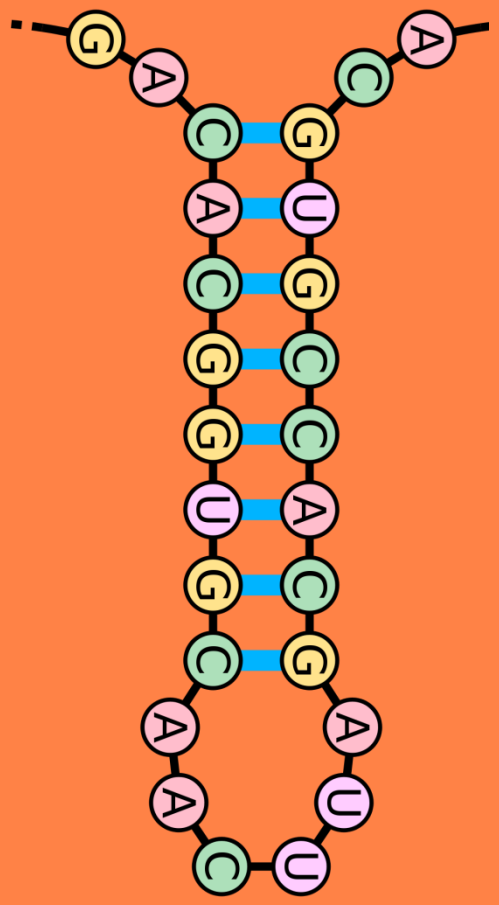


Sparse RNA folding: Time and Space Efficient Algorithms

Rolf Backofen, Dekel Tsur, Shay Zakov, Michal Ziv-Ukelson

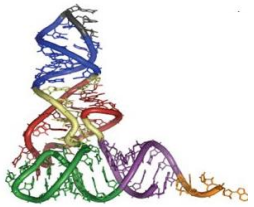
Published in 2011

Presented By

Nabila Shahnaz Khan

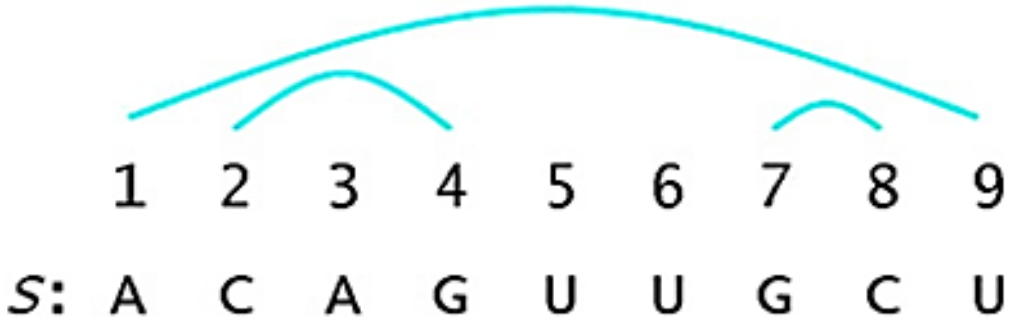
Time Optimization of RNA Single Strand Folding

- Classical algorithms are Zuker's $\mathbf{O(n^4)}$, Nussinov $\mathbf{O(n^3)}$ etc.
- In 2007, Wexler et al. proposed a solution which requires $\mathbf{O(n^2\psi(n))}$ time where $\boldsymbol{\psi(n)}$ represents a constant
- In this paper, $\mathbf{O(n^2\psi(n))}$ has been represented as $\mathbf{O(nZ)}$ where $\mathbf{n \leq Z < n^2}$
- This paper introduced a faster approach with run time complexity of $\mathbf{O(n^2+PZ)}$, here $\mathbf{P \leq n/2}$

