Stochastic Context Free Grammars

Gaurav Sharma

University of Rochester

Motivation I

- Consider the following words/phrases/setences:
  - kayak
  - madam
  - maam
  - race car
  - was it a rat i saw?
  - dammit, i'm mad!
  - never odd or even
- What pattern do you see in these words?
  - All are palindromes
    - Read the same backwards as forwards
    - Modulo some license with punctuation and spaces

Motivation II

- Modeling palindromic dependence
  - Is a Markovian model a good way to model palindromic dependence?
    - Alternatively, HMM incorporating stochastic variations
- Recall Markovian dependence
  - Future independent of past given present
    - Problematic for long range dependencies inherent in palindromes
    - Enlarging context does not really help
- Dependency structure is between the ends
- Questions:
  - What are meaningful generalizations of the palindromic dependence?
    - Deterministic/stochastic
  - Do these have utility beyond mere curiosities?

Motivation III

- Markov models and HMMs provide a model for sequential dependence
- Dependence is often not sequential, example grammar for a language
  - Consider a programming language, what dependence is not purely sequential?
  - Example: Opening brace needs a closing brace
  - Same is true of natural languages
    - "... We find that no finite-state Markov process that produces symbols with transition from state to state can serve as an English grammar..." [3]

Motivation IV

- Generalizing models for dependence?
  - Motivations from natural language processing
    - Grammars of language studied extensively for text parsing, understanding
    - Symbols and production rules
    - Chomsky hierarchy of grammars [1]: regular, context free, context sensitive, unrestricted
    - Complexity vs generality tradeoff
    - Stochastic grammars associate probabilities with production rules
    - Allow for probabilistic inference: e.g. most likely derivation, parameter estimation, ...

Motivating Example: RNA Secondary Structure

- RNA Structure and Hierarchy
  - Most likely secondary structure $\approx$ maximum base pairs

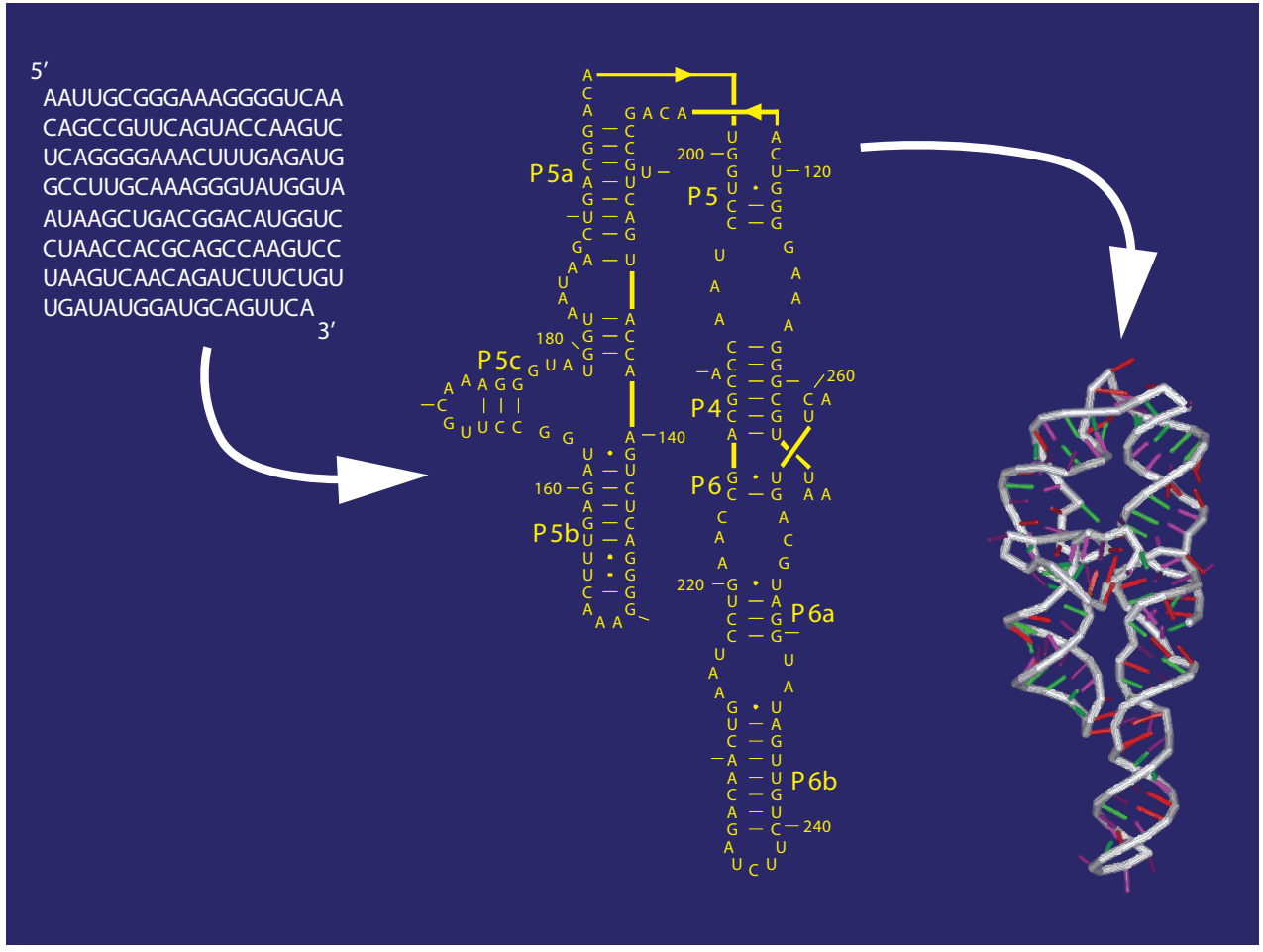


Figure: Hierarchy of RNA structure formation [10, 6, 7]

RNA Secondary Structure

- Secondary structure of an RNA molecule is defined as the set of base pairs in the RNA molecule
- Formation of hydrogen bonds between nucleotides
  - Canonical base pairs (complementary nucleotides (nts))
    - A can pair with U
    - G can pair with C and U
    - G-U pair called non Watson-Crick pair
- Canonical base pair set

$$\mathcal{C} = \{(A, U), (G, C), (G, U), (U, A), (C, G), (U, G)\}$$

- Watson-Crick base pair set (will often use for simplicity)

$$\mathcal{C} = \{(A, U), (G, C), (U, A), (C, G)\}$$

RNA Secondary Structures: Formal Definition

- Given an RNA sequence $\mathrm{x} = {}_1^N\mathrm{x}$, $x_n \in \mathcal{A} = \{A, U, G, C\}$
- Set of possible base pairs

$$\mathcal{B} = \{(i,j) | 1 \leq i < j \leq N, (x_i, x_j) \in \mathcal{C}\}$$

- Secondary structure: An RNA (secondary) structure on the sequence x is a subset of nonconflicting base pairs from $\mathcal{C}$
  - Base pairs are nonconflicting if any nucleotide participates in at most one pair
  - A secondary structure $\mathcal{S}$, is a selection of a certain number M of nonconflicting base pairs from $\mathcal{B}$, i.e., for a positive integer M,

$$\mathcal{S} = \{(i_1, j_1), (i_2, j_2), \ldots, (i_M, j_M) | (i_l, j_l) \in \mathcal{B}, \forall 1 \leq l \leq M, \text{nonconflicting}\}$$

is a secondary structure.

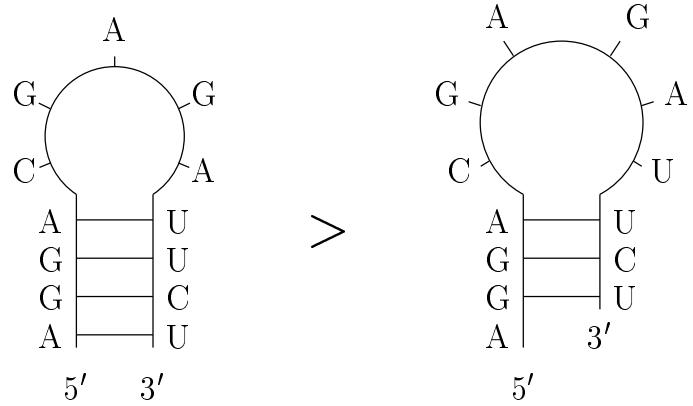
- $\tilde{\mathcal{S}}(\mathrm{x})$ = set of all (valid) secondary structures on the sequence x

RNA Structure Prediction: Maximizing Base Pairs

- "Optimal" secondary structure $\equiv$ maximizes the number of base pairs between complementary nucleotides

$$\tilde{\mathcal{S}}^*(\mathrm{x}) = \arg \max_{\mathcal{S} \in \tilde{\mathfrak{S}}(\mathrm{x})} |\mathcal{S}|$$

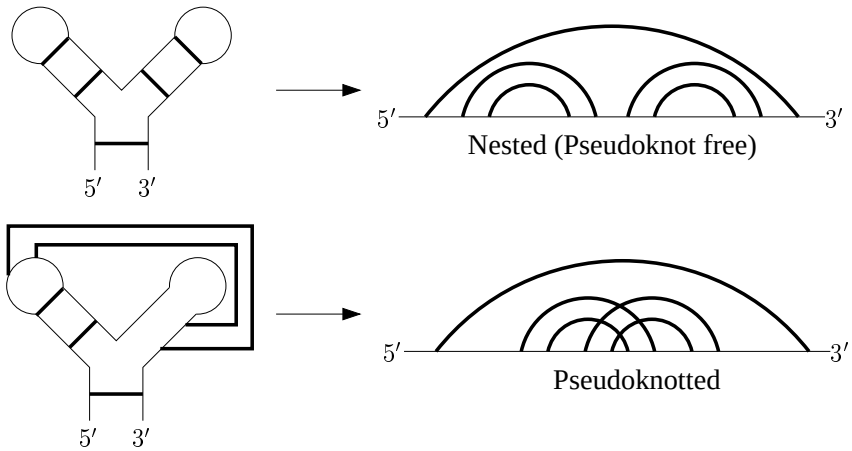
- $|\mathcal{D}|$ denotes the cardinality of the set $\mathcal{D}$
- Example: $\mathrm{x} = AGGACGGAUUCU$, $N = 12$. Structures



$$\mathcal{S}_1 = \{(1,12),(2,11),(3,10),(4,9)\}$$
$$\mathcal{S}_2 = \{(2,12),(3,11),(4,10)\}$$
$$|\mathcal{S}_1| > |\mathcal{S}_2|$$

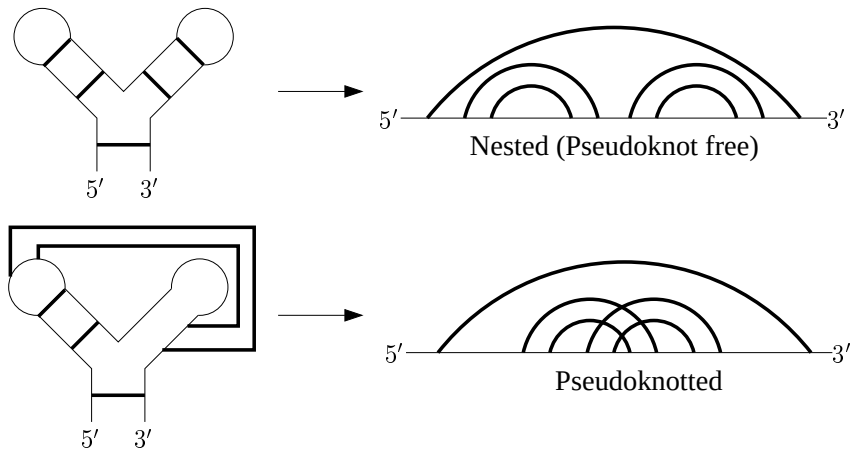
RNA Structure Prediction: Structure Search Space for Maximization

