t' \in \bar{I}$ be the largest $t' \leq t$. If it happens that $s' > t'$ (s, t are between same two indeces in I), we just run $\mathsf{TextSearchStart}(x[s : t], y, [n])$. Otherwise, we solve the problem $\mathsf{TextSearchStart}(x[s : s'], y, [n])$, recording the results as Q_i for $i \in [n]$. We also solve the problem $\mathsf{TextSearch}(x[t' : t], y, l)$, for all $l \in E \cdot \frac{|t - t'|}{\epsilon}$. We record these results as $W_{i,l}$. Finally, we compute the shortest paths in the following graph:

- the source node has an edge to each node (s', j), for $j \in \bar{J}$, with cost Q_j;
- create a terminal node (t, z) for each $z \in [n]$, and for each (t', j, l), $j \in \bar{J}, l \in L$, add edge from (t', j) to $(t, j + l)$ with cost $W_{j+l,l}$;
- finally run the shortest path algorithm from the source to all the terminal nodes. For each $z \in [n]$, output the distance to the node (t, z).

3.2 Analysis: correctness

First, we note that, by the construction of the graph, for any path of cost ζ from $(s_x, s_y) \in I \times J$ to $(t_x, t_y) \in \bar{I} \times \bar{J}$, then there's an edit distance matching of cost at most ζ: in particular, if edge

$(i,j) \to (i',j')$ has cost c, then we can align $x[i:i']$ to $y[j:j']$ with edit distance cost at most c. Hence, our main task is to prove that there exists a path of cost at most $O(\mathrm{ed}(x,y))$.

We use the following lemma to characterize a path in our graph, of cost close to $\mathrm{ed}(x,y)$.

Lemma 3.2. *Fix $\epsilon > 0$, as well as $1 \le \beta \le w \le m$ and $n \ge 1$, and two strings x, y of lengths m, n, divisible by w. Let $I = w\mathbb{Z} \cap [m]$ and $J = \Delta\mathbb{Z} \cap [n]$, where $\Delta = \epsilon w/\beta$. There's a matching between the x-blocks and y-blocks, (i, j_i, l_i) where $i \in I$, such that*[3]

$$\sum_{i \in I} \mathrm{ed}(X_i, Y_{j_i,l_i}) + (j_i - (j_{i-1} + l_{i-1})) \le (1 + O(\epsilon))\,\mathrm{ed}(x,y) + O(\epsilon n/\beta), \tag{1}$$

and (j_i, l_i) satisfy the following properties:

- *$(j_i, l_i) \in J \times L$;*
- *they are disjoint and in order: $j_i + l_i \le j_{i+1}$;*
- *for any $i < i'$, the triples (i,j,l) and (i',j',l') are compatible: i.e., $j_{i'} - j_i \le \frac{1}{\epsilon} \cdot (i' - i)$.*

Now consider a shortest path corresponding to the substring $x[s : s+\lambda_x]$. For simplicity, assume that $s, s+\lambda_x \in I$. Then we can apply the above lemma on $x[s : s+\lambda_x]$ and any substring of y with start and end position from J, with the same parameters β and w. When s or $s+\lambda_x \notin I$, we can repeat the same argument assuming we extend the substring of x and y correspondingly.

We fix the matching $(i, j_i, l_i)_{i \in I}$ from the above lemma. It now remains to show that the corresponding path in graph is not too large. While for some edges (i, j_i, l_i), its cost is close to the real cost of $\mathrm{ed}(X_i, Y_{j_i,l_i})$, sometimes the cost may be gross over-estimate. Nonetheless, we show that the overall cost is close to the one from Eqn. (1). For the rest, define c_i to be the result of running $\mathcal{A}_{\mathrm{ed}}$ to estimate $\mathrm{ed}(X_i, Y_{l_i,j_i})$: i.e., $\mathrm{ed}(X_i, Y_{l_i,j_i}) \le c_i \le \alpha\,\mathrm{ed}(X_i, Y_{l_i,j_i})$.

