# An Alternative Approach to Alignments

Vineet Joshi, Frank Lin, Prateek Tandon

## Analysis and Proofs

**Contents**

**Proofs**

### 1. Runtime Complexity Analysis

Let $i_1$ be the index in Text $T$ where we break off the matcing for the first exon.
Number of cells filled during the search for first exon $= m \times i_1$
Let $j_1$be the index of in Pattern where we break off the matcing for the first exon.

Similarly, $i_2$ be the index in Text $T$ where we break off the matcing for the second exon.
Number of cells filled during the search for second exon $= (m - j_1) \times (i_2 - i_1)$
Let $j_2$be the index of in Pattern where we break off the matcing for the second exon.

More generally,
$i_s$ be the index in Text $T$ where we break off the matcing for the $s^{th}$ exon.
Number of cells filled during the search for $s^{th}$ exon $= (m - j_{s-1}) \times (i_s - i_{s-1})$
Let $j_s$be the index of in Pattern where we break off the matcing for the $s^{th}$ exon.
Suppose, our pattern contains $l$ exons
Then, total number of cells filled will be

$$\begin{gathered} m \times i_1 + \sum_{s=2}^{l} (m - j_{s-1}) \times (i_s - i_{s-1}) \\ \leq m \times i_1 + \sum_{s=2}^{l} m \times (i_s - i_{s-1}) \\ = m \times i_1 + m \sum_{s=2}^{l} (i_s - i_{s-1}) \\ = m \times i_1 + m(n - i_1) \\ = mn \\ \leq 3mn \end{gathered}$$

$$(total\ number\ of\ cells\ filled\ by\ dynamic\ programming\ with\ affine\ gap\ penalties)$$

This reduces the computations by more than three times as compared to current method involving affine gap penalties. Since the size of n can be very large, this will reduce the computations significantly.
So, time complexity in our method = $O(m \times i_1) + O\left(\sum_{s=2}^{l}(m - j_{s-1}) \times (i_s - i_{s-1})\right) < O(mn)$

Average Case: $O(m \times i_1) + O\left(\sum_{s=2}^{l}(m - j_{s-1}) \times (i_s - i_{s-1})\right) < O(mn)$
Worst Case: $O(m \times i_1) + O\left(\sum_{s=2}^{l}(m - j_{s-1}) \times (i_s - i_{s-1})\right) = O(mn)$

But the probability of coming across the worst case is insignificant because there are a very small number of worst cases as compared to total cases.

In the average case the dynamic programming will stop before we reach the last index of the text T and number of cells filled in the average case will be far less than $m \times n$. The worst can only equal $m \times n$ when the entire pattern matches a one piece to the very end of the text.

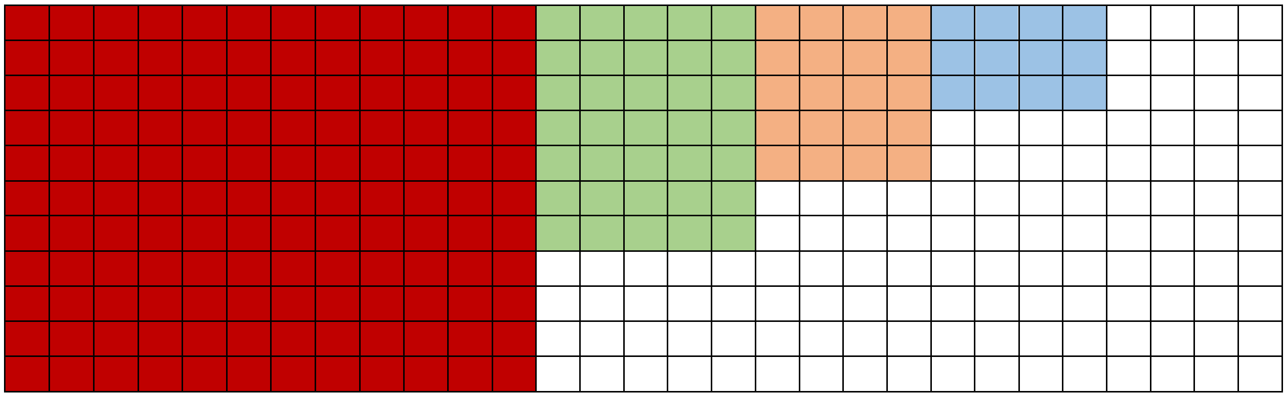

**Figure 1.** On an average case, the number of cells filled will be less than $m \times n$. This figure shows a typical comparison between cells filled in regular dynamic programming vs. the cells filled by our method in the average case. Here the white cells are the extra cells that will be filled by regular dynamic programming, when compared to our proposed method.