- Search space of all possible secondary structures grows exponentially with length N for the sequence
  - Shortest RNAs of interest are $N \approx 72$ nts
  - Brute force search over secondary structures is infeasible
- There is no known efficient algorithm that can handle the complete search space $\tilde{\mathcal{S}}(x)$
- Approach: restrict search space such that dynamic programming solution becomes possible
- Search over pseudoknot-free structures ($\equiv$ nesting structures)
  - Pseudoknot illustration



RNA Structure Prediction: Pseudoknot-free ($\equiv$ nesting) structures

- Formal Definition:
  - Crossing base pair in a secondary structure $\mathcal{S}$: A base pair $(i, j) \in \mathcal{S}$ is said to be a crossing base pair, if there exists (another) base pair $(i', j') \in \mathcal{S}$ such that either: $i' < i < j' < j$ or $i < i' < j < j'$.
  - A secondary structure $\mathcal{S}$ is said to be pseudoknot-free, or equivalently nested, if it has no crossing base pairs.
- Intuition: allows computation of base pair maximizing structure to be recursively separated into (noninteracting) inside and outside segments



RNA Structure Prediction for Pseudoknot-free ($\equiv$ nesting) structures

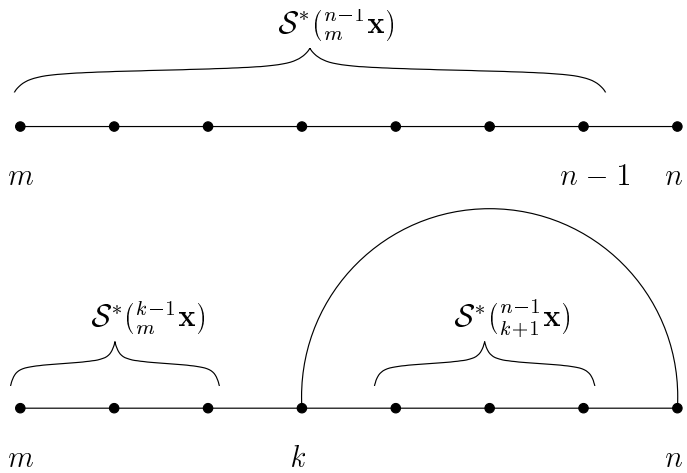
- Search Space: Set $\mathcal{S}(x)$ of pseudoknot-free (nesting) structures
- "Optimal" base pair maximizing pseudoknot-free (nesting) structure

$$\mathcal{S}^*(x) = \arg \max_{\mathcal{S} \in \mathcal{S}(x)} |\mathcal{S}|$$

- For equiprobable iid nts, it is estimated [13] that $|\mathcal{S}(x)| \approx 1.86^N$
- Brute force evaluation of optimal structure is still unviable
- However, a dynamic programming approach exists with time complexity N^3 and memory complexity N^2
- Will develop next
  - Nussinov Recursion
  - Analogous to string edit problem

RNA Structure Prediction: Recursive Decomposition of Optimal Structures

- Recursive determination of optimal nested (pseudoknot-free) structure
  - Consider optimal nesting structure $\mathcal{S}^*({}_m^n\mathrm{x})$ for the substring ${}_m^n\mathrm{x}$
    - Assume optimal nesting structures already computed for all substrings of ${}_m^{n-1}\mathrm{x}$
  - In optimal nesting structure for the substring ${}_m^n\mathrm{x}$ the nt at index n is either unpaired or paired with some nt at index k, $m \leq k \leq (n-1)$
    - Nesting condition implies $\mathcal{S}^*({}_m^n\mathrm{x})$ is made up from optimal structures for substrings of ${}_m^{n-1}\mathrm{x}$ in one of the two ways shown in the figure

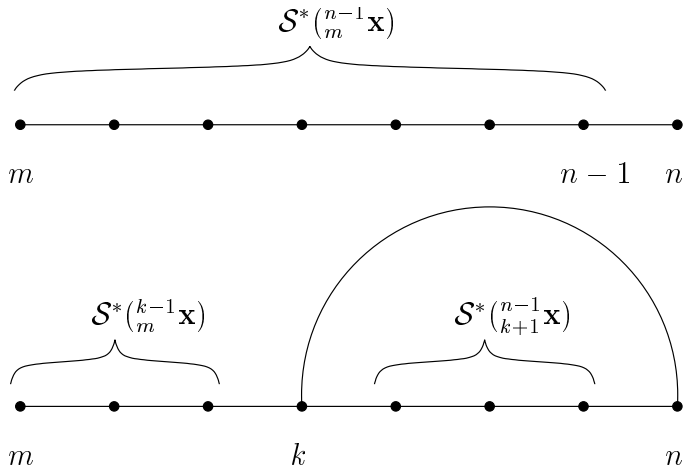


Dynamic Programming Recursions (Nussinov)

- Let $d(m, n) =$ maximum number of base pairs in nested structure for the substring ${}_m^n\mathrm{x}$
- Optimal structure recursion (figure) $\implies$ recursion for $d(m, n)$

$$d(m,n) = \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k: m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, k-1) + d(k+1, n-1) + 1 \end{cases}$$

- Observe: resulting inside-outside decomposition
- Note: Need to address special $k = m$ case with empty left string



Nussinov Dynamic Programming Algorithm

- Nussinov dynamic programming array
  - $d(m, n)$ = maximum number of base pairs in pseudo-knot free structure for the substring ${}^{n}_{m}\mathrm{x}$
  - Analogous to string edit problem, additional loop for multibranch loops
- Initialization: For $l = 1$ length sequences, no base pairs
  - For $m = 1$ to N: $n \leftarrow m + (l - 1) = m$, $d(m, n) \leftarrow 0$
- Recursively enlarge length l of the sequence on one (n) side

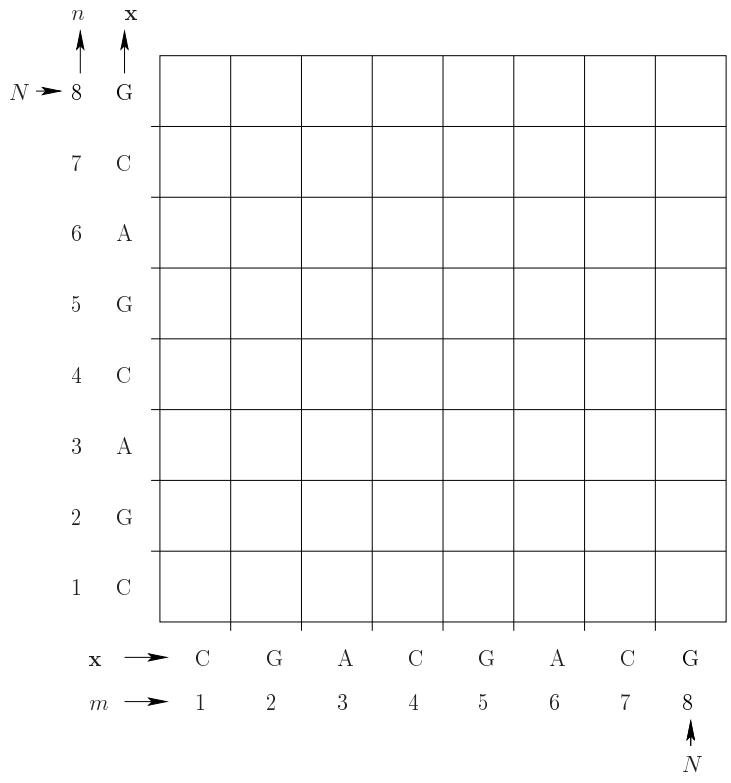
For $l = 2$ to N, for $m = 1$ to $(N + 1 - l)$, $n \leftarrow m + (l - 1)$

$$d(m, n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$

- **Note:** $\max(k-1, m)$ **addresses** $k = m$ **case** $\equiv$ **initializing** $d(m, m-1) = 0$ **with** $\max(k-1, m) \rightarrow k-1$

Nussinov Dynamic Programming Example

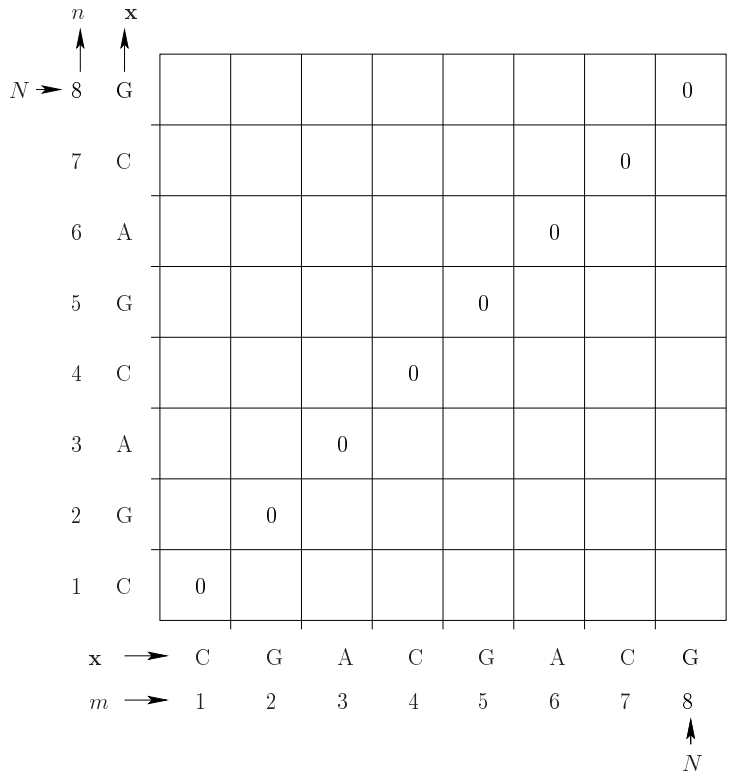
- Sequence x = *CGACGACG*, length $N = 8$
- Blank dynamic programming array
  - Want: $d(m, n)$ = max. no. of base pairs for substring ${}^{n}_{m}\mathrm{x}$



Nussinov Dynamic Programming: Step 0 (Initialization)

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}^{n}_{m}\mathrm{x}$
  - Initialization: For sequences of length $l = 1$