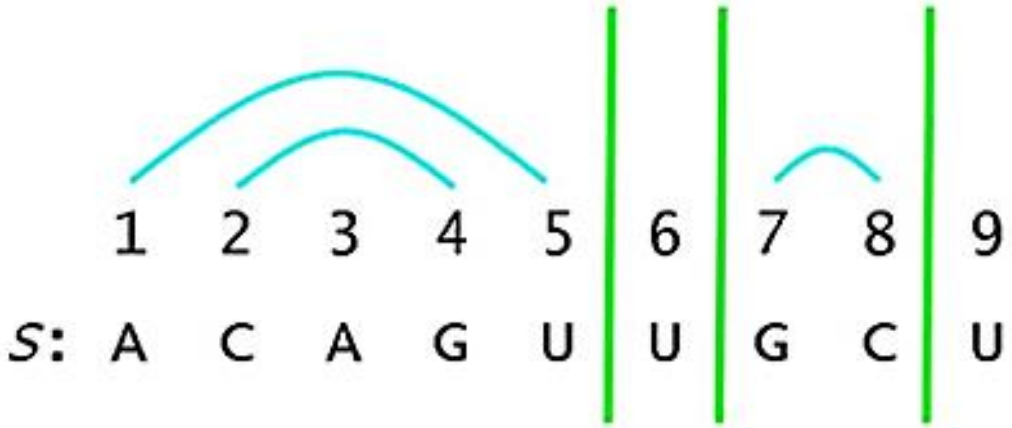


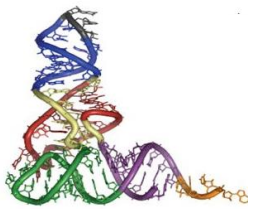
Time Optimized SSF on Base-pair Maximization

- **L(i, j) is the folding for S(i, j) with maximum number of base-pairs**
- **L^C(i, j) is co-terminus solution**



- **L^P(i, j) is partitionable solution**

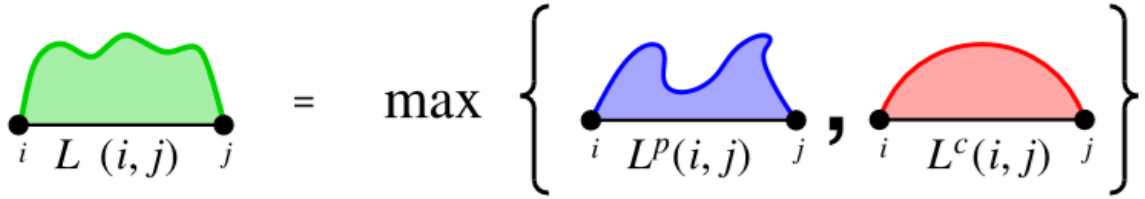


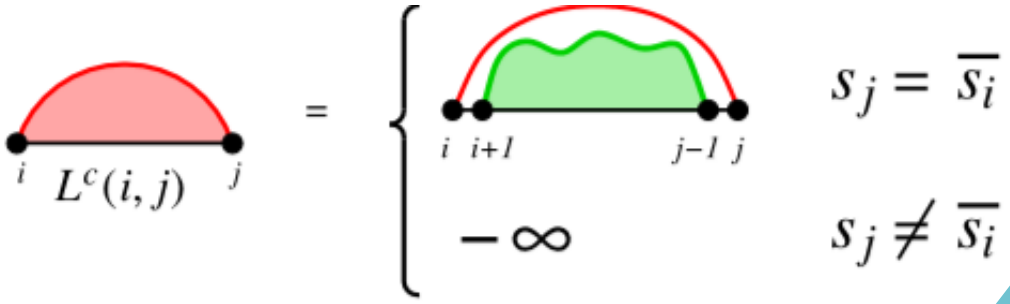


Time Optimized SSF on Base-pair Maximization

L(i, j) is the folding for S(i, j) with maximum number of base-pairs

$$L(i, j) = \max\{ L^C(i, j), L^P(i, j)\}$$

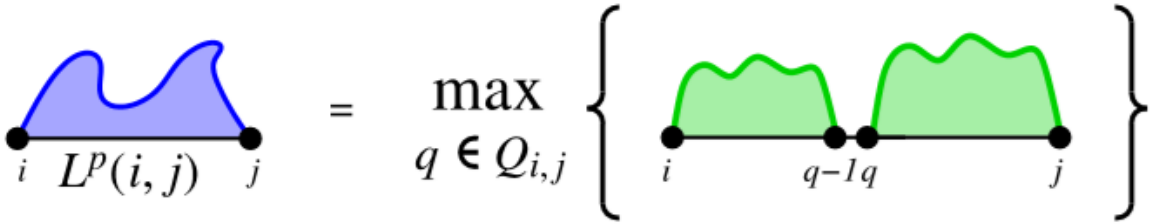




$$L^c(i, j) = \begin{cases} L(i+1, j-1) + 1, & s_j \text{ and } si \text{ are complementary} \\ -\infty, & otherwise \end{cases}$$

$$L^P(i, j) = \max\{L(i, q-1)+L(q, j)\}$$

$q: i < q \leq j$



Steps Followed for Reducing the Time Complexity

Applying Inverse Triangle Inequality

01

Using OCT sub-instances O(nZ)

