DNA sequence design

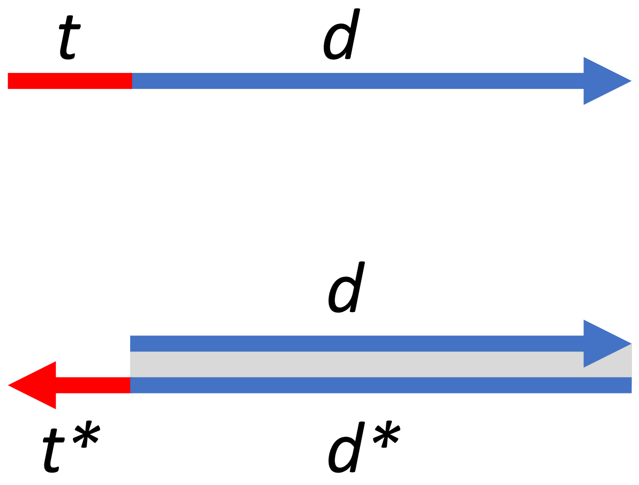
slides © 2021, David Doty

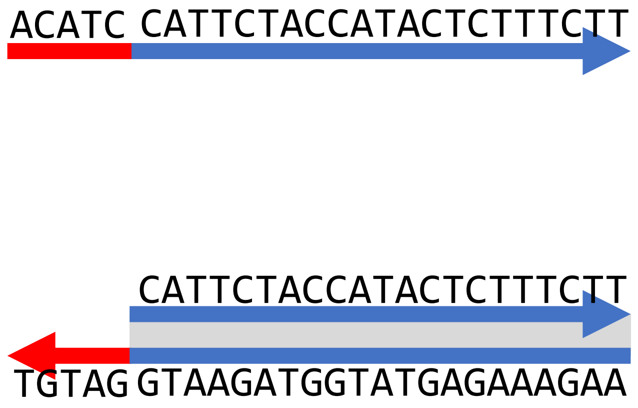
ECS 232: Theory of Molecular Computation, UC Davis

Two layers of abstraction in DNA nanotech

DNA *strands* with abstract "binding domains"

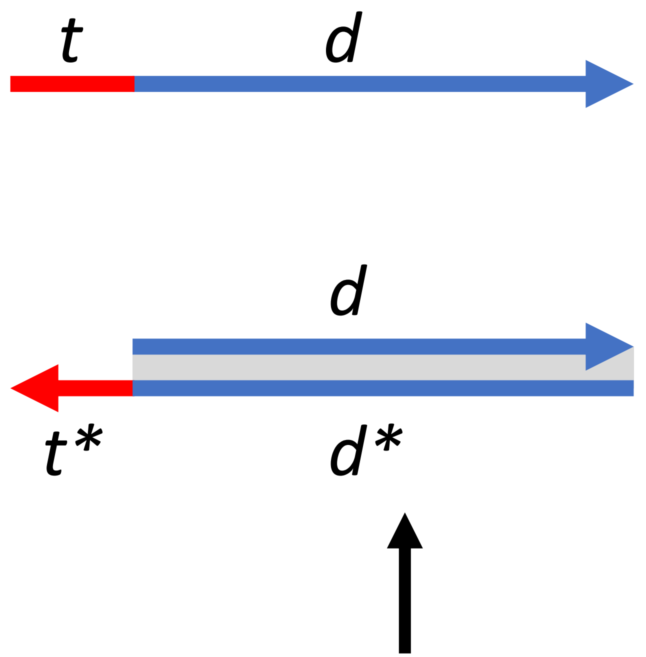
DNA *sequences*





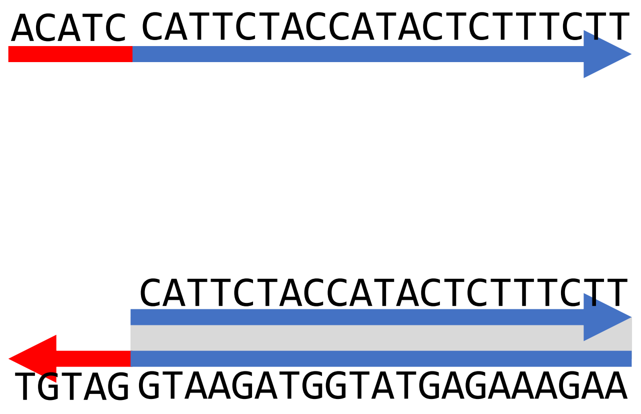
Two layers of abstraction in DNA nanotech