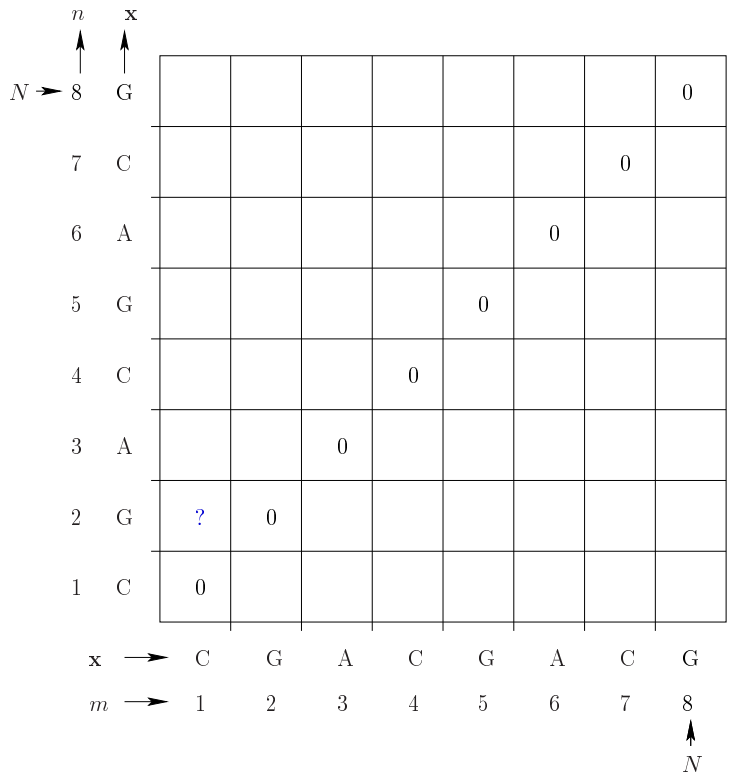
For $m = 0$ to N, $d(m, m) \leftarrow 0$



Nussinov Dynamic Programming: Step 1a

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}^{n}_{m}\mathrm{x}$
  - Recursion: For $l = 2$, for $m = 1$, $n \leftarrow m + (l - 1) = m + 1 = 2$

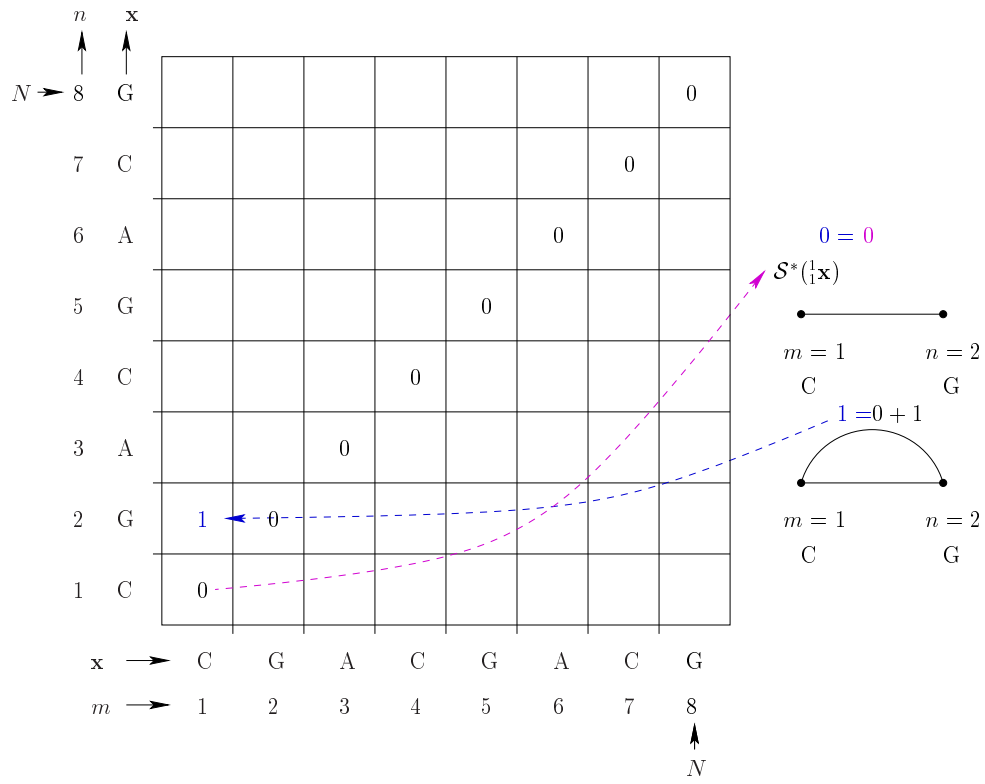
$$d(m,n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$



Nussinov Dynamic Programming: Step 1b

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}_m^n\mathrm{x}$
  - Recursion: For $l = 2$, for $m = 1$, $n \leftarrow m + (l - 1) = m + 1 = 2$

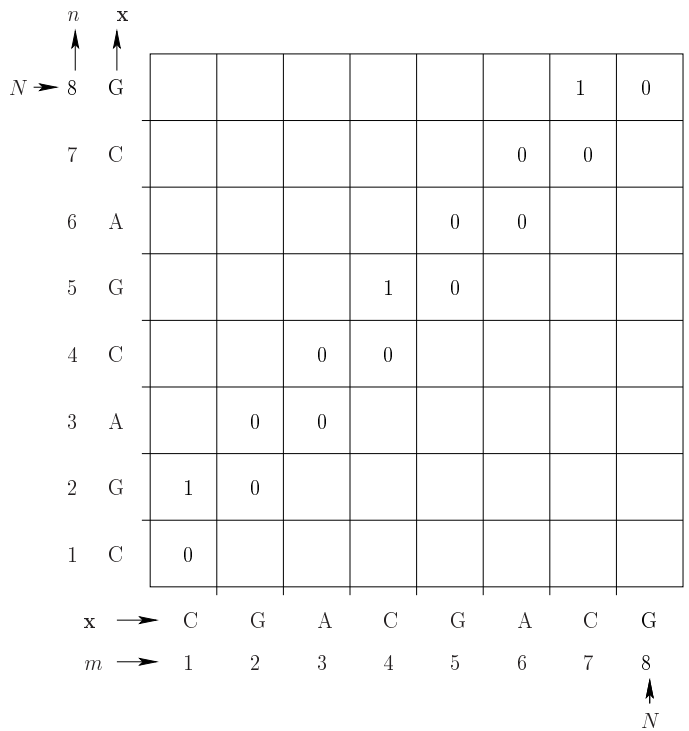
$$d(m,n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$



Nussinov Dynamic Programming: Step 1

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}^{n}_{m}\mathrm{x}$
  - Recursion: For $l = 2$, for $m = 1$ to $(N + 1 - l) = 7$, $n \leftarrow m + (1 - 1) = m + 1$

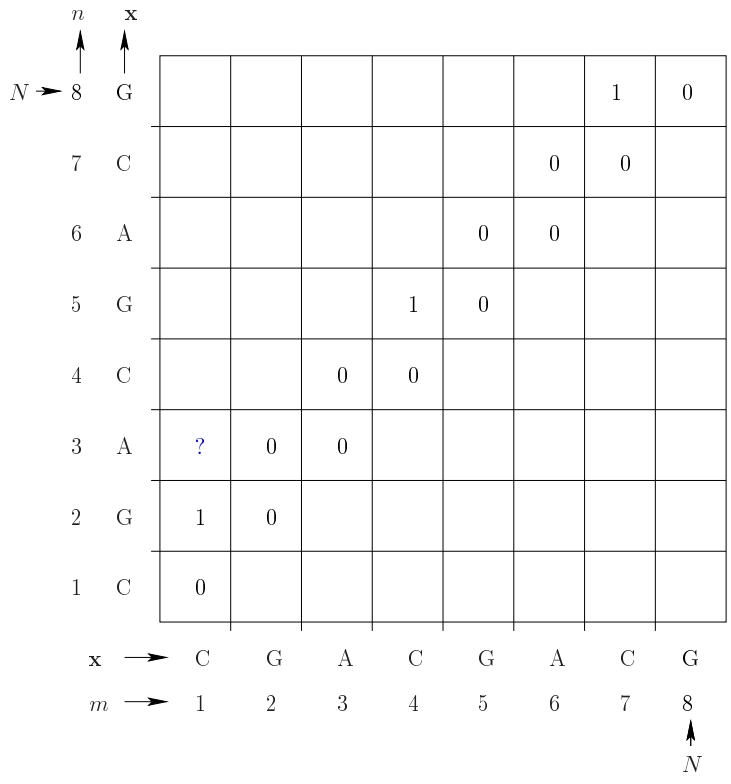
$$d(m,n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$



Nussinov Dynamic Programming: Step 2a

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}^{n}_{m}\mathrm{x}$
  - Recursion: For $l = 3$, for $m = 1$, $n \leftarrow m + (1 - 1) = m + 2 = 3$

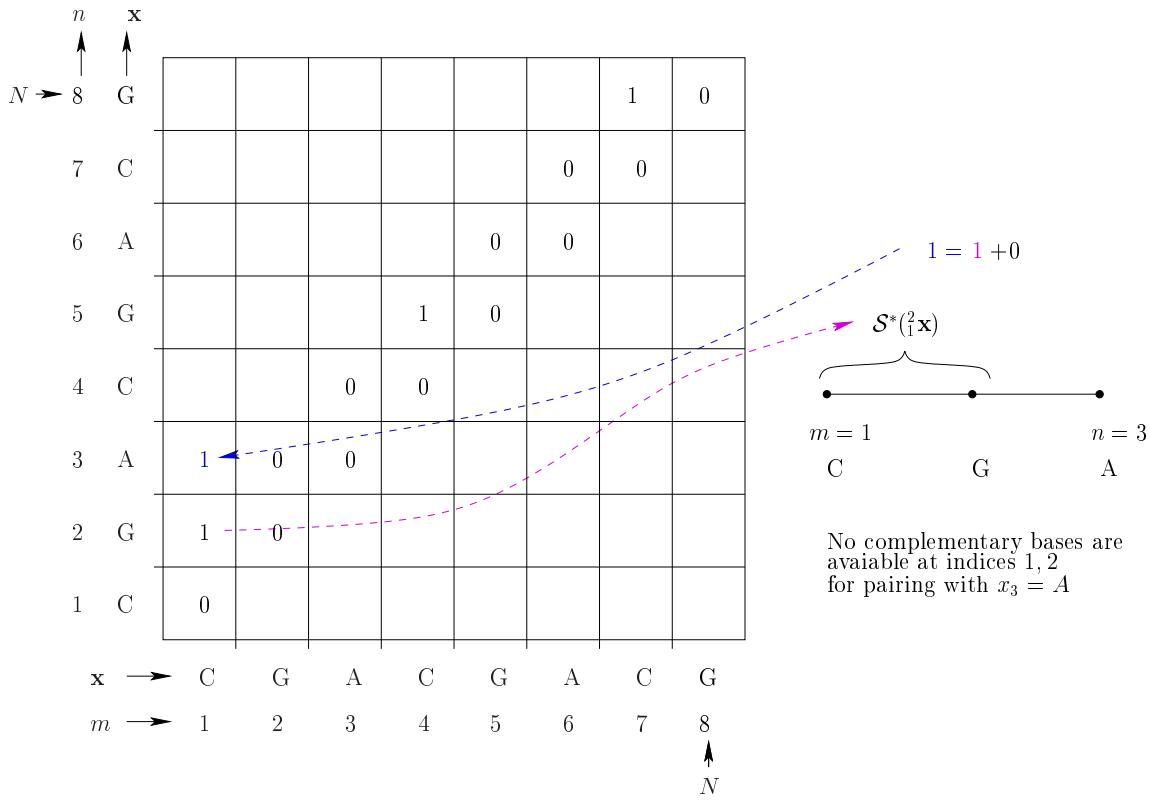
$$d(m,n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$



Nussinov Dynamic Programming: Step 2b

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}^{n}_{m}\mathrm{x}$
  - Recursion: For $l = 3$, for $m = 1$, $n \leftarrow m + (1 - 1) = m + 2 = 3$