First note that, in the first stage of the algorithm, for each $(j,l) \in J \times L$, we add edge corresponding to (i,j,l), with cost $d_i + K_{\pi_i,(j,l)}$. In particular for $(j,l) = (j_i, l_i)$, we have

$$K_{\pi_i,(j_i,l_i)} \le \alpha \cdot \mathrm{ed}(Y_{\pi_i}, Y_{j_i,l_i}) \le \alpha\,\mathrm{ed}(Y_{\pi_i}, X_i) + \alpha\,\mathrm{ed}(X_i, Y_{j_i,l_i})$$

and hence the edge (i, j_i, l_i) has cost at most $d_i + K_{\pi_i,(j_i,l_i)} \le d_i + \alpha\,\mathrm{ed}(Y_{\pi_i}, X_i) + \alpha\,\mathrm{ed}(X_i, Y_{j_i,l_i})$ (it's "at most" because it may be replaced by lower-cost edge later on). Note that for i's where $c_i \ge d_i$, the cost of the edge (i, j_i, l_i) is in fact a good approximation: $\le c_i + \alpha c_i + \alpha\,\mathrm{ed}(X_i, Y_{j_i,l_i}) \le (2\alpha + \alpha^2)\,\mathrm{ed}(X_i, Y_{j_i,l_i})$.

The main challenge is to bound the cost of edges (i, j_i, l_i) when $c_i < d_i$, which is the purpose of the second phase of the algorithm. Define the set Z of "sparse" blocks to be the set of $i \in I$ where $c_i < d_i$. For a node v with sub-interval $U_v \subseteq I$, we call v to be *successful* if: 1) if the set of anchors $Q_v \subset U_v$ includes at least one $q \in Z$, and 2) for some $q \in Q_v \cap Z \neq \emptyset$, we have that $(j_q, l_q) \in S_q$.

Lemma 3.3. *Consider any node v, and suppose all its ancestors are successful. If $\sum_{i \in U_v} c_i \le \sum_{i \in U_v} (1-\epsilon) d_i$, then v is also successful with high probability.*

Proof. We first show that $Q_v \cap Z \neq \emptyset$, using the following claim.

Claim 3.4. *Consider p pairs of reals $c_i, d_i \ge 0$, such that $\sum_i c_i < (1-\epsilon)\sum_i d_i$. Then, if we pick $i \in [p]$ with probability $d_i / \sum_i d_i$, we have that $\Pr_i[c_i < d_i] \ge \epsilon$.*

[3] By convention, $j_{-1} = l_{-1} = 0$.

Proof. We have that:

$$1-\epsilon > \frac{\sum_i c_i}{\sum_i d_i} = \frac{1}{\sum_i d_i}\left(\sum_{i:c_i\ge d_i} c_i + \sum_{i:c_i<d_i} c_i\right) \ge \frac{1}{\sum_i d_i}\cdot \sum_{i:c_i\ge d_i} d_i = 1-\Pr_i[c_i<d_i],$$

and hence $\Pr_i[c_i < d_i] \ge \epsilon$. ☐

In particular, we apply the above claim to pairs (c_i, d_i) for $i \in U_v$. By the assumption that $\sum_{i\in U_v} c_i \le \sum_{i\in U_v}(1-\epsilon)d_i$, we have that, by our choice of a random anchor i, $\Pr_i[i \in Z] = \Pr_i[c_i < d_i] \ge \epsilon$. Hence, if we sample $h = \Omega(\frac{\log n}{\epsilon})$ anchors, we will sample at least one anchor from the set $U_v \cap Z$ with high probability.

Now, for an anchor $q \in Q_v \cap Z$, we want to prove that $(j_q, l_q) \in S_q$. We prove this by induction on the distance from the root. The base case is simple: when the node v is the root, then follows from the definition of $S_q = \{\pi : \mathcal{A}_{\mathrm{ed}}(X_q, Y_\pi) < d_q\}$.

Suppose v is not root, and all its ancestors are successful. Let r be v's parent. Since r is successful, there exists anchor $q' \in Q_r \cap Z$. By inductive hypothesis, $(j_{q'}, l_{q'}) \in S_{q'}$. By Lemma 3.2, the triples (q, j_q, l_q) and $(q', j_{q'}, l_{q'})$ are compatible. Hence $(j_q, l_q) \in \check{S}_q$, and, since $\mathcal{A}_{\mathrm{ed}}(X_q, Y_{j_q,l_q}) = c_q < d_i$, we have that $(j_q, l_q) \in S_q$, completing the inductive proof. ☐