DNA *strands* with abstract "binding domains"



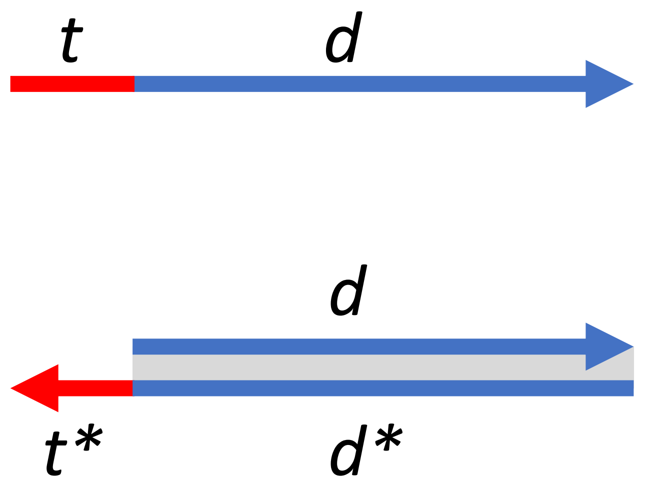
This describes ideally how we **want** strands to bind.

DNA *sequences*



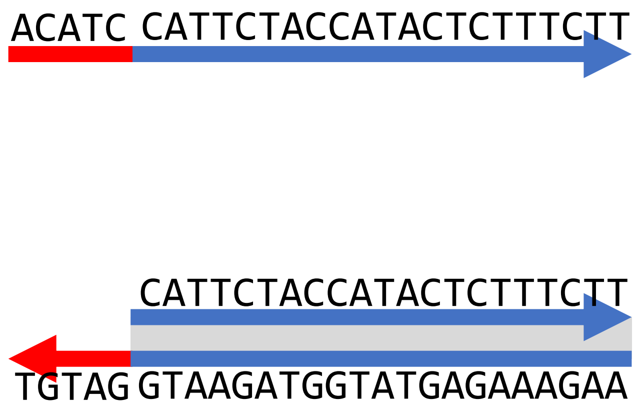
Two layers of abstraction in DNA nanotech

DNA *strands* with abstract "binding domains"



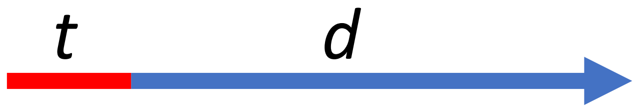
How to **design** DNA sequences to achieve "ideal" binding?

DNA *sequences*

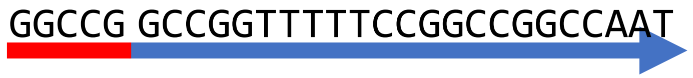


This describes ideally how we **want** strands to bind.