02

Further reducing split points using STEP $O(n^2+PZ)$

03

Applying Inverse Triangle Inequality Property

- **A scoring scheme sustains the inverse triangle inequality property if for every sub-instance S(i, j) and for every split-point q ∈ Q(i, j),**

$$L(i, j) \geqslant L(i, q-1) + L(q, j)$$

- **The SSF scoring scheme shown before follows this equation, so it sustains the inverse triangle inequality property**
- **The computational improvements are done based on this property**

Using OCT Sub-instances

- **Time complexity bottleneck in SSF algorithms is dictated by the consideration of O(n) split-points q in the computation of $L^P(i, j)$**

- **Based on inverse triangle inequality, Wexler et al. observed that it is sufficient to examine only a subset of the split-points in order to compute $L^P(i, j)$**

- **A sub-instance $S_{i,j}$ is optimally co-terminus (OCT) if every optimal folding of $S_{i,j}$ is co-terminus**

$$L(i, j) = L^c(i, j) > L^p(i, j)$$

- **Any sub-instance of length 1 is an OCT**

Using OCT Sub-instances

- **For a sub-instance $S_{i,j}$ with $j > i$, call a split-point $q \in Q_{i,j}$ for which**

$$L^p(i, j) = L(i, q-1) + L(q, j)$$

- **For every sub-instance $S_{i,j}$ with $j > i$, there is an optimal split-point $q \in Q_{i,j}$ such that $S_{q,j}$ is an OCT** [Proof given in last slide]

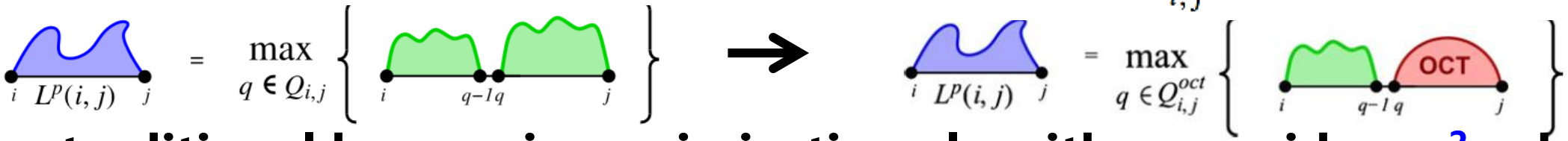
- **Considering such split points, subset of split-points with respect to $S_{i,j}$ is,**

$$Q_{i,j}^{oct} = \{q \in Q_{i,j} \colon S_{q,j} \text{ is an OCT}\}$$

Using OCT Sub-instances

- **So, the split-points are restricted**

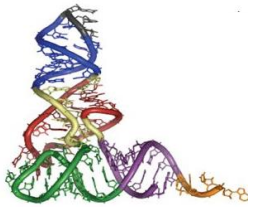
$$L^p(i,j) = \max_{q \in Q_{i,j}} \{L(i,q-1) + L(q,j)\} \rightarrow L^p(i,j) = \max_{q \in Q^{oct}_{i,j}} \{L(i,q-1) + L(q,j)\}$$



- **The traditional base-pair maximization algorithm considers n^2 sub-structures and for each substructure, average number of split-points is n. So, runtime $O(n^3)$**

- **After improvement, average number of split-points is $\Theta(|Q^{oct}_{i,j}|)$, so complexity $O(nZ)$**

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} |Q^{oct}_{i,j}| \leqslant \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} |Q^{oct}_{1,j}| \leqslant \sum_{i=1}^{n-1} Z < nZ.$$



Further reducing split points using STEP $O(n^2+PZ)$

- A sub-instance $S_{i,j}$, with $j > i$ is a **STEP** if $L(i, j) > L(i, i)+L(i+1, j)$ [means split-point $i+1$ can not give optimal solution]

