Regular Expression
Constrained Sequence Alignment

By Abdullah N. Arslan
Department of Computer science
University of Vermont

Presented by Tomer Heber & Raz Nissim
Motivation

- When comparing two proteins, it may be important to take into account a common specific structure.
- Families of similar protein sequences include a conserved region called a motif.
- The PROSITE database is a collection of these motifs, represented as regular expressions.
- Our problem concentrates on sequence alignment of proteins that contain these motifs and need to be aligned according to them.
The Optimal Sequence Alignment Problem:

• **String Alignment:**
  “Given two strings s and t, a global alignment is obtained by inserting spaces into s and t so that the characters of the resulting strings can be put in one-to-one correspondence to each other.”

H - A S K E L L
P A A - C A - L

* Two characters correspond/match each other.

• The **Optimal** Sequence Alignment” is a string alignment that has the maximum number of characters that correspond to each other.
Example:

Given two strings:
S1 = TGFPSVGVTKTKDDA
S2 = TFSVAKDDDGKA

The **Optimal** Sequence Alignment for this two strings is 8.

*By adding the gaps to both strings, 8 characters correspond to each other in both strings: T,F,S,V,K,D,D,A. There are no alignments of 9 or more corresponding characters, thus this is the optimal solution.*
**Definition:** Given strings $S_1, S_2$ and the regular expression $R$, RECSA is the problem of finding the optimal global sequence alignment between $S_1$ and $S_2$ where there exists $s_1$ subset of $S_1$ and $s_2$ subset of $S_2$ such that the following constraints are satisfied:

1. Substring $s_1$ is aligned with substring $s_2$ (local alignment).
2. Both $s_1$ and $s_2$ match the expression $R$, $s_1, s_2 \in L(R)$. 

**Regular Expression Constrained Sequence Alignment (RECSA)**
Example:

Given two strings and a regular expression:
S\(_1\) = TGFPSVGKT\(K\)DDA
S\(_2\) = TFSVAK\(D\)DDDGKSA
\(R = (G + A)\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma\)\(G\)K(S + T)

The **Optimal** global sequence alignment Constrained by the regular expression \(R\) is 4:

```
T - - - | G F P S V G K T | K D D D A
| T F S V A K D D D D G K S - - - A
```

\(*s_1 = GFPSVGKT\) is a substring of S\(_1\), and s\(_2\) = AKDDDGKS is a substring of S\(_2\). s\(_1\) and s\(_2\) are aligned with each other and both match the regular expression \(R\). 4 characters correspond/match to each other in strings S\(_1\) and S\(_2\).\)
Needleman–Wunsch algorithm

• This algorithm is used for computing the maximum global score given two strings of size n and m in O(nm).

Given the strings $S_1$, $S_2$ where $|S_1| = n$, $|S_2| = m$, and a weight function $\gamma : R \to R$. The dynamic formulation to compute the maximum global alignment score is:

$$
\mu_{i,j} = \begin{cases} 
\mu_{i-1,j} + \gamma(S_1[i] \to \varepsilon), \\
\max \left( \mu_{i-1,j-1} + \gamma(S_1[i] \to S_2[i]), \mu_{i,j-1} + \gamma(\varepsilon \to S_2[j]) \right)
\end{cases}
$$

$\mu_{0,0} = 0$

$\mu_{n,m}$ = answer
Needleman–Wunsch algorithm – Continued

• $\mu_{i,j}$ = The optimal global alignment score of the substrings $S_1[1..i], S_2[1..j]$.

• Insertion or Deletion are referred to as gaps.

• $\gamma(S_1[i] \rightarrow \varepsilon)$ = The price of adding a gap to S2.

• $\gamma(\varepsilon \rightarrow S_2[j])$ = The price of adding a gap to S1.

• $\gamma(S_1[i] \rightarrow S_2[i])$ = The price of a match or a mismatch of a character.

Question: Can we improve the runtime for finding an optimal global alignment?

Hint: LCS and optimal sequence alignment problems are equivalent.
Reminder

Given two strings $S_1$, $S_2$ and a regular expression $R$, we want to find two substrings $s_1$ and $s_2$ (substrings of $S_1$ and $S_2$) that uphold these constraints:

1. $s_1$ is aligned with $s_2$.
2. Both $s_1$ and $s_2$ match the regular expression $R$.