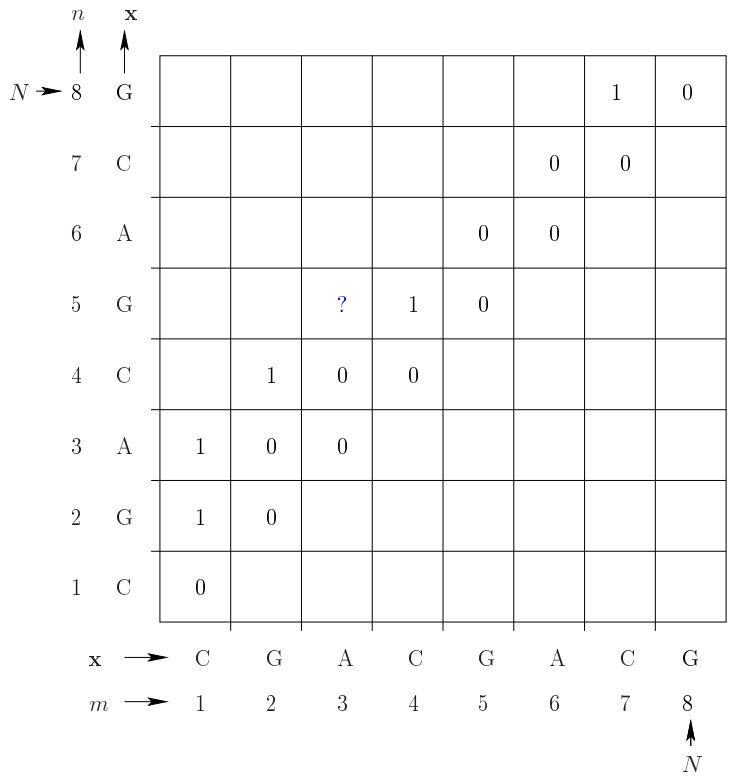
$$d(m,n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$



Nussinov Dynamic Programming: Step 2c

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}^{n}_{m}\mathrm{x}$
  - Recursion: For $l = 3$, for $m = 3$, $n \leftarrow m + (1 - 1) = m + 2 = 5$

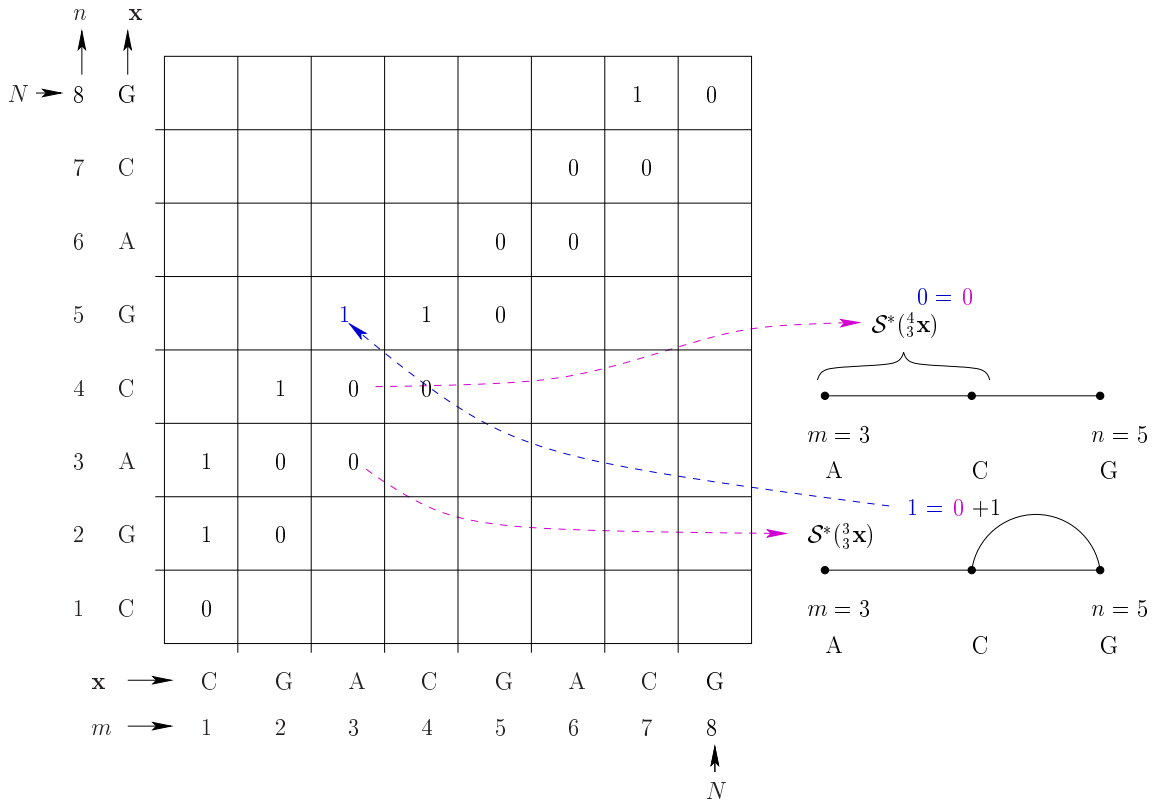
$$d(m,n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$



Nussinov Dynamic Programming: Step 2d

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}^{n}_{m}\mathrm{x}$
  - Recursion: For $l = 3$, for $m = 3$, $n \leftarrow m + (1 - 1) = m + 2 = 5$

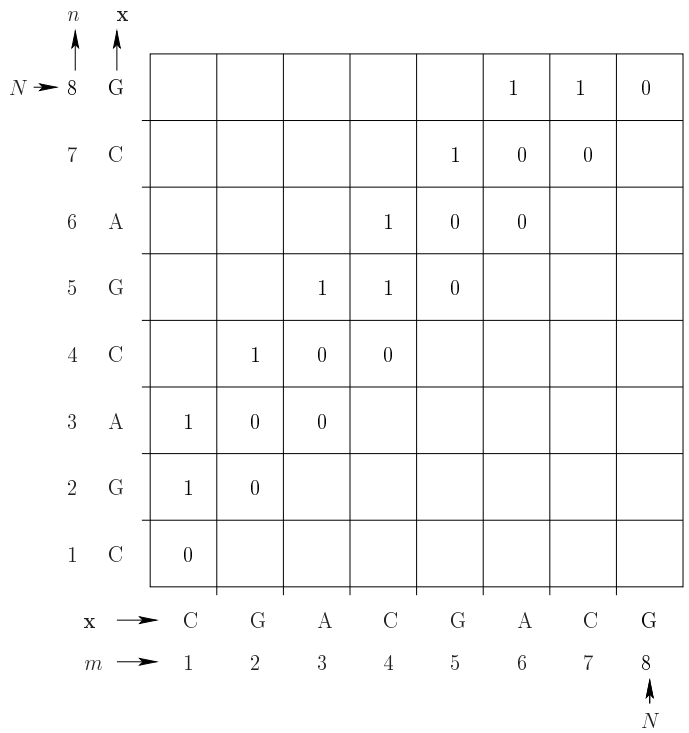
$$d(m,n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$



Nussinov Dynamic Programming: Step 2

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}^{n}_{m}\mathrm{x}$
  - Recursion: For $l = 3$, for $m = 1$ to $(N + 1 - l) = 5$, $n \leftarrow m + (1 - 1) = m + 2$

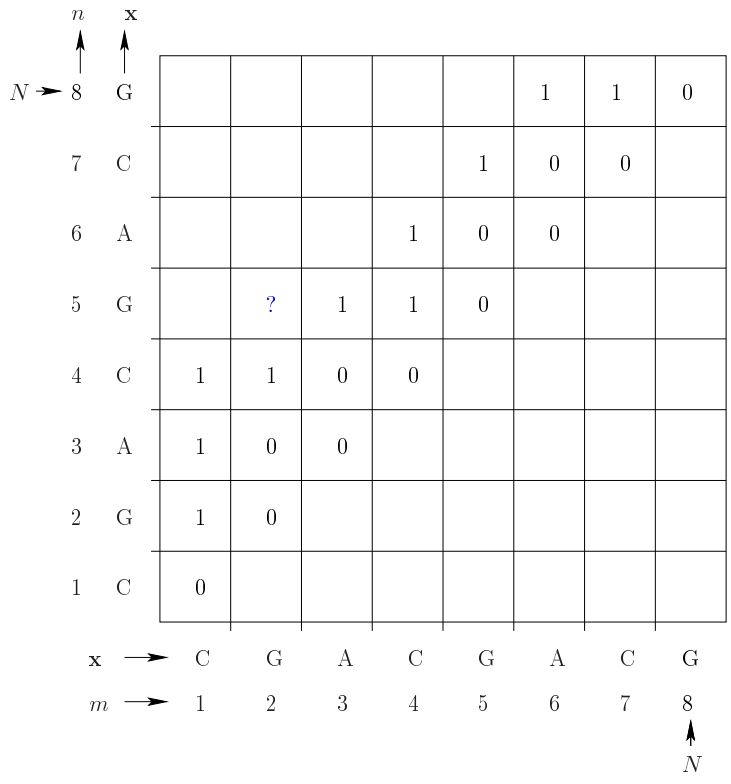
$$d(m,n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$



Nussinov Dynamic Programming: Step 3a

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}^{n}_{m}\mathrm{x}$
  - Recursion: For $l = 4$, for $m = 2$, $n \leftarrow m + (1 - 1) = m + 3 = 5$

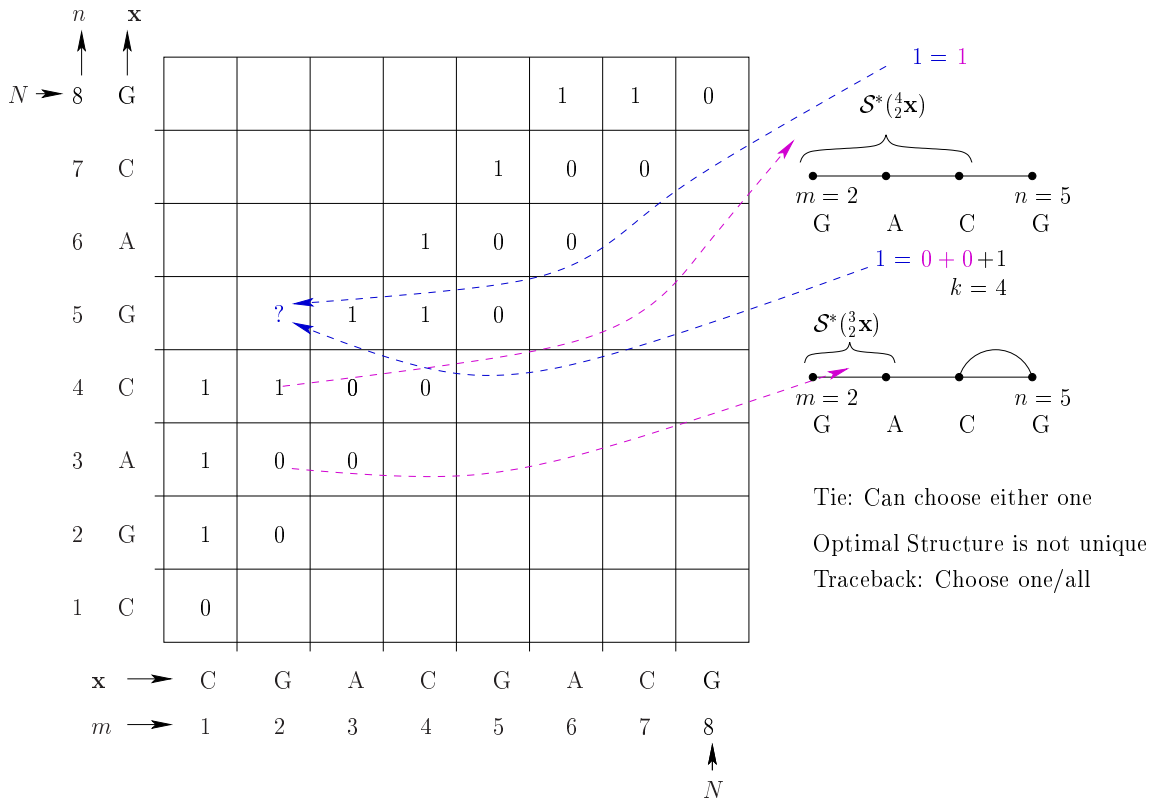
$$d(m, n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$