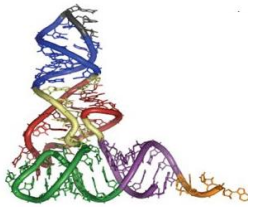
- It implies that if $S_{i,j}$ is a **STEP**, i is paired in every optimal folding of $S_{i,j}$

- For any sub-instance $S_{i,j}$ such that $j > i$, there is an optimal split-point q with respect to $S_{i,j}$ such that either $q = i+1$ (prefix not STEP), or $S_{i,q-1}$ is a STEP and $S_{q,j}$ is an OCT

- For this case, the subset of split-points with respect to $S_{i,j}$ is,

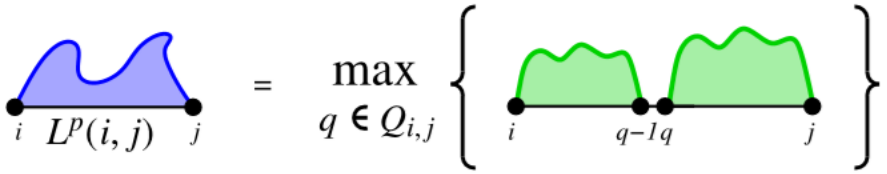
$$Q_{i,j}^{\text{step-oct}} = \{q \in Q_{i,j}:\ S_{i,q-1} \text{ is a STEP and } S_{q,j} \text{ is an OCT}\}$$

- So, $Q_{i,j}^{step-oct} < Q_{i,j}^{oct}$

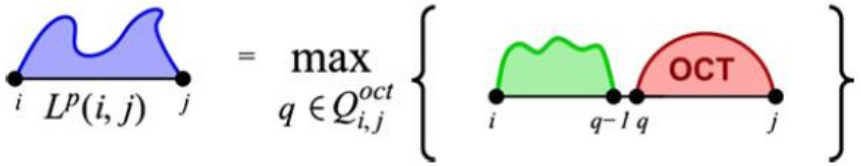


Further reducing split points using STEP $O(n^2+PZ)$

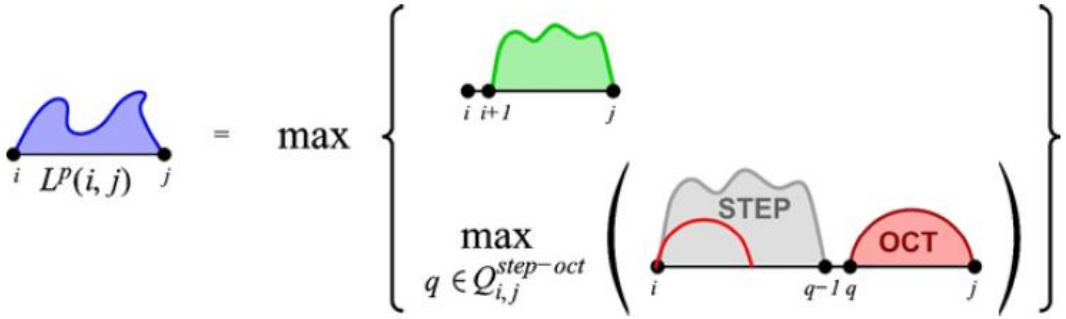
$$L^p(i,j) = \max_{q \in Q_{i,j}} \{L(i,q-1) + L(q,j)\}$$



$$L^p(i,j) = \max_{q \in Q^{oct}_{i,j}} \{L(i,q-1) + L(q,j)\}$$



$$L^p(i,j) = \max_{q \in \{i+1\} \cup Q^{\text{step-oct}}_{i,j}} \{L(i,q-1) + L(q,j)\}$$



Let P = P(S) denote the maximum number of STEP sub-instances of S such that P < n

Forward RNA Folding Algorithm [$O(n^2+PZ)$]