Previous example:

$$R = (G + A)ΣΣΣΣGΚ(S + T)$$

\[
\begin{array}{ccccccccccccccc}
\end{array}
\]
Review of Automata (State Machines)

An **automaton** is represented by the 5-tuple, where:

- **Q** - Set of **states**.
- **Σ** - Finite set of **symbols**, that we will call the **alphabet** of the language the automaton accepts.
- **δ** - **Transition function**, that is \( \delta : Q \times \Sigma \to Q \).
- **q₀** - **Start state**, that is, the state in which the automaton is when no input has been processed yet (Obviously, \( q₀ \in Q \)).
- **F** - Set of states of **Q** (i.e. \( F \subseteq Q \)), called **accept states**.
Review of Automata (cont.)

• There exists a strong equivalence: for every regular language, there is a finite state automaton (DFA or NFA), and vice versa.

• Example- This automaton accepts the regular language: \[ A \ (C + G)^* \ (S + T) \]
Example of N x N “multiplication automaton”:
Constructing our **Weighted Finite Automaton**

We construct $M$ from a given regular expression $R$ in several steps:

(1) First, given a regular expression $R$ we construct a **nondeterministic finite automaton** $N=(Q,\Sigma,\delta,q_0,F)$, with no $\varepsilon$-moves, such that $L(N)=L(R)$. $N$ accepts only the set of strings described by the regular expression $R$. 
Constructing our **Weighted Finite Automaton** (cont.)

(2)

We define a weighted $N \times N$ automaton as the finite automaton $M = (Q^M, W^M, \Sigma^M, q_0^M, F^M)$ which we construct as follows:

- $Q^M = Q \times Q$ is the set of states. Each state of $M$ corresponds to a pair of states in $N$. $M$ remembers in each state what part of the regular expression has been seen in $S_1$ and $S_2$.

- $W^M : Q^M \rightarrow \mathbb{R}$ is a function that assigns real weights to each state in $Q^M$. Initially all weights are $-\infty$. 
Step 2 (cont.)

• $\Sigma^M = (\Sigma \times \Sigma) \setminus \{\varepsilon \rightarrow \varepsilon\}$. The alphabet for $M$ is the set of edit operations which does not include $\varepsilon \rightarrow \varepsilon$.

• $q^M_{00} = (q_0, q_0)$ is the start state whose weight is 0 and always stays 0.

• $F^M = (F \times F)$. Is the set of final states. If $M$ is in a final state then $M$ has processed an alignment that satisfies the regular expression constraint (there are substrings $s_1$ of $S_1$ and $s_2$ of $S_2$ that are aligned and both $s_1$ and $s_2$ take $N$ to final states.
Step 2 – The Transition Function

\[ \delta^M : \Sigma^M \times Q^M \rightarrow Q^M \]. \( M \) moves on edit operations:

- For \( x \neq \epsilon \), \( \delta^M \left( (p, q), x \rightarrow \epsilon \right) = \{(p', q) \mid p' \in \delta(p, x)\} \)
- For \( y \neq \epsilon \), \( \delta^M \left( (p, q), \epsilon \rightarrow y \right) = \{(p, q') \mid q' \in \delta(q, y)\} \)
- For \( x \neq \epsilon, y \neq \epsilon \), \( \delta^M \left( (p, q), x \rightarrow y \right) = \{(p', q') \mid p' \in \delta(p, x), q' \in \delta(q, y)\} \)

• Once an alignment satisfies the regular expression constraint, i.e. once a final state is reached in \( M \), the rest of the alignment does not alter the satisfaction of the constraint. 

(\( M \) has the option of staying in a final state on any input after that final state is reached)
The main idea

<table>
<thead>
<tr>
<th></th>
<th>S₂[1]</th>
<th></th>
<th>S₂[j]</th>
<th></th>
<th>S₂[m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S₁[1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S₁[i]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S₁[n]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- M_{i-1,j-1}
- M_{i-1,j}
- M_{i,j}
- S₁[i] → S₂[j]
- S₁[i] → ε
- ε → S₂[j]
Denotations

• For any given weighted N × N automaton M we denote by $M^{x\to y}$ for any $X \to Y \in \Sigma^M$ a copy of the automaton M after making the move on $X \to Y$.