Nussinov Dynamic Programming: Step 3b

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}^{n}_{m}\mathrm{x}$
  - Recursion: For $l = 4$, for $m = 2$, $n \leftarrow m + (1 - 1) = m + 3 = 5$

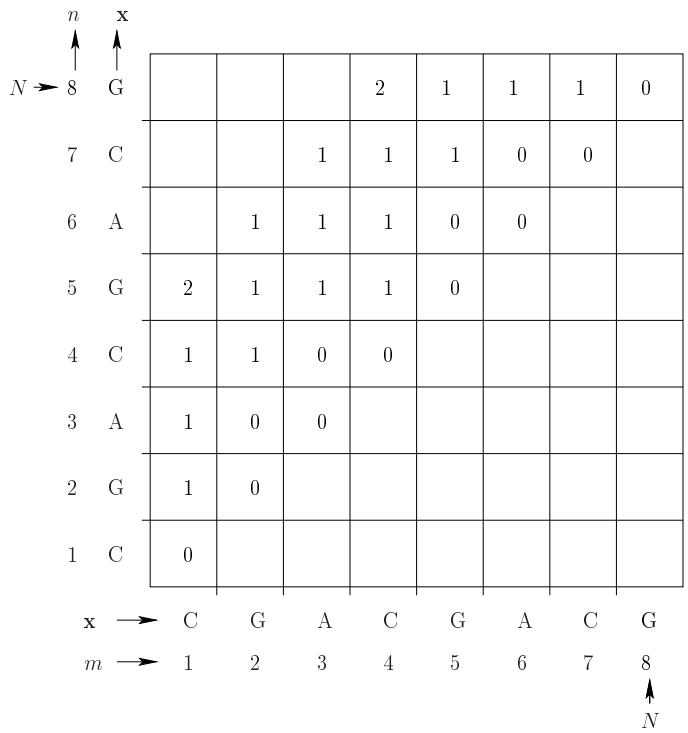
$$d(m,n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$



Nussinov Dynamic Programming: Step 3

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}^{n}_{m}\mathrm{x}$
  - Recursion: For $l = 4$, for $m = 1$ to $(N + 1 - l) = 5$, $n \leftarrow m + (1 - 1) = m + 3$

$$d(m,n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$



Nussinov Dynamic Programming: Step 4a

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}^{n}_{m}\mathrm{x}$
  - Recursion: For $l = 8$, for $m = 1$, $n \leftarrow m + (1 - 1) = m + 7 = 8$

$$d(m,n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$

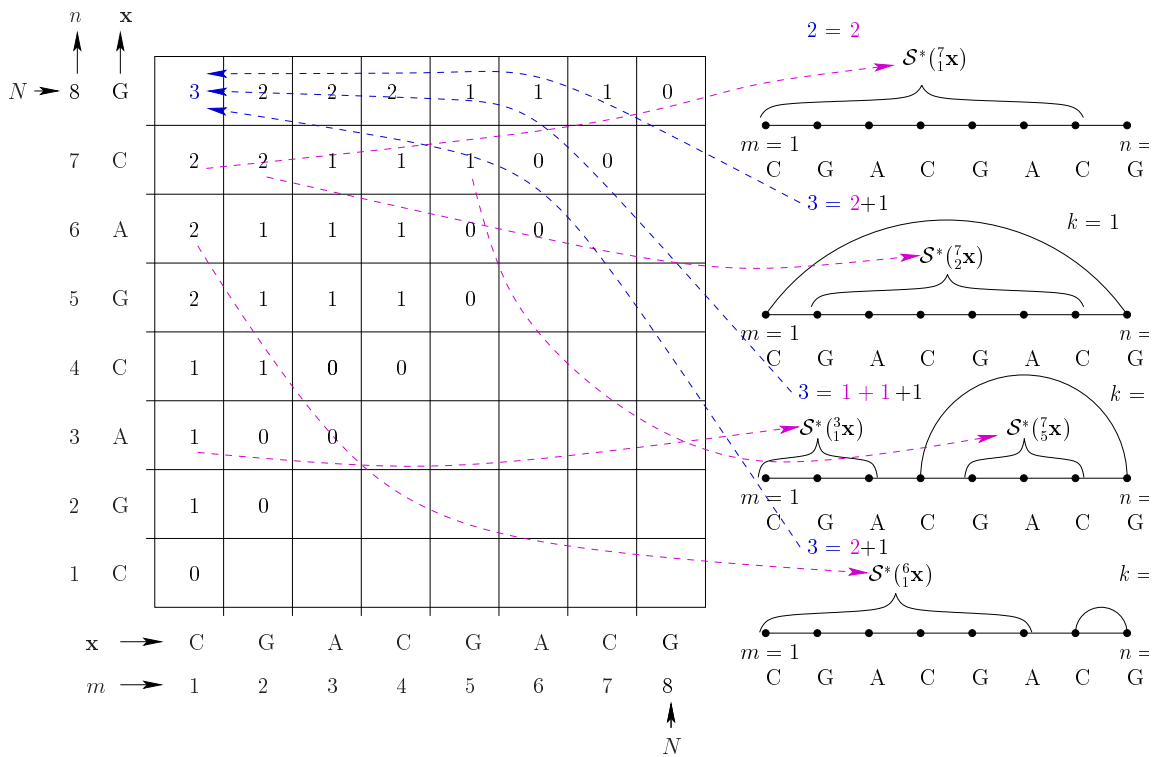
n	x								
$N \rightarrow 8$	G	?	2	2	2	1	1	1	0
7	C	2	2	1	1	1	0	0	
6	A	2	1	1	1	0	0		
5	G	2	1	1	1	0			
4	C	1	1	0	0				
3	A	1	0	0					
2	G	1	0						
1	C	0							
x $\rightarrow$		C	G	A	C	G	A	C	G
$m \rightarrow$		1	2	3	4	5	6	7	8

N

Nussinov Dynamic Programming: Step 4b

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}_m^n\mathrm{x}$
  - Recursion: For $l = 8$, for $m = 1$, $n \leftarrow m + (1 - 1) = m + 7 = 8$

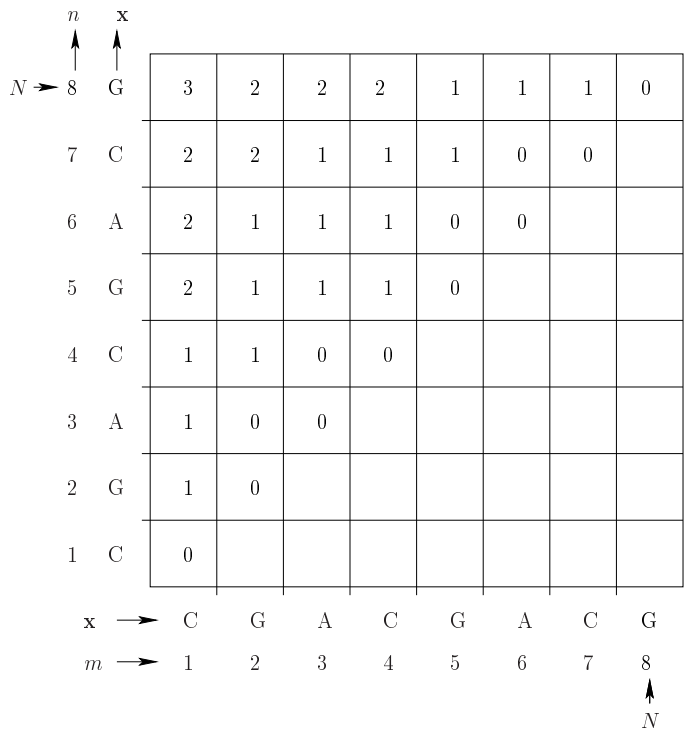
$$d(m,n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$



Nussinov Dynamic Programming: Step 4

- Dynamic programming array $d(m, n) =$ max. no. of base pairs for substring ${}^{n}_{m}\mathrm{x}$
  - Recursion: For $l = 8$, for $m = 1$ to $(N + 1 - l) = 1$, $n \leftarrow m + (1 - 1) = m + 7 = 8$

$$d(m,n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \le k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$



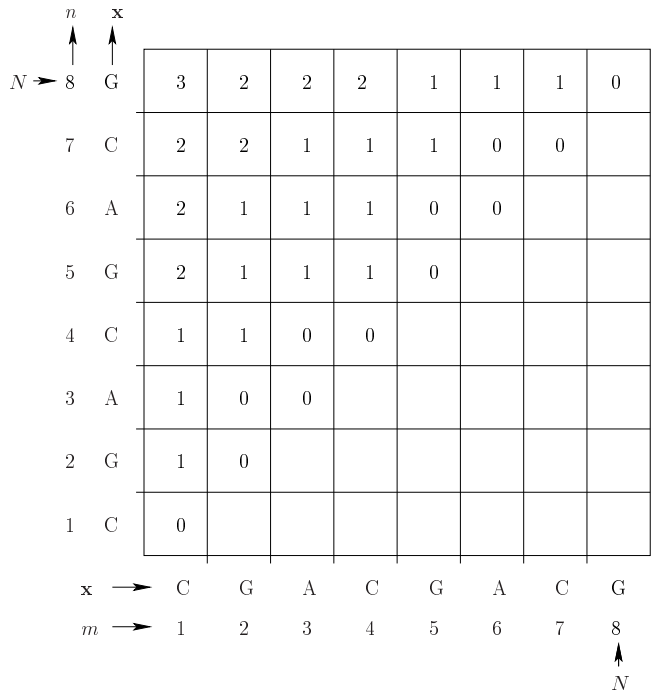
Nussinov Dynamic Programming Algorithm: Observations on Array Filling

- Principal idea behind dynamic programming
  - Need to consider only optimal substructures when creating larger optimal structures from smaller ones
    - Non-optimal substructures would not contribute to an optimal nesting structure
    - as they could be improved upon retaining other base pairs
- Order of filling in the arrays uses above principal idea
  - Array is filled for increasing length l, corresponding feasible locations $(m, m + (l - 1))$
    - Constraint: Locations corresponding to shorter lengths need to be filled before those of longer lengths (re-used)
    - For a fixed l, the different feasible locations of the type $(m, m + (l - 1))$ can be filled in any order
    - Algorithm description assumed a specific order (increasing m) but other orders can be used
    - Specifically, parallelizable for faster implementation

Nussinov Dynamic Programming: Complete Array

-Complete dynamic programming array $d(m, n)$ after completion of recursions

- Optimal structure has $d(1, N) = 3$ base pairs
- Need traceback for generating the optimal structure (next)



- Upon completion of algorithm $d(1, N)$ is the maximum number of base pairs over all nested structures on the sequence x
  - Traceback to obtain a structure that maximizes the number of base pairs