By the above lemma, the entire tree has the following structure (whp): a "top section" of the tree is composed of a set of successful nodes whose ancestors are also successful, and some of these nodes have unsuccessful children v, for which, by the above, we must have $\sum_{i\in U_v} c_i \ge (1-\epsilon)\sum_{i\in U_v} d_i$. Hence, we can partition the set I into $\{U_v\}_{v\in V}$ with the following property: each U_v, where $v \in V$ for some index set V, is either a successful singleton (leaf), or $\sum_{i\in U_v} c_i \ge \sum_{i\in U_v}(1-\epsilon)d_i$. Note that in the latter case, the edges (i, j_i, l_i), for $i \in U_v$, have cost at most $d_i + K_{\pi_i,(j_i,l_i)}$ (from the first phase). Hence, overall, the cost of edges (i, j_i, l_i) for $i \in U$ is at most:

$$\begin{aligned}
&\sum_{v\in V \text{ successful}} \sum_{i\in U_v} c_i + \sum_{v\in V \text{ not successful}} \sum_{i\in U_v} d_i + K_{\pi_i,(j_i,l_i)} \\
\le &\sum_{v\in V \text{ successful}} \sum_{i\in U_v} c_i + \sum_{v\in V \text{ not successful}} \sum_{i\in U_v} d_i + \alpha(\mathrm{ed}(X_i, Y_{\pi_i}) + \mathrm{ed}(X_i, Y_{j_i,l_i})) \\
\le &\sum_{v\in V \text{ successful}} \sum_{i\in U_v} c_i + \sum_{v\in V \text{ not successful}} \sum_{i\in U_v} \tfrac{(1+\alpha)c_i}{1-\epsilon} + \alpha\cdot \mathrm{ed}(X_i, Y_{j_i,l_i}) \\
\le &\sum_{i\in I}(2\alpha+\alpha^2)(1+2\epsilon)\cdot \mathrm{ed}(X_i, Y_{j_i,l_i}).
\end{aligned}$$

We've shown that the cost of the edges (i, j_i, l_i) is at most $(2\alpha+\alpha^2)(1+2\epsilon)$ times the real cost. Together with Lemma 3.2 this completes the proof of the correctness.

3.3 Analysis: run-time

Phase one of the algorithm takes the following times:

- To run the initial TextSearch x-vs-y calls: $O(b\cdot|L|\cdot k\cdot t(w))$;
- To run the MultiTextSearch y-vs-y calls: $O(|L|^2\cdot\sum_{j\in C} n/w\cdot t(w, s_j))$, where $s_j = \sum_{l\in L}|S_{j,l}|$;

- To add edges in the graph: $O(bk|L|)$.

In phase two, the main work is to estimate distances between q and $\breve{S}_q$ over all anchors. Hence, we need to show that the size of the sets $\breve{S}_q$, over all anchors q, is not too large overall. First we argue that the sets S_q are small, using the following claim:

Claim 3.5. *There are at most $c \cdot |J_w|/k$ indeces $j \in J_w$ such that there exists $j' \in \Delta\mathbb{Z} \cap [w], l \in L$ with $\mathcal{A}_{\text{ed}}(X_i, Y_{j+j',l}) < d_i$, with 99% probability, for some large constant $c > 0$.*

Proof. Consider all starting positions $j \in J_w$, and sort them according to the value of $\min_{j',l} \mathcal{A}_{\text{ed}}(X_i, Y_{j+j',l})$ where $j' \in \Delta\mathbb{Z} \cap [w]$ and $l \in L$. With at least 99% probability, the set C includes a $j \in J_w$ of rank at most $O(|J_w|/k)$ in that sorting. This implies the statement. □

The anchors from the root perform h calls to TextSearch, for a total time of $h \cdot O(n/w \cdot t(w))$. Now fix a node v at level $j \geq 1$ (with the convention that the root is at level 0), one of its anchors $q \in Q_v$, and let r be the parent of v. Recall that $\breve{S}_q$ consists of all (j, l) that are compatible with anything from $\cup_{q' \in Q_r} q' \times S_{q'}$. Consider one of r's anchors $q' \in Q_r$, and let $(j', l') \in S_{q'}$. We want to upper-bound the number of pairs $(j, l) \in J \times L$ such that (q, j, l) is compatible with (q', j', l'), i.e., $|j - j'| \leq \frac{1}{\epsilon} \cdot |q - q'|$. Note that $|q - q'|$ is upper bounded by the diameter of $U_{q'}$ (as $q, q' \in U_{q'}$), which is $m \cdot 2^{-j+1}$. Hence we must have $|j - j'| \leq \frac{1}{\epsilon} \cdot m \cdot 2^{-j+1}$. Thus all compatible (j, l) are covered by at most $2\frac{1/\epsilon \cdot m \cdot 2^{-j+1}}{w}$ blocks $[j : j + w]$ where $j \in J_w$, yielding the same upper bound on the number of TextSearch calls (for each fixed length $l \in L$). Each such call takes time $O(\frac{1}{\epsilon^2} t(w))$.