Algorithm 3: Forward RNA folding

input : An RNA string $S = s_1 s_2 \cdots s_n$
output: $L(1, n)$

```
for i ← n down to 1 do
    set explicitly the values of M[i, i − 1] and M[i, i] to the solutions for the corresponding trivial sub-instances;
    mark S_{i,i} as an OCT;
    set M[i, j] ← M[i, i] + M[i + 1, j] for all i < j ⩽ n;
    for j ← i + 1 to n do
        compute L^c(i, j) = f(M[i + 1, j − 1], s_i, s_j);
        if M[i, j] < L^c(i, j) then
            set M[i, j] ← L^c(i, j);
            mark S_{i,j} as an OCT;
        if S_{i,j} is a STEP then
            for all j′ s.t. S_{j+1,j′} is an OCT do
                set M[i, j′] ← max{M[i, j′], M[i, j] + M[j + 1, j′]};
    discard from memory the values in all entries in row i + 1 of M that do not correspond to OCT sub-instances;
return M[1, n].
```

Explanation with Example

- **Suppose, i=3**

$O(n^2+PZ)$

Algorithm 3: Forward RNA folding

input : An RNA string $S = s_1 s_2 \cdots s_n$
output: $L(1, n)$

for $i \leftarrow n$ *down to* 1 **do**
  set explicitly the values of $M[i, i-1]$ and $M[i, i]$ to the solutions for the corresponding trivial sub-instances;
  mark $S_{i,i}$ as an OCT;
  set $M[i, j] \leftarrow M[i, i] + M[i+1, j]$ for all $i < j \leqslant n$;
  for $j \leftarrow i+1$ *to* n **do**
    compute $L^c(i, j) = f(M[i+1, j-1], s_i, s_j)$;
    if $M[i, j] < L^c(i, j)$ **then**
      set $M[i, j] \leftarrow L^c(i, j)$;
      mark $S_{i,j}$ as an OCT;
    if $S_{i,j}$ *is a STEP* **then**
      for all j' *s.t.* $S_{j+1,j'}$ *is an OCT* **do**
        set $M[i, j'] \leftarrow \max\{M[i, j'], M[i, j] + M[j+1, j']\}$;
  discard from memory the values in all entries in row $i+1$ of M that do not correspond to OCT sub-instances;
return $M[1, n]$.

Columns:
Left -right

Rows:
Bottom-up

		1	2	3	4	5	6	7	8	9
		A	C	A	G	U	U	G	C	A
1	A									
2	C									
3	A		0	0	0	0	0	0	1	2
4	G			0	0	0	0	0	1	2
5	U					0				
6	U						0			2
7	G							0	1	
8	C								0	
9	A									0

OCT candidate list: (9,9), ……., (3,3)

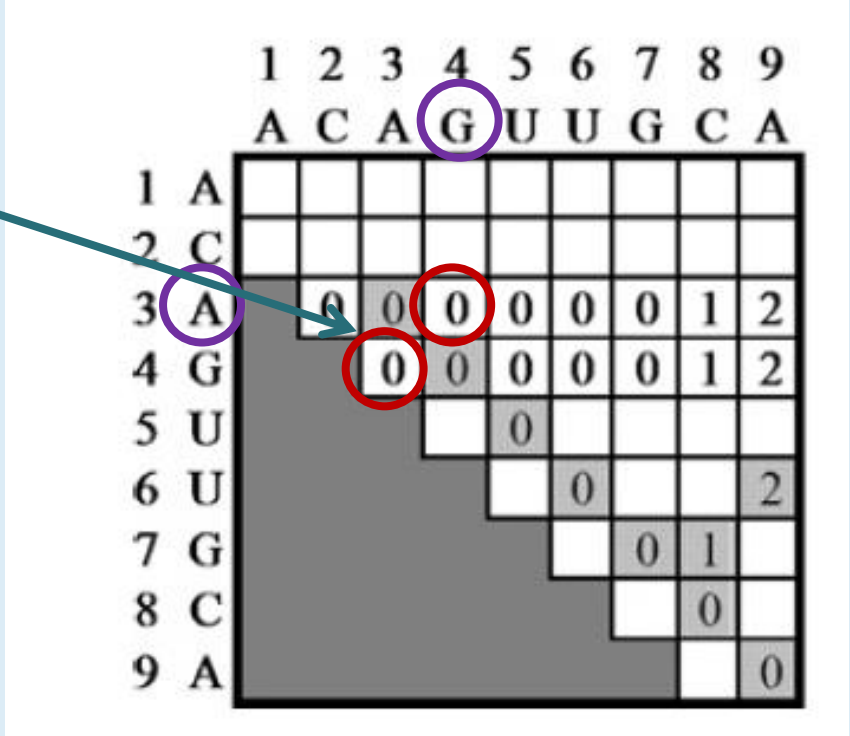
Explanation with Example

- **Suppose, i=3 and j=4;** $Q_{3,4}^{step-oct} = \emptyset$**,**
- **M[3,4] = max (L^{C}(3, 4) , M[3,4])**
- **L^{C}(3, 4) = M[4,3] + 0 = 0**