Nussinov Dynamic Programming Fill-Step Complexity

- Recall algorithm: progressively expands sequence length l

For $l = 2$ to N, for $m = 1$ to $(N + 1 - l)$, $n \leftarrow m + (l - 1)$

$$d(m,n) \leftarrow \max \begin{cases} d(m, n-1) \\ \max\limits_{\substack{k:m \leq k < n, \\ (x_k, x_n) \in \mathcal{C}}} d(m, \max(k-1, m)) + d(k+1, n-1) + 1 \end{cases}$$

- $O(l)$ computations in inner most max (multibranching test)
- Upto a scale factor the number of operations is

$$\sum_{l=1}^{N} (N - l) \times l = N \sum_{l=1}^{N} l - \sum_{l=1}^{N} l^2 \quad = \frac{N^2(N+1)}{2} - \frac{N(N+1)(2N+1)}{6}$$

$$= \frac{N^3}{6} + \text{lower order terms}$$

- $\implies$ Time complexity of fill step is $O(N^3)$

Nussinov Dynamic Programming Algorithm: Traceback

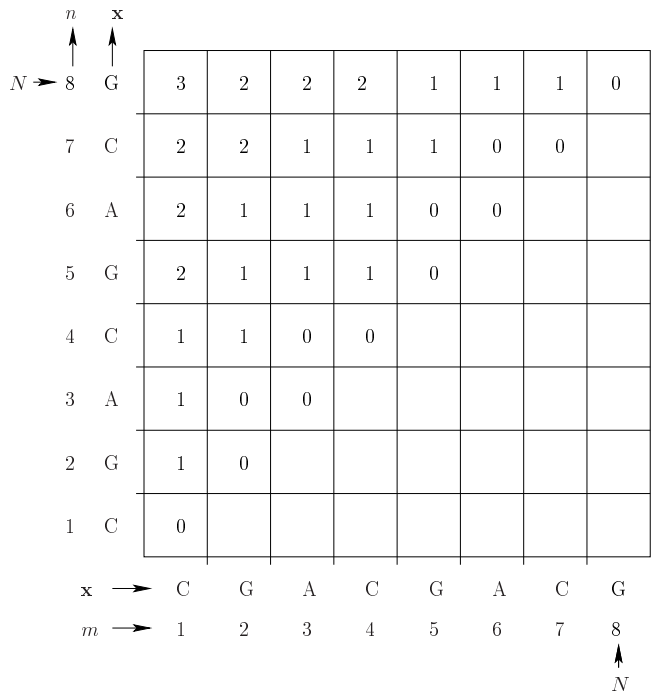
- Traceback: Recursive algorithm to output an optimal structure

Input: $\mathrm{x}, d(\cdot,\cdot)$ // Sequence, Filled Dynamic Programming Array
Output: $\mathcal{S}^*(\mathrm{x})$ // An optimal structure for x
$\mathcal{S}^*(\mathrm{x}) \leftarrow \mathtt{TB}(1, N, d, \mathrm{x})$ // traceback from $(1, N)$
$\mathtt{TB}(m, n, d, \mathrm{x})$ // Traceback function **if** $n \leq m$ **then**
  $\mathcal{S} \leftarrow \phi$ // empty substructure
else if $d(m, n) = d(m, n-1)$ **then**
  $\mathcal{S} \leftarrow \mathtt{TB}(m, n-1, d, \mathrm{x})$ // nt. n unpaired
else // nt n paired with k for some $m \leq k \leq (n-1)$
  for $k \leftarrow m$ **to** $(n-1)$ **do** // find k and recurse
    if $(x_k, x_n) \in \mathcal{C}$ ***and***
     $d(m, n) = d(m, \max(k-1, m)) + d(k+1, n-1) + 1$ **then**
      $\mathcal{S} \leftarrow \mathtt{TB}(m, (k-1), d, \mathrm{x}) \bigcup \{(k, n)\} \bigcup \mathtt{TB}(k+1, (n-1), d, \mathrm{x})$
    end
  end
end
return $\mathcal{S}$

Algorithm 1: Nussinov Traceback Algorithm

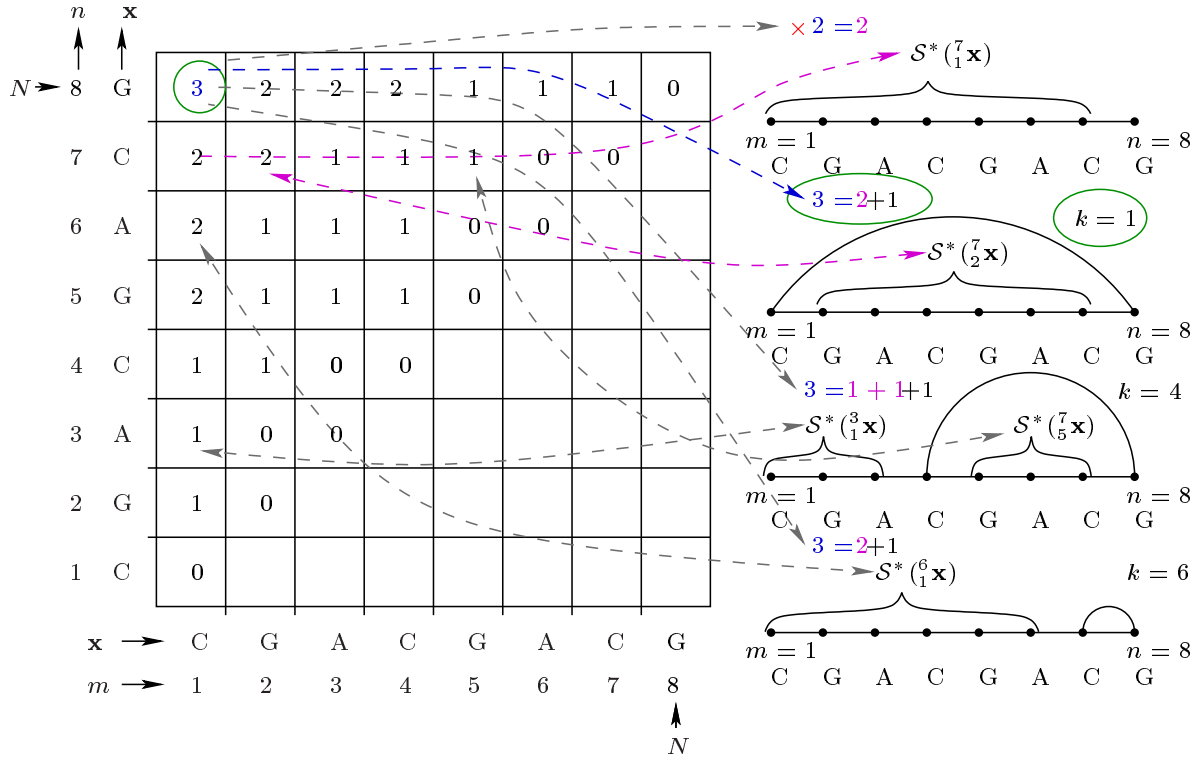
Nussinov Traceback Example

- Recall: $N = 8$, complete dynamic programming array $d(m, n)$ after completion of recursions



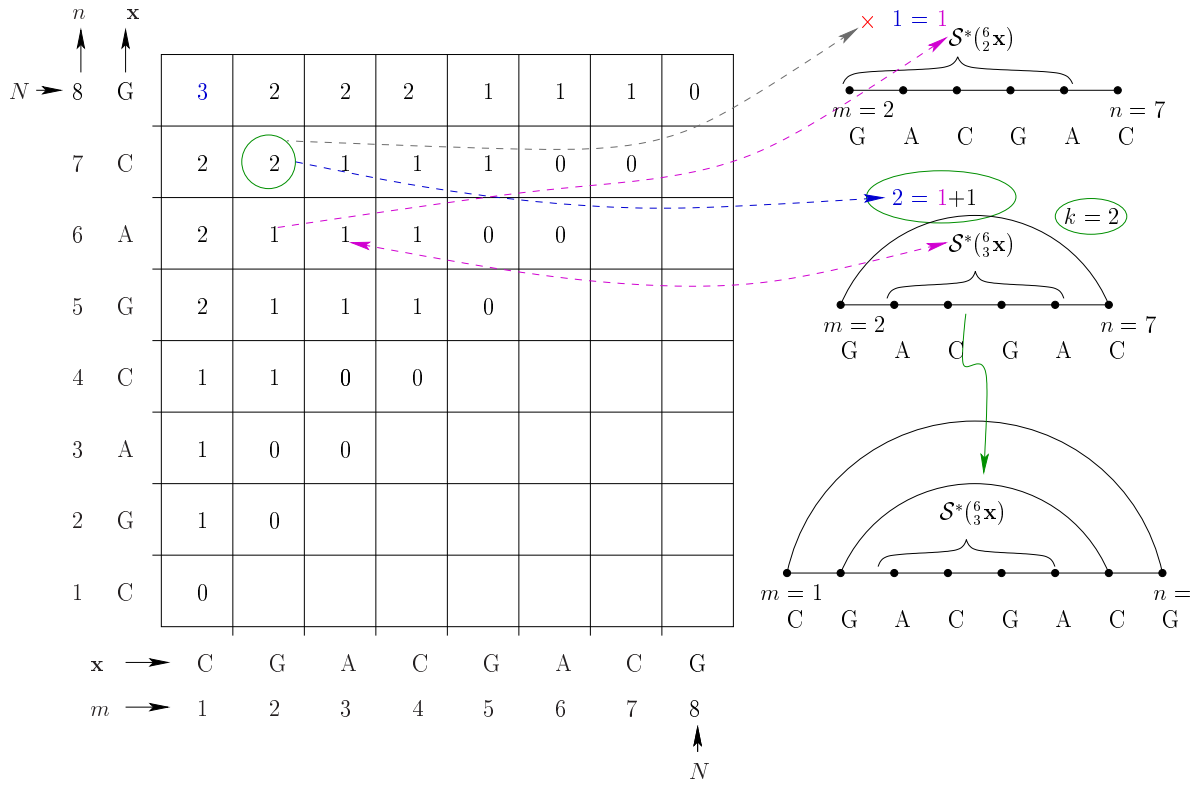
- Upon completion of algorithm $d(1, N)$ is the maximum number of base pairs over all nested structures on the sequence x
  - Optimal structure has $d(1, N) = d(1, 8) = 3$ base pairs
  - Traceback: $\mathcal{S}^*(\mathrm{x}) \leftarrow \mathrm{TraceBack}(1, N, d, \mathrm{x})$