The total number of such TextSearch calls for a fixed anchor q is at most

$$|Q_r| \cdot \max_{q' \in Q_r} |S_{q'}| \cdot \frac{4m}{\epsilon w} 2^{-j} \cdot |L| \leq h \cdot c\frac{n/w}{k} \cdot \frac{4}{\epsilon} \frac{m}{w} 2^{-j} \cdot |L|.$$

Over all vertices v and their anchors, the runtime to run all the TextSearch calls becomes:

$$O(h \cdot n/w \cdot t(w)) + \sum_{j=1}^{O(\log b)} 2^j \cdot h \cdot \left(h \cdot c\frac{n/w}{k} \cdot \frac{4m}{\epsilon w} 2^{-j} \cdot |L|\right) \cdot O(\tfrac{1}{\epsilon^2} t(w))$$

$$\leq (\tfrac{\log n}{\epsilon})^{O(1)} \cdot \left(\tfrac{n}{w} \cdot t(w) + \tfrac{n}{wk} \cdot \tfrac{m}{w} \cdot t(w)\right) \leq (\tfrac{\log n}{\epsilon})^{O(1)} \cdot O(\tfrac{nm}{w^2 k} \cdot t(w)).$$

The third phase takes time $|S_x| \cdot \tilde{O}(n/w \cdot t(w) + |I| \cdot |J| \cdot |L| + n) = \tilde{O}(|S_x| \cdot (n/w \cdot t(w) + \frac{nm\beta}{w^2}))$.

Thus, the overall runtime is, for $k = \sqrt{m/w}$, $s = |S_x|$, and $\sum_{j \in C} s_j \leq b$, up to $(\frac{\log n}{\epsilon})^{O(1)}$ factor:

$$\frac{km}{w} \cdot t(w) + \frac{n}{w} \sum_{j \in C} t(w, s_j) + \frac{nm}{w^2 k} t(w) + s(\tfrac{n}{w} \cdot t(w) + \tfrac{nm\beta}{w^2}) = \frac{n\sqrt{m}}{w^{1.5}} \cdot t(w) + \frac{n}{w} \cdot t(w, \tfrac{m}{w}) + s(\tfrac{n}{w} \cdot t(w) + \tfrac{nm\beta}{w^2}).$$

Remark 3.6. *It is possible to adapt the algorithm to use a primitive $\mathcal{A}_{TS}$ for* TextSearch *only. Assuming that* TextSearch *on strings of lenth $O(w)$ runs in time $t(w)$, the algorithm runtime becomes:*

$$(\tfrac{n\sqrt{m\beta}}{w^{1.5}} \cdot t(w) + \tfrac{nm\beta}{w^2}) \cdot (\tfrac{\log n}{\epsilon})^{O(1)}.$$

For this, the phase 1 of the algorithm would pick C randomly from J, compute the distance between all $Y_{j,l}$, $j \in C$, and substrings of y using TextSearch *primitive.*

3.4 Proof of Block-matching Lemma

Proof of Lemma 3.2. To analyze $\mathrm{ed}(x,y)$, we considering the optimal alignment $A : [m] \to [n] \cup \{\perp\}$ that certifies $\mathrm{ed}(x,y)$. In particular, we have that, $A(i) < A(j)$ for any $i < j \in [m] \setminus A^{-1}(\perp)$, and

$$\mathrm{ed}(x,y) = \#\left\{ i \in [m] \setminus A^{-1}(\perp) : x[i] \neq y[A(i)] \right\} + 2\left| A^{-1}(\perp) \right|.$$

Note that the starting and ending positions of the blocks in y are always multiples of $\Delta \leq \epsilon/\beta \cdot w$. We call mini-blocks to be length-Δ blocks starting at positions $\Delta\mathbb{Z}$ in y. For each block $X_i = x[i : i + w]$, we define s_i to be starting point of the first mini-block containing $A(z) \neq \perp$ for $z \in [i : i + w]$. Similarly define $t_i \in S$ to be the last such mini-block. If s_i, t_i do not exist, we define $s_i = t_{i-1} + \Delta$ and $t_i = t_{i-1}$ (where $t_{-1} = -\Delta$ by convention). Note that $t_{i-1} \leq s_i$ for all i.

The starting point is match the block X_i to string $y[s_i : t_i + \Delta]$, i.e., to set $j_i = s_i$ and $l_i = t_i - s_i + \Delta$. In that case the LHS in Eqn. (1) is upper-bounded by $\mathrm{ed}(x,y) + 2\Delta \cdot b$ (each block may introduce error of $\leq 2\Delta$ due to rounding). However, we also need to make sure that the intervals $[j_i : j_i + l_i]$ satisfy the desired properties: they are disjoint, not too long, and are not too spread out. For this purpose, we modify j_i, l_i in a few steps below, controlling the incurred error.