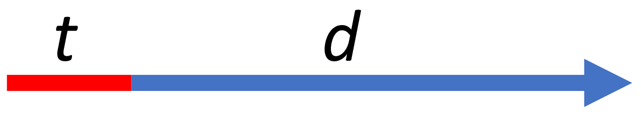
DNA sequence design



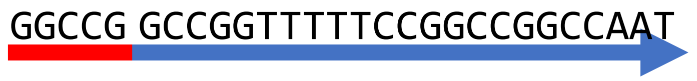
bad choice of DNA sequence



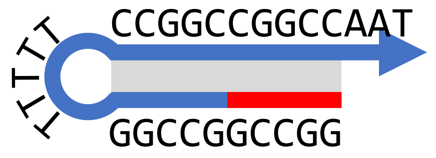
DNA sequence design



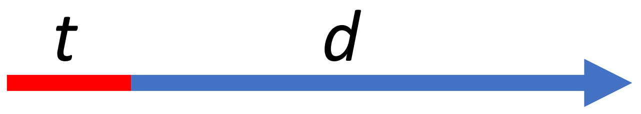
bad choice of DNA sequence



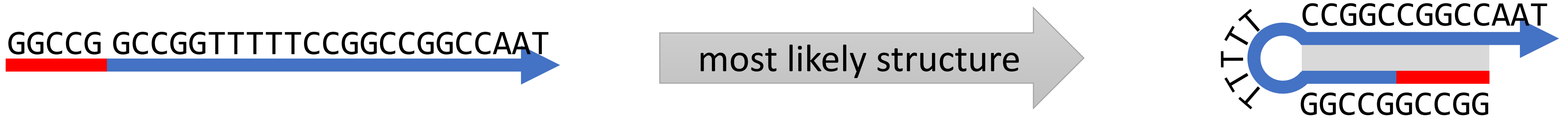
most likely structure



DNA sequence design



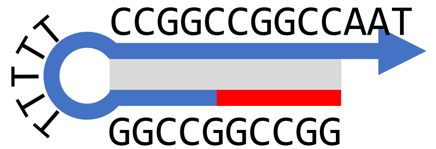
bad choice of DNA sequence



Why is this bad?
If we want the strand to bind to other strands, it first has to break up its own structure.
i.e., *effective* binding rate/strength is lowered

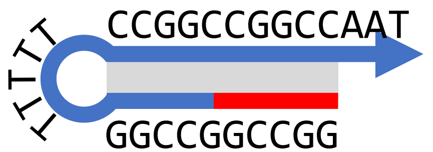
Common DNA sequence design goals: **What to avoid**

- Excessive secondary structure of strands

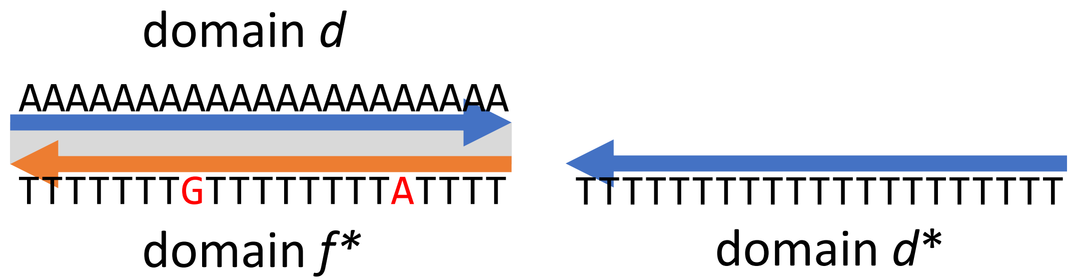


Common DNA sequence design goals: <u>**What to avoid**</u>

- Excessive secondary structure of strands

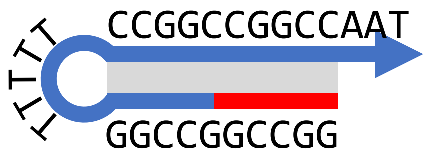


- Significant interaction between non-complementary domains

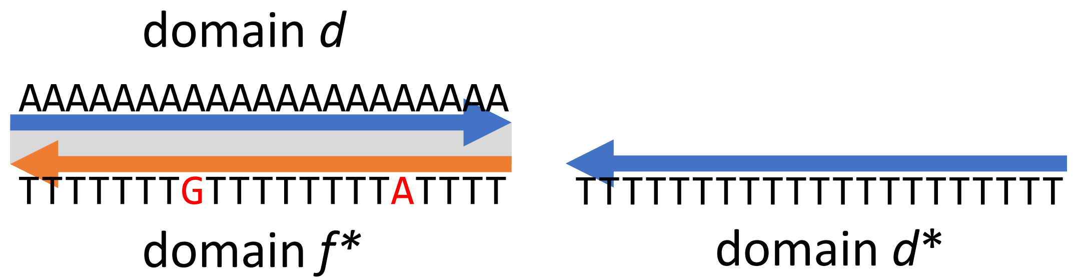


Common DNA sequence design goals: <u>**What to avoid**</u>

- Excessive secondary structure of strands



- Significant interaction between non-complementary domains



- Certain string-based rules, e.g.
  - some patterns such as GGGG (forms "*G-tetraplex*": https://www.idtdna.com/pages/education/decoded/article/g-repeats-structural-challenges-for-oligo-design)
  - > 70 % or < 30% G/C content (G/C binds more strongly)
  - domains ending in A/T (they "breathe" more)

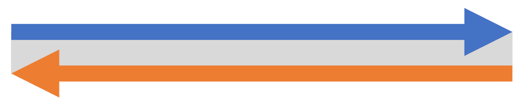
- And often other constraints

DNA energy models

How do we predict what structures DNA strands are likely to form?