• Given two weighted N × N automata M1 and M2, we define a commutative and associative operation max-M such that max-M{M1, M2} is a weighted N × N automaton M with state weights calculated as follows:

For all $(p, q) \in Q^M$

$$W^M(p, q) = \max\{W^{M_1}(p, q), W^{M_2}(p, q)\}.$$
Denotations (cont.)

• We denote by $S[i..j]$ the substring of $S$ from positions $i$ to $j$, $i \leq j$. Let $S[i]$ denote the $i$-th symbol of string $S$.

• We denote by $M_{i,j}$ the optimally weighted automaton for $S_1[1...i]$ and $S_2[1...j]$. 
The Algorithm

• Let $|S_1| = n$, $|S_2| = m$ with $n \geq m$, and let $N$ be a non-deterministic automaton with no $E$-moves equivalent to regular expression $R$, and let $M$ be a weighted $N \times N$ automaton constructed from $N$ as we described.

• For all $i, j$, $0 \leq i \leq n$, $0 \leq j \leq m$, both $M_{i,0}$ and $M_{0,j}$ are identical weighted $N \times N$ automaton whose state-weights are all $-\infty$ (except for the weight of the start state $(q_0, q_0)$ which is always 0).
Algorithm (cont.)

• Optimal automata $M_{i,j}$ are computed with this formula:

$$M_{i,j} = \max_M \left\{ M_{i-1,j}^{S_1[i] \rightarrow \epsilon}, M_{i-1,j-1}^{S_1[i] \rightarrow S_2[j]}, M_{i,j-1}^{\epsilon \rightarrow S_2[j]} \right\}$$

• $M_{i-1,j}^{S_1[i] \rightarrow \epsilon}$ is the copy of the optimally weighted automaton $M_{i-1,j}$ after making the move $S_1[i] \rightarrow \epsilon$. (we’ll see an example later on...
Algorithm (cont.)

• After we finish calculating $M_{i,j}$ for all $i$, $j$, we’ll focus on $M_{n,m}$ (the optimally weighted automaton for substrings $s_1=S_1$, $s_2=S_2$)

• We output the weight:

$$\max \{ W_{n,m}^M(p, q) \mid (p, q) \in F_{n,m}^M \}$$

(the maximum of the weights of the accepting states in $M_{n,m}$.)
Calculating the weights

- Upon making a move, these two steps must be followed:

*Step 1.* For all \((p, q)\), if there exists \((p', q')\) such that \((p, q) \in \delta^M((p', q'), x \rightarrow y)\) and \(W^M(p', q') \neq -\infty\) then \(W^M(p, q) = \max\{W^M(p', q') + \gamma(x \rightarrow y) \mid (p, q) \in \delta^M((p', q'), x \rightarrow y)\}\). New active states are those that are reachable from the active states on input \(x \rightarrow y\). The weights of the active states are updated using the weight \(\gamma(x \rightarrow y)\) of the edit operation \(x \rightarrow y\), and the weights of the states through which new states are reached.

*Step 2.* For all \((p, q)\), if there does not exist \((p', q')\) such that \((p, q) \in \delta^M((p', q'), x \rightarrow y)\) and \(W^M(p', q') \neq -\infty\) then \(W^M(p, q) = -\infty\). After the move some previously active states may become inactive. This occurs when a suffix of \(S_1\) (or \(S_2\)) partially matching the regular expression \(R\) no longer partially matches \(R\) when the suffix is extended with \(x\) (or \(y\)). If a state is no longer active then its weight is set to \(-\infty\).
Calculating the weights (example)

\[ \gamma(\varepsilon \rightarrow a) = -20 \]

• Step 1:
(q1,q3) gets the value:

\[
\max \{ W^M(q0, q1) + \gamma(\varepsilon \rightarrow a), W^M(q1, q1) + \gamma(\varepsilon \rightarrow a) \}
\]

In this case, \(\max\{80-20, 90-20\} = 70\).
Calculating the weights (example cont.)

• Step 2:
  If \((p,q)\) is reachable with the move \(b \rightarrow a\) only from parents that have a weight of \(-\infty\), its weight is updated to \(-\infty\) as well.