We ensure disjointness as follows: if $j_{i-1} + l_{i-1} = j_i + \Delta$, we set $j'_i = j_i + \Delta$ and $l'_i = l_i - \Delta$ (and $j'_i = j_i, l'_i = l_i$ is otherwise). Note that, for some blocks i, it may now be that $l'_i = 0$, in which case X_i just matches to an empty block. We now have that $j'_{i-1} + l'_{i-1} \leq j'_i$. The incurred error per block is Δ.

Second, we ensure that the block lengths are valid: in particular, that each block length $l'_i \in w \pm \Delta E \cap [w/\epsilon]$. We now set l''_i using l'_i as follows: if $l'_i < w$ we round it up to nearest index in $w + \Delta \cdot E$, and otherwise take the minimum between rounding down in $w + \Delta \cdot E$ or $l''_i = w/\epsilon$. Let's analyze the incurred error. After changing from l'_i to l''_i, the LHS in Eqn. (1) increases by at most:

$$2 \cdot |l'_i - l''_i| = 2 \cdot |(l'_i - w) - (l''_i - w)| \leq 2\epsilon(1+\epsilon) \cdot |l'_i - w| \leq 2\epsilon(1+\epsilon) \cdot \mathrm{ed}(X_i, y[s'_i : t_i + \Delta]),$$

where we've used the fact that $\mathrm{ed}(X_i, y[j'_i : j'_i + l'_i]) \geq |l'_i - w|$. Hence, we get that:

$$\sum_i \mathrm{ed}(X_i, y[j'_i : j'_i + l''_i]) + (j'_i - (j'_{i-1} + l''_{i-1})) \leq \mathrm{ed}(x,y) + O(\Delta \cdot b) + 2\epsilon(1+\epsilon) \cdot (\mathrm{ed}(x,y) + O(\Delta \cdot b))$$

$$= (1 + O(\epsilon))\,\mathrm{ed}(x,y) + O(\epsilon n/\beta).$$

We are left with the final property to ensure: that $|j_{i'} - j_i| \leq \frac{1}{\epsilon}|i' - i|$ for all $i < i'$. Note that it is enough to ensure this for $i' = i+1$ (by triangle inequality). We ensure that by constructing j''_i by adjusting j'_i as necessary, iterating over $i \in I$ in order. For current i, suppose $j'_i - j''_{i-1} > w/\epsilon$. Then we simply set $j''_i = j''_{i-1} + w/\epsilon$ (note that $j''_{i+1} - j''_i = w/\epsilon \geq l''_i$, so keeping the same lengths is ok), and leave $j''_i = j'_i$ otherwise. Note that the LHS can increase only by at most w as some characters from X_i may lose their matches altogether. To account for this increase in cost, we "charge" this cost to

$$(j''_i - j''_{i-1} - l''_{i-1}) + \mathrm{ed}(X_{i-1}, Y_{j'_{i-1}, l''_{i-1}}) = w/\epsilon + (\mathrm{ed}(X_{i-1}, Y''_{j'_{i-1}, l_{i-1}}) - l''_{i-1}) \geq \tfrac{w}{\epsilon} - w \geq \tfrac{w}{2\epsilon}.$$

Since $\sum_i j''_i - (j''_{i-1} + l''_{i-1})$ remains the same overall, , the extra cost is only at most a factor of 2ϵ of the total cost when using indeces j'_i, l''_i. Hence the total cost of the changes here increases the cost by at most a factor of $1 + 2\epsilon$.

The final output indeces are (j''_i, l''_i). □

References

[AHVW16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the Symposium on Theory of Computing (STOC)*, 2016.

[AKO10] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, 2010.

[AO12] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM J. Comput.*, 41(6):1635–1648, 2012.

[BEK+03] Tuğkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of the Symposium on Theory of Computing (STOC)*, 2003.

[BES06] Tuğkan Batu, Funda Ergün, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 792–801, 2006.

[BI15] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Symposium on Theory of Computing (STOC)*, 2015.

[BJKK04] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 550–559, 2004.

[DGKS18] Debarati Das, Elazar Goldenberg, Michal Koucky, and Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, 2018.

[LV89] G. Landau and U. Vishkin. Fast parallel and serial approximate string matching. *J. Algor. 10*, pages 157–169, 1989. Preliminary version in ACM STOC'86.

[MP80] William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980.

[Mye86] Eugene W. Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[Sah08] Süleyman Cenk Sahinalp. Edit distance under block operations. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms.* Springer, 2008.

[Ukk85] E. Ukkonen. Algorithms for approximate string matching. *Information and Control 64, 100–118*, 1985.