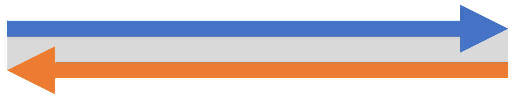
DNA duplex energy model (simple versions)

- How strongly does a DNA strand bind to its perfect complement?



DNA duplex energy model (simple versions)

- How strongly does a DNA strand bind to its perfect complement?
- 1st approximation: proportional to length:
  - Δ*G*(5'-AAGGTTAC-3' ,
    3'-TTCCAATG-5') = 1+1+1+1+1+1+1+1 = 8



DNA duplex energy model (simple versions)

- How strongly does a DNA strand bind to its perfect complement?
- 1st approximation: proportional to length:
  - Δ*G*(5'-AAGGTTAC-3' ,
    3'-TTCCAATG-5') = 1+1+1+1+1+1+1+1 = 8

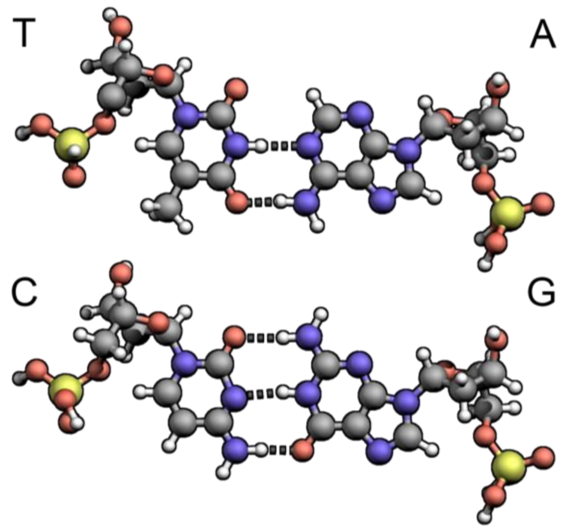
- 2nd approximation: depends on base pair:
  - G/C about twice as strong as A/T
  - Δ*G*(5'-AAGGTTAC-3' ,
    3'-TTCCAATG-5') = 1+1+2+2+1+1+1+2 = 11

DNA duplex energy model (simple versions)

- How strongly does a DNA strand bind to its perfect complement?
- 1st approximation: proportional to length:
  - Δ*G*(5'-AAGGTTAC-3' ,
    3'-TTCCAATG-5') = 1+1+1+1+1+1+1+1 = 8

- 2nd approximation: depends on base pair:
  - G/C about twice as strong as A/T
  - Δ*G*(5'-AAGGTTAC-3' ,
    3'-TTCCAATG-5') = 1+1+2+2+1+1+1+2 = 11
- 3rd approximation: nearest neighbor model (used in practice):
  - depends on base pair, *and* on the neighboring base pairs