• It is imperative that this step is done only after step 1 is completed, since some updates could be missed.
Optimally Weighted - definition

• \( M_{i,j} \) is optimally weighted for \( S1[1..i] \), and \( S2[1..j] \), if the following two properties hold:

\textit{Property 1:} For all final states \((p, q) \in F^{M_{i,j}}\), \( W^{M_{i,j}}(p, q) \) is the maximum alignment score between \( S1[1..i] \) and \( S2[1..j] \) over all alignments that include a region in which substring \( s_1 \) of \( S1[1..i] \) is aligned with substring \( s_2 \) of \( S2[1..j] \), and \( s_1, s_2 \in L(R) \), i.e. \( N \) on input \( s_1 \) enters final state \( p \in F \), and on input \( s_2 \) enters final state \( q \in F \). If there do not exists such \( s_1 \) and \( s_2 \) then \( W^{M_{i,j}}(p, q) = -\infty \) ((\( p, q \) is an inactive state in \( M_{i,j} \)).

\textit{Property 2:} For all states \((p, q) \notin F^{M_{i,j}}\), \( W^{M_{i,j}}(p, q) \) is the maximum alignment score between \( S1[1..i] \) and \( S2[1..j] \) over all alignments that include a region in which \( s_1 \) is aligned with \( s_2 \), and \( s_1 \) is a suffix of \( S1 \), and \( s_2 \) is a suffix of \( S2 \), and \( N \) on input \( s_1 \) enters state \( p \in Q \), and on input \( s_2 \) enters state \( q \in Q \). If there do not exists such \( s_1 \) and \( s_2 \) then \( W^{M_{i,j}}(p, q) = -\infty \) ((\( p, q \) is an inactive state in \( M_{i,j} \)).
Proof of correctness

• **Claim:**

For all \( i,j \), \( M_{i,j} \) computed by

\[
M_{i,j} = \max_{M} \left\{ M_{i-1,j}^{S_1[i] \rightarrow e}, M_{i-1,j-1}^{S_1[i] \rightarrow S_2[j]}, M_{i,j-1}^{e \rightarrow S_2[j]} \right\}
\]

is optimally weighted.

• **Proof:** By induction on nodes \((i,j)\)...
Proof of correctness (cont.)

• Base case:  
  \( i = 0, \) for all \( j \) the weights of \( M_{0,j} \) are \(-\infty\) (or 0). The two properties hold, and the claim is true.

• Assuming the claim is true for \( M_{i-1,j}, M_{i,j-1} \) and \( M_{i-1,j-1} \), we will show that the following automata are optimal:

1. \( M_{i-1,j}^{S_1[i] \rightarrow \epsilon} \) when \( ((i - 1, j), (i, j)) \) is a required arc for the alignments,
2. \( M_{i, j-1}^{\epsilon \rightarrow S_2[j]} \) when \( ((i, j - 1), (i, j)) \) is a required arc for the alignments,
3. \( M_{i-1, j-1}^{S_1[i] \rightarrow S_2[j]} \) when \( ((i - 1, j - 1), (i, j)) \) is a required arc for the alignments.
Proof of correctness (Intuition)

• $M_{i,j}$‘s values are computed by taking the maximal of its parents’ values for each state, and adding to them the score of one edit action.

• Optimality of $M_{i,j}$ follows from its parents’ assumption of optimality and since an optimal constrained alignment at node (i,j) uses one of these arcs, and we compute maximum scores for all possible optimal alignments.
Run-Time Analysis

• Computing each $M_{i,j}$ takes time $O(r)$ where $r$ is the size of the transition function of $M$.

* We note that $r = O(t^4)$ where $t$ is the number of states in automaton $N$ accepting the language $L(R)$. This is because $M$ has $O(t^2)$ states and the transition function of a nondeterministic automata is of size $O(\text{states}^2)$

• We need to compute $O(nm)$ automata and therefore we have a running time of $O(nmr)$. 
Optimal alignment path reconstruction

• In order to reconstruct the optimal path, we can store for each active state of each automaton, the neighboring automaton and the state in that automaton from which the optimal score is obtained.

• This is possible by altering the Max-M function.

• Starting at $M_{n,m}$, we generate the alignment path in reverse.
Questions??

• A detailed example will be uploaded for further reference.

Thanks 😊