Nussinov Traceback Example: Step 1



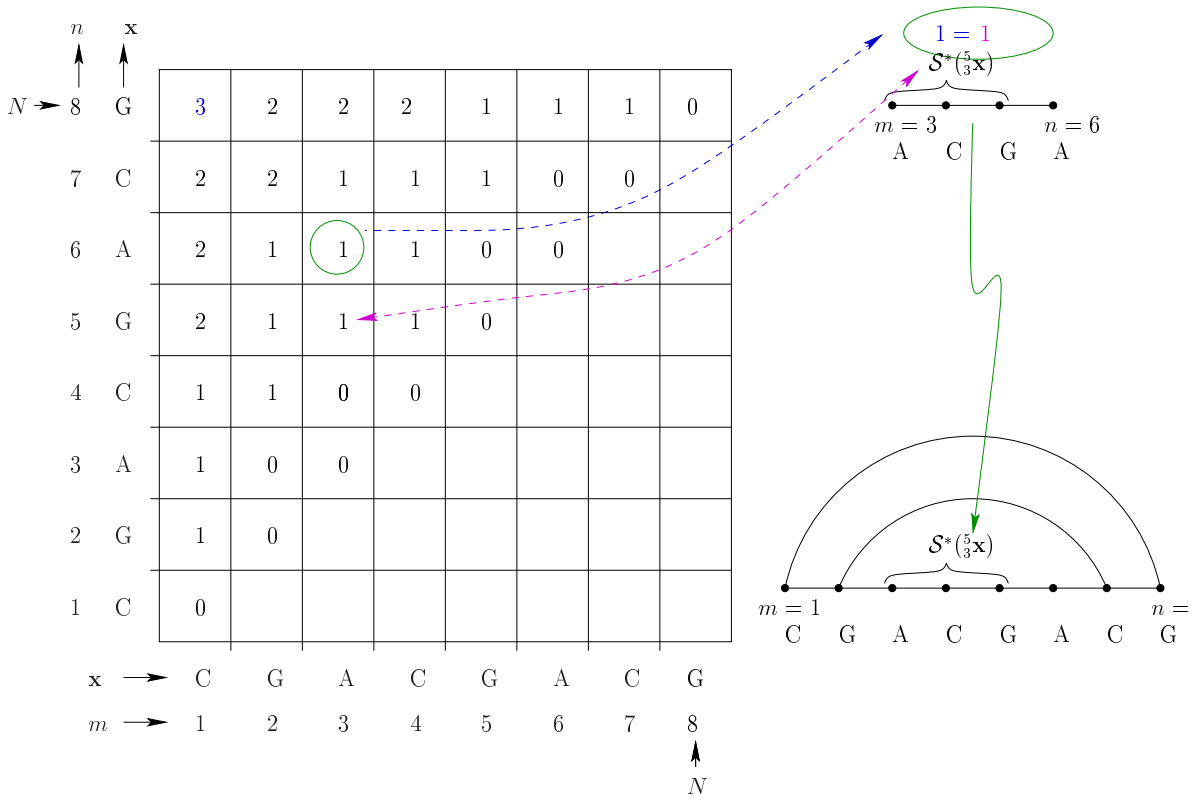
- Note traceback picks first optimal structure it encounters ($k = 1$)

Nussinov Traceback Example: Step 2



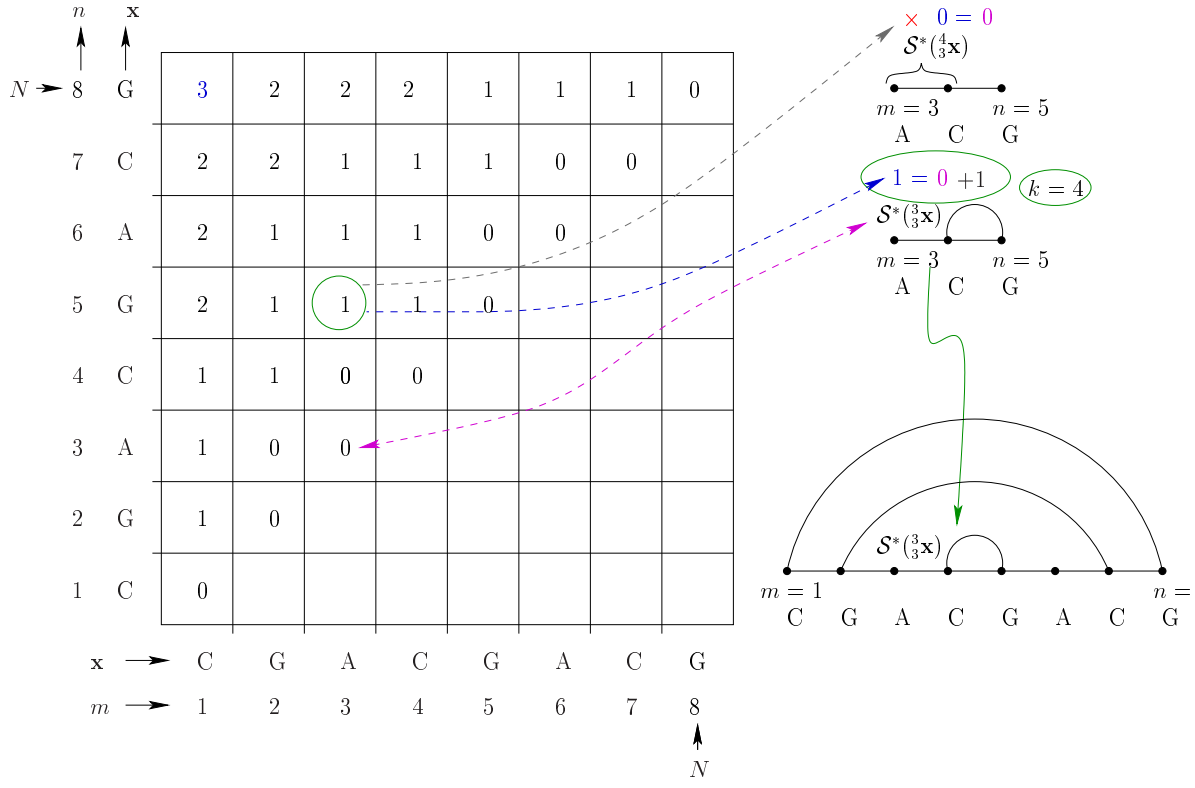
- Note traceback picks first optimal structure it encounters ($k = 2$)

Nussinov Traceback Example: Step 3



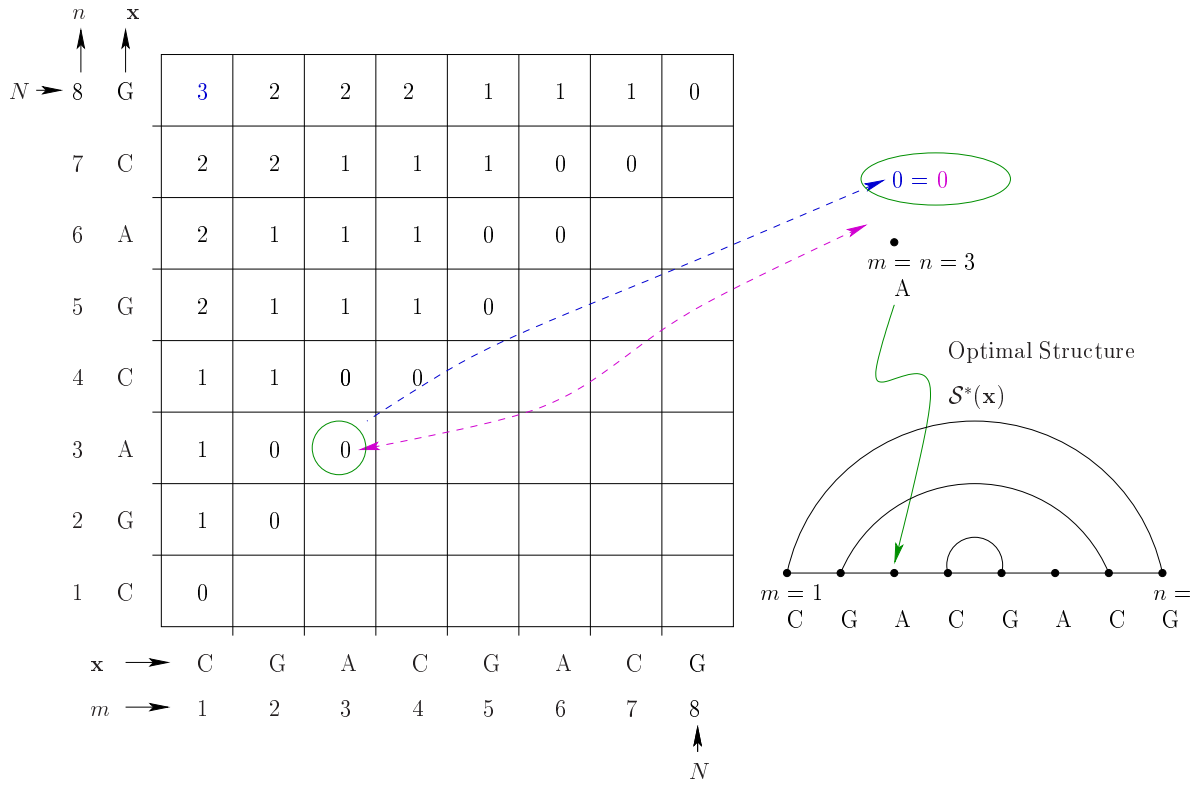
- Note traceback picks first optimal structure it encounters

Nussinov Traceback Example: Step 4



- Note traceback picks first optimal structure it encounters ($k = 4$)

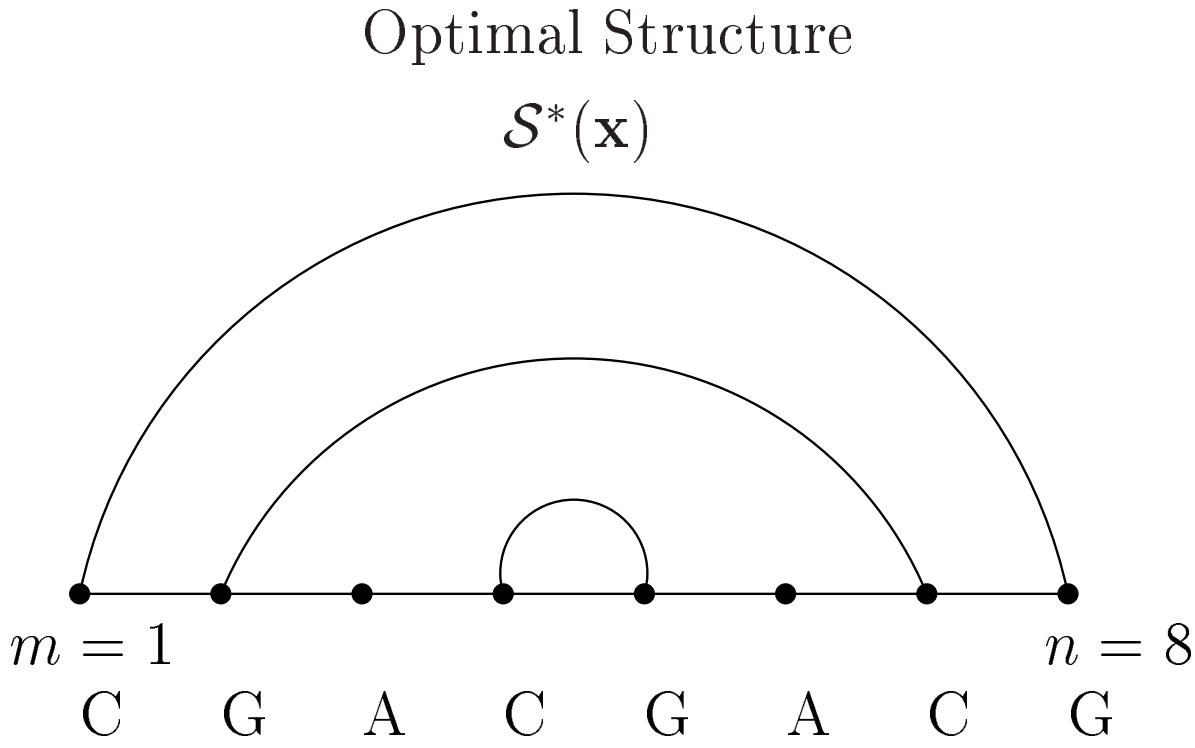
Nussinov Traceback Example: Step 5



- Note for single nucleotide sequence traceback return null (no base pairs) as the optimal structure