Algorithm 3: Forward RNA folding

input : An RNA string $S = s_1 s_2 \cdots s_n$
output: $L(1, n)$

for $i \leftarrow n$ *down to* 1 **do**
  set explicitly the values of $M[i, i-1]$ and $M[i, i]$ to the solutions for the corresponding trivial sub-instances;
  mark $S_{i,i}$ as an OCT;
  set $M[i, j] \leftarrow M[i, i] + M[i+1, j]$ for all $i < j \leqslant n$;
  for $j \leftarrow i+1$ *to* n **do**
    compute $L^c(i, j) = f(M[i+1, j-1], s_i, s_j)$;
    if $M[i, j] < L^c(i, j)$ **then**
      set $M[i, j] \leftarrow L^c(i, j)$;
      mark $S_{i,j}$ as an OCT;
    if $S_{i,j}$ *is a STEP* **then**
      for all j' *s.t.* $S_{j+1,j'}$ *is an OCT* **do**
        set $M[i, j'] \leftarrow \max\{M[i, j'], M[i, j] + M[j+1, j']\}$;
  discard from memory the values in all entries in row $i+1$ of M that do not correspond to OCT sub-instances;
return $M[1, n]$.



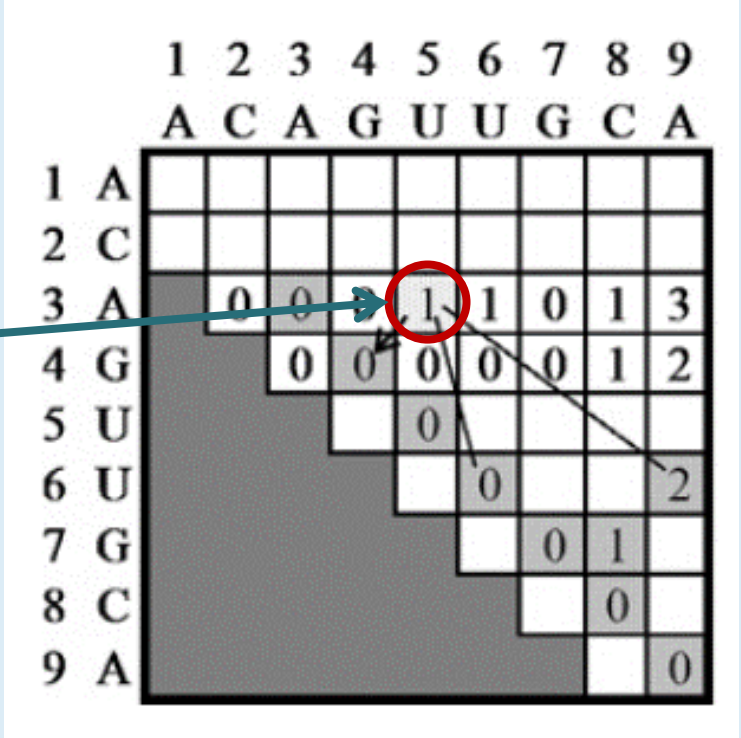
Explanation with Example

- **Suppose, i=3 and j=5;**
- **M[3,5] = L^P(3, 5) = 0; M[i, j] already contains optimal value of L^P(i, j)**
- **M[3,5] = L^C(3, 5) = M[4,4] + 1 = 1**
- **OCT candidate list: (9,9), ……., (3,3), (3,5)**