## 2. Further speed up using Ukkonen's method for evaluating columns in Edit Distance Matrix

As we discussed earlier, total number of columns filled can be lesser than the number of columns evaluated if we store columns in a suffix tree and just copy its value if we come across when we come across a repeating column.
Time to construct a suffix tree and for all its operations = $O(n)$
Let $q_i$ be the number of distinct columns evaluated during matching of exon $i$.

Using Ukkonen's approach…
Number of cells evaluated during matching of the first exon $= m \times q_1 \leq m \times i_1$
Number of cells evaluated during matching of the second exon
$$= (m - j_1) \times q_2 \leq (m - j_1) \times (i_2 - i_1)$$

More generally,
Number of cells evaluated during matching of the s$^{th}$ exon
$$= (m - j_{s-1}) \times q_s \leq (m - j_{s-1}) \times (i_s - i_{s-1})$$

Thus, evaluation for each exon is further sped up which gives an overall speed up

## 3. Suffix Tree Editing Time Complexity Proofs

We require a suffix tree of the text T in order to store edit distance columns based on their viable prefix from the root. We construct a suffix tree in time $O(n)$. We don't have to change this tree when we start matching for a new exon in the text even though our edit distances matrices change because each new text $T'$ is a suffix of our old text $T$. So, the suffix tree for $T$ in this case will also contain all paths of the suffix tree for $T'$ thus eliminating the need to create a new one. The columns stored in the nodes of T will not be useful as we will now be working with columns of length smaller than the previously evaluated columns ($m - j_{s-1}$ instead of $m - j_{s-2}$). One way to remove these unwanted columns will be to go across each node and delete them but this operation will take a time of $O(n)$ each time we start matching for a new exon. In order to prevent this additional work, we let the old columns stay in the nodes and when computing new columns when we reach a node, we check if the old column stored in the node is of length $m - j_{s-1}$. If there is such a column stored we simply copy it over to the edit distance matrix. If there is no column or a column of different length we compute a new column of length $m - j_{s-1}$and replace it with replace it with the old column. This way we don't perform any additional preprocessing steps over the suffix tree when searching for a new exon.

Now, as Ukkonen had proved that that the suffix tree traversal takes at most $2n$ steps, proportional to the filled edit distance columns ($n$ goto transitions and at most $n$ suffix transitions).

So, our method will have suffix tree traversals $\leq 2(i_1 - 0) + 2(i_2 - i_1) + \cdots + 2(i_l - i_{l-1}) \leq 2n$

Time complexity of operations associated with suffix tree $\leq \underset{\text{tree building}}{O(n)} + \underset{\text{suffix tree traversals}}{O(n)}$
$= O(n)$

## 4. Total Time Complexity For Evaluating All The Cells

$$= O(n) + O(mq_1) + O\left(\sum_{s=2}^{l} (m - j_{s-1}) \times q_s\right)$$

(Suffix tree operations + cells evaluated for first exon + cells evaluated for remaining exons)

Since, Ukkonen states that the upper bound on $q_s$

$$\leq O\left(\min\left((i_s - i_{s-1}), \frac{(i_s - i_{s-1})}{n - i_{s-1}} (m - j_{s-1})^{k_s+1} |\Sigma|^{k_s}\right)\right),$$

where $k_s$ is the mismatches encountered during matching of exon $s$. We can use this upper bound as matching for each exon is exactly the same as using Ukkonen's method for the substring of T and exon of P.

We divide the second term in the min(, ) by $\dfrac{(i_s - i_{s-1})}{n - i_{s-1}}$ because it gives us estimate for the distinct columns evaluated when compared with the entire remaining length of the Text. We only want number of distinct columns evaluated in the region $\mathrm{i_s - i_{s-1}}$ (average length of introns), so the scaledown.