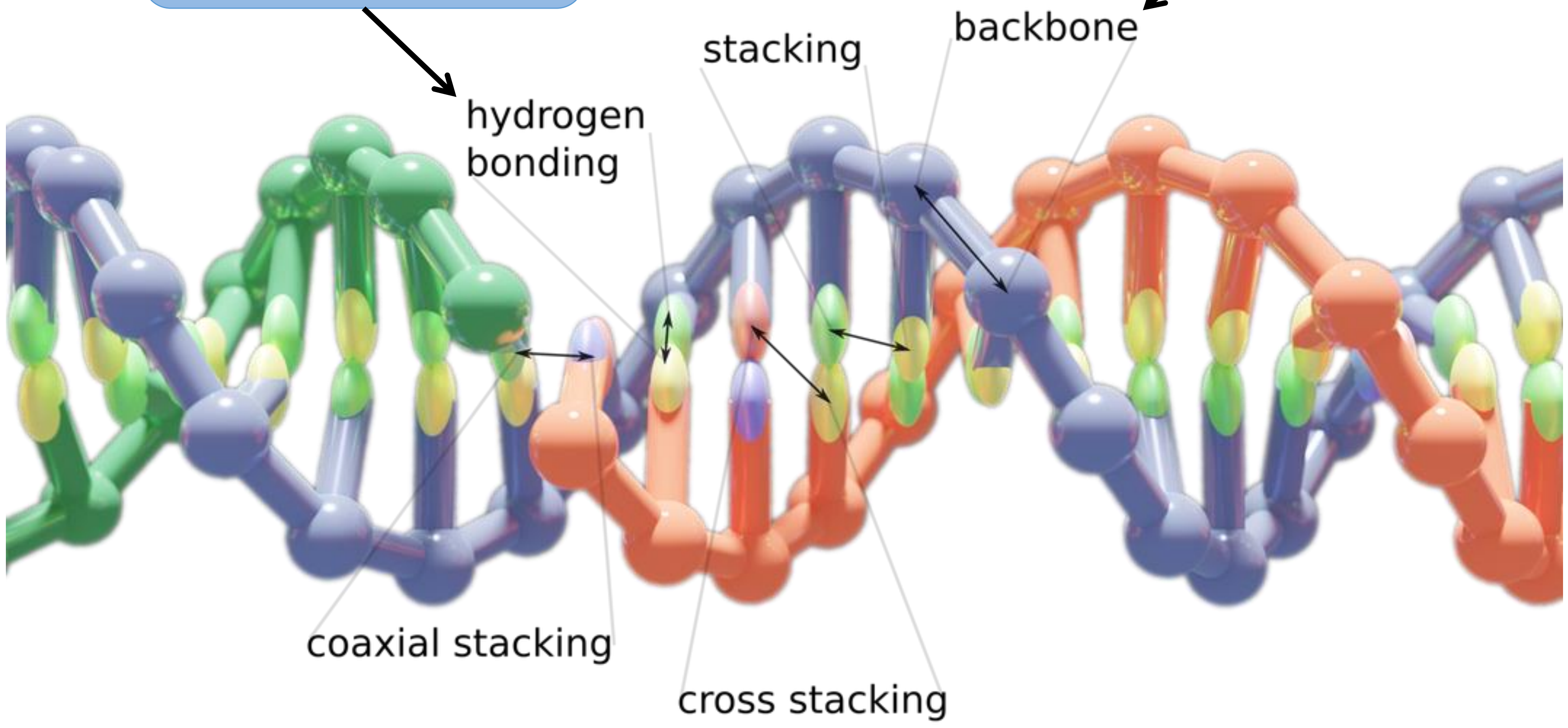
Why do the neighbors matter?



Much of DNA stability is not from base pair (formed by hydrogen bonds) but from “stacking” interactions between adjacent bases.

Responsible for specific base-pairing

Holds adjacent bases together on a single strand (strong covalent bond)



source: https://dna-robotics.eu/2019/11/29/simulating-dna/

Nearest neighbor energy model

```
      ↓   ↓   ↓
5′ C-G-T-T-G-A 3′
   *  *  *  *  *  *
3′ G-C-A-A-C-T 5′
        ↑   ↑
```

$$\Delta G^\circ_{37}(\text{pred.}) = \Delta G^\circ(\text{CG/GC}) + \Delta G^\circ(\text{GT/CA}) + \Delta G^\circ(\text{TT/AA})$$

$$+ \Delta G^\circ(\text{TG/AC}) + \Delta G^\circ(\text{GA/CT}) + \Delta G^\circ(\text{init.})$$

$$= -2.17 - 1.44 - 1.00 - 1.45 - 1.30 + \boxed{0.98 + 1.03}$$

$$\Delta G^\circ_{37}(\text{pred.}) = -5.35 \text{ kcal/mol}$$

$$\Delta G^\circ_{37}(\text{obs.}) = -5.20 \text{ kcal/mol}$$

ΔG_{init} = penalty for bringing together two strands (TODO: maybe not… not explained in paper) (*different terms if end is C/G or A/T*)

Table 1. Compari

Sequence	Unified (ref. 22)
AA/TT	−1.00
AT/TA	−0.88
TA/AT	−0.58
CA/GT	−1.45
GT/CA	−1.44
CT/GA	−1.28
GA/CT	−1.30
CG/GC	−2.17
GC/CG	−2.24
GG/CC	−1.84
Average	−1.42

[*A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics*, John SantaLucia Jr., PNAS 1998]

Energy of non-duplex secondary structures

What about DNA strands that are not perfectly complementary, but *some* bases match?

Energy of **non-duplex** secondary structures

What about DNA strands that are not perfectly complementary, but *some* bases match?