Algorithm 3: Forward RNA folding

input : An RNA string $S = s_1 s_2 \cdots s_n$
output: $L(1, n)$

```
for i ← n down to 1 do
    set explicitly the values of M[i, i − 1] and M[i, i] to the solutions for the corresponding trivial sub-instances;
    mark S_{i,i} as an OCT;
    set M[i, j] ← M[i, i] + M[i + 1, j] for all i < j ⩽ n;
    for j ← i + 1 to n do
        compute L^c(i, j) = f(M[i + 1, j − 1], s_i, s_j);
        if M[i, j] < L^c(i, j) then
            set M[i, j] ← L^c(i, j);
            mark S_{i,j} as an OCT;
        if S_{i,j} is a STEP then
            for all j' s.t. S_{j+1,j'} is an OCT do
                set M[i, j'] ← max{M[i, j'], M[i, j] + M[j + 1, j']};
    discard from memory the values in all entries in row i + 1 of M that do not correspond to OCT sub-instances;
return M[1, n].
```



Explanation with Example

- **Suppose, i=3 and j=5**
- **M[3,5] > M[3,3] + M[4,5]; so $S_{3,5}$ is a STEM**
- **Possible Set of $S_{j+1,j'}$ = ($S_{6,j'}$) = {(6,6), (6,7), (6,8), (6,9)} s.t. j'>j**

Algorithm 3: Forward RNA folding

input : An RNA string $S = s_1 s_2 \cdots s_n$
output: $L(1,n)$

```
for i ← n down to 1 do
    set explicitly the values of M[i, i−1] and M[i, i] to the solutions for the corresponding trivial sub-instances;
    mark S_{i,i} as an OCT;
    set M[i, j] ← M[i, i] + M[i+1, j] for all i < j ⩽ n;
    for j ← i+1 to n do
        compute L^c(i, j) = f(M[i+1, j−1], s_i, s_j);
        if M[i, j] < L^c(i, j) then
            set M[i, j] ← L^c(i, j);
            mark S_{i,j} as an OCT;
        if S_{i,j} is a STEM then
            for all j' s.t. S_{j+1,j'} is an OCT do
                set M[i, j'] ← max{M[i, j'], M[i, j] + M[j+1, j']};
    discard from memory the values in all entries in row i+1 of M that do not correspond to OCT sub-instances;
return M[1,n].
```

		1	2	3	4	5	6	7	8	9
		A	C	A	G	U	U	G	C	A
1	A									
2	C									
3	A		0	0	0	1	1	0	1	3
4	G			0	0	0	0	0	1	2
5	U					0				
6	U						0			2
7	G							0	1	
8	C								0	
9	A									0

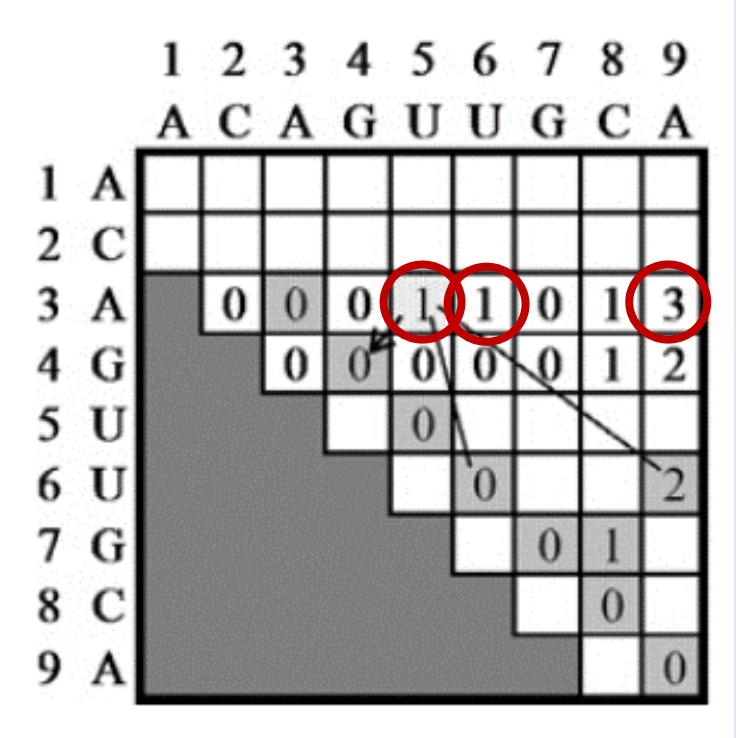
Explanation with Example

- **OCT candidate list: {(9,9), ….(6,6),…, (3,3), (7,8), (6,9), (3,5)}**
- **Possible Set of $S_{j+1,\, j'}$ = ($S_{6,\, j'}$) = {(6,6), (6,7), (6,8), (6,9)}**
- **OCT ∩ Possible Set of $S_{6,\, j'}$ = {(6,6),(6,9)}**
- **So, forward computation will update M[3,6] and M[3,9]**