Therefore, total time complexity (Let's call it $(X)$ )

$$= O(n) + O\left(m \times \min\left(i_1, \frac{i_1}{n} m^{k_1+1} |\Sigma|^{k_1}\right)\right)$$
$$+ O\left(\sum_{s=2}^{l} (m - j_{s-1}) \times \min\left((i_s - i_{s-1}), \frac{(i_s - i_{s-1})}{n - i_{s-1}} (m - j_{s-1})^{k_s+1} |\Sigma|^{k_s}\right)\right)$$
$$\leq O(n) + O(m \times i_1) + O\left(\sum_{s=2}^{l} (m - j_{s-1}) \times (i_2 - i_1)\right)$$

So the worst case time complexity remains
$= O(n) + O(m \times i_1) + O\left(\sum_{s=2}^{l} (m - j_{s-1}) \times (i_2 - i_1)\right)$
$\cong O(m \times i_1) + O\left(\sum_{s=2}^{l} (m - j_{s-1}) \times (i_s - i_{s-1})\right)$
$\leq O(mn)$
But the average case would perform better than $O(m \times i_1) + O\left(\sum_{s=2}^{l} (m - j_{s-1}) \times (i_s - i_{s-1})\right)$
To summarize,

$$O(X) \leq O(m \times i_1) + O\left(\sum_{s=2}^{l} (m - j_{s-1}) \times (i_s - i_{s-1})\right) \leq O(mn) \leq 3 \times O(mn)^*$$

$*$ (3 matrices for affine gaps)

## <u>5. Statistical Significance Analysis</u>

Average length of exons is around 150 $bp$[]
Number of k-differences that can occur in this length depends on the sequencing methods used to sequence the mRNA and Presence of Single point mutations. Combined, these won't typically exceed 6%.
We set a threshold "L" of 100 $bp$. So we must allow for 6 differences during matching of this region.
We can have 3 different kinds of mutations: Deletions, Substitutions and Insertions.
When there are exactly 6 differences, we can divide these among insertions, deletions and substitutions in 28 different ways (using pirates and gold coins problem). For each case we then calculate the different ways these mutations can happen in the 100 bp region. For example when there are x substitutions, y deletions and z insertions in the pattern, the different mutations will be equal to

$\binom{100}{x}\binom{100-x}{y}\binom{100-x-y}{z}4^z$ ($4^z$ because insertions can add in A, C, G or T)

Total number of mutations with exactly 6 differences will be

$$= \sum_{x,y,z>0\ .x+y+z=6} \binom{100}{x}\binom{100-x}{y}\binom{100-x-y}{z}4^z$$
$$\leq \sum_{x,y,z>0\ .x+y+z=6} \binom{100}{2}\binom{100}{2}\binom{100}{2}4^2$$
*(due to symmetry)
$$= 28\binom{100}{2}\binom{100}{2}\binom{100}{2}4^2$$

Since a similar analysis for 1,2,3,4 and 5 differences will have a smaller number of mutations than for 6 differences (6 differences will make the highest combination of different mutations), we can say that the total number of mutations for up to 6 differences will have the upper bound

$$= 6 \times 28 \binom{100}{2}\binom{100}{2}\binom{100}{2} 4^2$$

Size of Human genome $\sim 3 \times 10^9\ bp$[]

Therefore, the probability of seeing 100 $bp$ length aligning by chance is

$$< \frac{6 \times 28 \binom{100}{2}\binom{100}{2}\binom{100}{2} 4^2 \times 3 \times 10^9}{4^{100}}$$
$$\sim 6.086491\mathrm{e} - 37$$

Since the probability of finding an alignment of over 100 bp by chance alone is very small, we can say with confidence that whatever alignments we get bears statistical relevance to the biological question at hand.

## <u>6. Comparison of q and n For Our Purposes</u>

As earlier, we assume average length of exons is around 150 $bp$ [13].
Average number of exons in a cDNA ≈ 9 []
Average Length of introns ≈ $3500\ bp$ [14]
Sequencing errors by modern methods for NGS ≈ 1%[]
Frequency of SNPs ≈ 1 in every 600 bp[]
It is safe to assume k for 150 bp exon will be ≈ 2.5

Therefore total number of distinct columns evaluated

$$\approx 9 \times \text{columns evaluated for each exon}$$
$$\leq 9 \times \frac{12}{5}(1350)^{2+1}\frac{3500}{3 \times 10^9}\ (4+1)^2$$
$$\cong 1.27348 \times 10^8$$
$$\cong 0.05 \text{ times the length of the human genome}$$

So in this case q<<n.

[Reference: http://www.cs.cmu.edu/~epxing/Class/10810-05/Lecture10.pdf]