Nussinov Traceback Example: Final Optimal Structure

- (An) Optimal structure from traceback
  - Easy to see that number of base pairs is $d(1, N) = 3$



Nussinov Dynamic Programming Algo.: Traceback Notes

- Can have ties for max while filling array D, traceback algo. picks one based on order of its search
  - Can modify also to get all optimal structures
- Note bifurcation in the k loop for the traceback
  - Incorporated as a "recursive-function" in the traceback algorithm statement
  - Usu. implemented as a push-down (last-in first-out) stack
  - Push pair of indices (m, n) onto the stack identifying that the optimal substructure between (m, n) is part of the overall optimal structure (see next slide)
  - Not exercised in our simple example (see Assignment)
- Complexity of traceback is $O(N)$
  - Each step reduces remaining seq. size by atleast 1
    - Non-bifurcating cases reduce length by 1 (on the right)
    - Bifurcation cases reduce length by 2 (one on either side)

Nussinov Dynamic Programming Algorithm: Traceback II

Input: x, $d(\cdot,\cdot)$ // Sequence, Filled Dynamic Programming Array
Given: Initialized stack, length N for seq. x, set $\mathcal{C}$ of complementary bases
Output: $\mathcal{S}^*(x)$ // An optimal structure for x
Initialization:
$\mathcal{S}^*(x) \leftarrow \phi$ // Initialize to empty structure
Push $(1, N)$ onto stack // Determine optimal structure for entire seq. (nt. 1 to N)
while stack *is non-empty* **do**
  Pop (m, n) from stack // traceback from $d[m][n]$ determines substructure for ${}^{n}_{m}x$
  if $n > m$ **then** // Note: this subdomain is done when $m \leq n$
    if $d[m][n] = d[m][n-1]$ **then** Push $(m, n-1)$ onto stack // n·
    else for $k \leftarrow m$ **to** $(n-1)$ **do** // find optimal k such that $n-k$
      if $(x_k, x_n) \in \mathcal{C}$ ***and***
       $d(m,n) = d(m, \max(k-1, m)) + d(k+1, n-1) + 1$ **then**
        $\mathcal{S}^*(x) \leftarrow \mathcal{S}^*(x) \bigcup \{(k, n)\}$ // Record $n-k$
        Push $(m, k-1)$ and $(k+1, n-1)$ onto stack // nested subs.
      end
    end
    // Nothing to do if $m \leq n$
  end
end

Algorithm 2: Nussinov Traceback Using Push-down Stack

Nussinov Dynamic Programming Complexity

- Time complexity
  - Array fill step $O(N^3)$
  - Traceback $O(N)$
  - Array filling is the speed limiting step
- Memory requirement is $O(N^2)$
  - Obvious from algorithm description and example

RNA Structure Prediction: Additional Considerations

- Usually loops are constrained to be longer than a minimum length
  - Incorporated in definition of search space
    - Allow only pairs (i, j) such that $(j - i) > L$ where L is the minimum loop length
    - Minor modifications to approach developed (Assignment)
- Programming also requires taking care of boundary conditions
  - Sometimes convenient to initialize diagonal below the main diagonal also to 0
- Output structures represented in various alternative formats:
  - dot-bracket (dot represent unpaired nts and matching brackets represent pairs)
  - 2D diagrams (Circle, line+arc)
  - Tree-like representation

RNA Structure Prediction: Comments

- Maximizing base pairs as an "Optimality" criterion?
  - Simplistic idea generalizes to more useful definitions of optimal
  - Specific useful versions: minimum free energy or SCFG, both are probabilistic models
- Can also modify algorithm to output suboptimal secondary structures
  - Wuchty [11]
- Extensions have been developed that also allow for determination of optimal structures with some classes of pseudo-knots
  - Dirks and Pierce [5]
  - Can still be formulated as a dynamic program
    - Considerable increase in computational complexity $O(N^6)$

Stochastic Context Free Grammars

- RNA base pair maximization example
  - Similarities with string edit distance
    - Dynamic programming solution in both cases
    - Heuristically defined metric instead of a formal probabilistic model
  - Difference with string edit distance
    - Nussinov dynamic program based on inside-outside decomposition instead of "past and future" decomposition used for string edit
    - Different algorithmic structure
- Formal probabilistic model?
  - Two commonly used versions
    - Stochastic context free grammars (SCFGs, will discuss next)
    - Physics motivated free-energy models
- What does grammar have to do with it?
  - Original motivations came from natural languages and their grammar

Context Free Grammar Formal Definition

- A context free grammar (CFG) is a generative model defined by a 4 -tuple $\mathcal{G} = (\mathcal{V}, \mathcal{A}, \mathcal{R}, S)$ where
  - $\mathcal{V}$ is a finite set of non-terminals (unobserved)
  - $\mathcal{A}$ is a finite set of terminals (observed)
  - $\mathcal{R}$ is a finite set of production rules of the form $\phi \rightarrow \psi$, where $\phi \in \mathcal{V}$ and $\psi \in (\mathcal{V} \bigcup \mathcal{A})^*$
  - $S \in \mathcal{V}$ is the special start symbol
- Called "context-free" because LHS of production rules can contain only a single symbol, which must be a non-terminal
  - excludes context-dependent rules, such as $aWb \rightarrow aaWbb$ or $Tb \rightarrow bT$
- $\mathcal{L}(\mathcal{G})$ used to denote language generated by the grammar $\mathcal{G}$

Stochastic Context Free Grammars

- Context free grammars where there are probabilities associated with the production rules
- The resulting model can give likelihood of a given string

Example:

- Rules F : $S \underset{p_a}{\longrightarrow} aS, S \underset{p_b}{\longrightarrow} bS, S \underset{p_{ab}}{\longrightarrow} aSb, S \underset{p_\epsilon}{\longrightarrow} \epsilon$
- Starting with S, the derivations of abb with this grammar are:
  - $S \Rightarrow aS \Rightarrow abS \Rightarrow abbS \Rightarrow abb$ with probability $p_a p_b p_b p_\epsilon$
  - $S \Rightarrow aSb \Rightarrow abSb \Rightarrow abb$ with probability $p_{ab} p_b p_\epsilon$
- Probability of abb in this grammar is $p_a p_b p_b p_\epsilon + p_{ab} p_b p_\epsilon$

Stochastic Context Free Grammar Formal Definition

- A stochastic context free grammar (SCFG) is a generative probabilistic model defined by a 5 -tuple $\mathcal{G} = (\mathcal{V}, \mathcal{A}, \mathcal{R}, S, \mathcal{P})$ where the 4 - tuple $(\mathcal{V}, \mathcal{A}, \mathcal{R}, S)$ forms a CFG, and $\mathcal{P} : \mathcal{R} \mapsto [0, 1]$ apportions probabilities to the production rules for each non-terminal, i.e., $\sum_{\psi} \mathcal{P}(\phi \to \psi) = 1$, for all $\phi \in \mathcal{V}$
- Given a string x in the language $\mathcal{L}$ generated by the SCFG, the probability of a derivation $S \to \psi_1 \to \psi_2 \cdots \to \psi_M = \text{x}$ that generates the string x is given by the product $\mathcal{P}(S \to \psi_1) \prod_{n=1}^{M-1} \mathcal{P}(\psi_n \to \psi_{n+1})$
- The probability of the string x in the language $\mathcal{L}$ can be obtained by summing up (marginalizing) over all derivations that produce x
- Grammar is unambiguous if only one derivation exists for every string in the language
  - an unambiguous grammar provides some advantages but usually there will be multiple parses for a given string from the language

Three Fundamental SCFG Problems

- Given an observation sequence $\mathrm{x} = \{x_1, x_2, \ldots, x_N\}$ and a SCFG ($\mathcal{G}$), what is the probability (or likelihood) of x, $P(\mathrm{x}|\mathcal{G})$
- Given an observation sequence $\mathrm{x} = \{x_1, x_2, \ldots, x_N\}$, what is the optimal parse tree that best explains x for a given SCFG?
- Given an observation sequence x, what is the set of optimal SCFG parameters that maximizes likelihood of x?
- Analogous to the three HMM problems

Summary: Stochastic Context Free Grammars (SCFGs) I

- Commonality of techniques with other probabilistic models/algorithms (particularly HMMs)
  - Dynamic programming
    - Includes both CYK and inside-out algorithms
  - Computation of marginal probabilities
    - Can be cast as an instance of belief propagation
- Implementation issues
  - Similar to HMMs
  - Require scaling/log-domain implementation to avoid underflow

Summary: Stochastic Context Free Grammars (SCFGs) II

- Inspired by natural languages
  - SCFG = CFG + Probabilities for production rules
- Offer useful generalization of HMMs
  - Can capture wider range of dependencies than is feasible with HMMs (inside-out)
  - Computationally more demanding but still tractable for inference/estimation tasks
- Have been used in a variety of problems
  - Natural language understanding
  - Bioinformatics (RNA secondary structure)
  - Probabilistic framework can often provide advantages over more ad hoc heuristics
    - Specifically, parameter estimation from data and local estimates of uncertainty
    - Have also inspired approximations that reduce computational complexity

References I

[1] Noam Chomsky. "On certain formal properties of grammars". In: *Info. and Cntrl.* 2.2 (1959), pp. 137 –167. ISSN: 0019-9958. DOI: https://doi.org/10.1016/S0019-9958(59)90362-6.

[2] Noam Chomsky. *Syntactic Structures*. The Hague, Netherlands: Mouton & Co., 1957.

[3] Noam Chomsky. "Three Models for the Description of Language". In: *IRE Transactions on Information Theory* 2 (1956), pp. 113–123.

[4] William John Cocke and Jacob T Schwartz. *Programming languages and their compilers*. New York, NY: Courant Institute of Mathematical Sciences, 1970.

References II

[5] Robert M. Dirks and Niles A. Pierce. "A partition function algorithm for nucleic acid secondary structure including pseudoknots". In: *Journal of Computational Chemistry* 24.13 (2003), pp. 1664–1677. ISSN: 1096-987X. DOI: 10.1002/jcc.10296. URL: http://dx.doi.org/10.1002/jcc.10296.

[6] J. A. Doudna and T. R. Cech. "The chemical repertoire of natural ribozymes". In: *Nature* 418.6894 (2002), pp. 222–228.

[7] J.A. Doudna and J.H. Cate. "RNA structure: crystal clear?" In: *Current Opinions in Structural Biology* 7 (1997), pp. 310–316.

[8] R. Durbin et al. *Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids*. Cambridge, UK: Cambridge University Press, 1999. ISBN: 0521629713.

References III

[9] Tadao Kasami. *An efficient recognition and syntax-analysis algorithm for context-free languages*. Tech. rep. AFCRL-65-758. Bedford, MA: Air Force Cambridge Research Laboratory, 1965.

[10] Richard B. Waring and R. Wayne Davies. "Assessment of a model for intron RNA secondary structure relevant to RNA self-splicing – a review". In: *Gene* 28.3 (1984), pp. 277–291.

[11] S. Wuchty et al. "Complete suboptimal folding of RNA and the stability of secondary structures". In: *Biopolymers* 49 (1999), pp. 145–165.

[12] Daniel H Younger. "Recognition and parsing of context-free languages in time n^3". In: *Information and control* 10.2 (1967), pp. 189–208.

References IV

[13] Michael Zuker and David Sankoff. "RNA secondary structures and their prediction". In: *B. Math. Biol.* 46.4 (1984), pp. 591–621.