Algorithm 3: Forward RNA folding

input : An RNA string $S = s_1 s_2 \cdots s_n$
output: $L(1, n)$

```
for i ← n down to 1 do
    set explicitly the values of M[i, i − 1] and M[i, i] to the solutions for the corresponding trivial sub-instances;
    mark S_{i,i} as an OCT;
    set M[i, j] ← M[i, i] + M[i + 1, j] for all i < j ⩽ n;
    for j ← i + 1 to n do
        compute L^c(i, j) = f(M[i + 1, j − 1], s_i, s_j);
        if M[i, j] < L^c(i, j) then
            set M[i, j] ← L^c(i, j);
            mark S_{i,j} as an OCT;
        if S_{i,j} is a STEP then
            for all j' s.t. S_{j+1,j'} is an OCT do
                set M[i, j'] ← max{M[i, j'], M[i, j] + M[j + 1, j']};
    discard from memory the values in all entries in row i + 1 of M that do not correspond to OCT sub-instances;
return M[1, n].
```



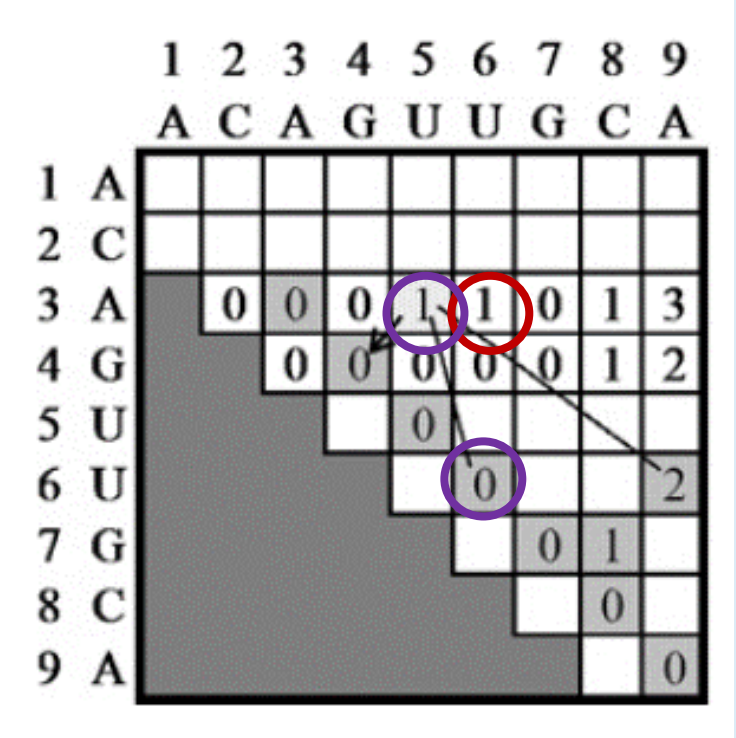
Explanation with Example

- **M[3,6] = max{M[3,6], M[3,5] + M[6,6]}**
- **M[3,9] = max{M[3,9], M[3,5] + M[6,9]}**

Algorithm 3: Forward RNA folding

input : An RNA string $S = s_1 s_2 \cdots s_n$
output: $L(1, n)$

```
for i ← n down to 1 do
    set explicitly the values of M[i, i − 1] and M[i, i] to the solutions for the corresponding trivial sub-instances;
    mark S_{i,i} as an OCT;
    set M[i, j] ← M[i, i] + M[i + 1, j] for all i < j ⩽ n;
    for j ← i + 1 to n do
        compute L^c(i, j) = f(M[i + 1, j − 1], s_i, s_j);
        if M[i, j] < L^c(i, j) then
            set M[i, j] ← L^c(i, j);
            mark S_{i,j} as an OCT;
        if S_{i,j} is a STEP then
            for all j′ s.t. S_{j+1,j′} is an OCT do
                set M[i, j′] ← max{M[i, j′], M[i, j] + M[j + 1, j′]};
    discard from memory the values in all entries in row i + 1 of M that do not correspond to OCT sub-instances;
return M[1, n].
```



THANK YOU