Definition: A secondary structure of a set of DNA strands is a set of base pairs among complementary bases.
Formally, it is a *matching* on the graph $G=(V,E)$, where
V = { bases in each strand }
$E = \{ \{u,v\} \mid \{u,v\} = \{A,T\} \text{ or } \{u,v\} = \{G,C\} \}$

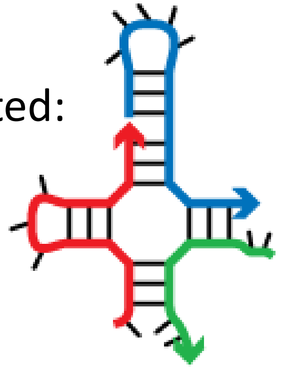
Energy of **<u>non-duplex</u>** secondary structures

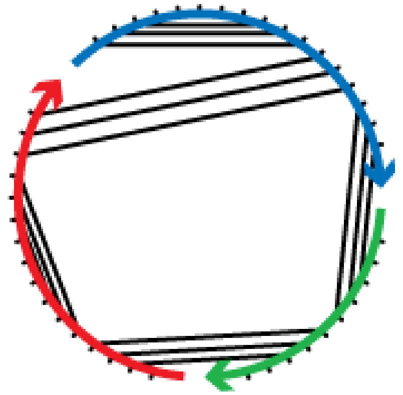
What about DNA strands that are not perfectly complementary, but *some* bases match?

Definition: A <u>secondary structure</u> of a set of DNA strands is a set of base pairs among complementary bases. Formally, it is a *matching* on the graph $G=(V,E)$, where
V = { bases in each strand }
E = { $\{u,v\}$ | $\{u,v\}$ = {A,T} or $\{u,v\}$ = {G,C} }

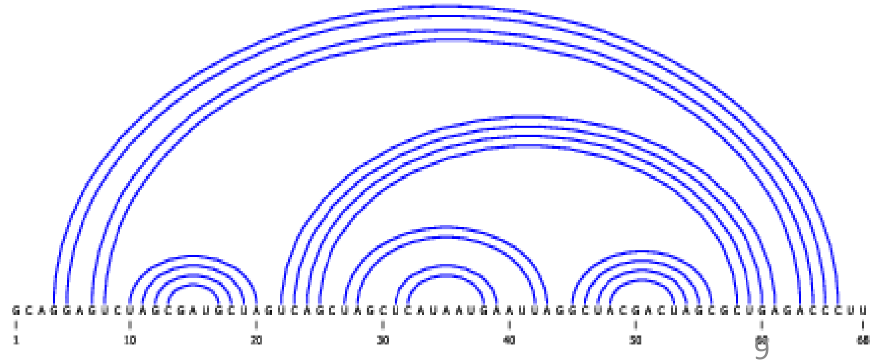
Definition: A secondary structure is <u>unpseudoknotted</u> (with respect to a particular circular ordering of the strands) if, drawing strands in 5'-3' order in a *circle* and connecting the base pairs by *straight lines*, **no lines cross**.

unpseudoknotted:





sometimes drawn with strands straight and base pairs as curved arcs:



G C A G G A G U C U A G C G A U G C U A G U C A G C U A G C U C A U A A U G A A U U A G G C U A C G A C U A G C G C U G A G A C C C U U
1
10
20
30
40
50
60
68

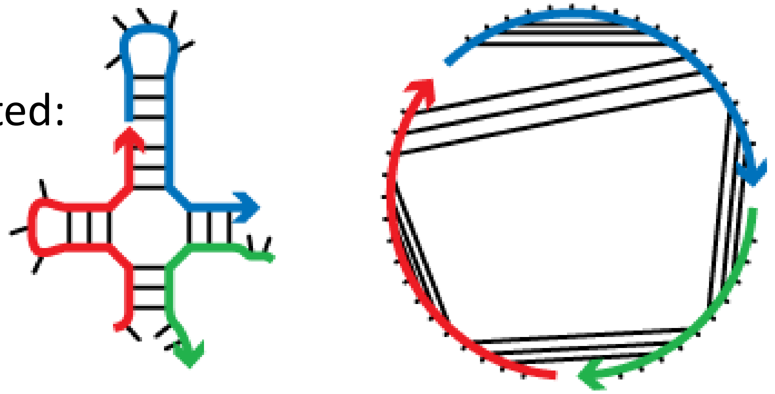
Energy of **non-duplex** secondary structures

What about DNA strands that are not perfectly complementary, but *some* bases match?

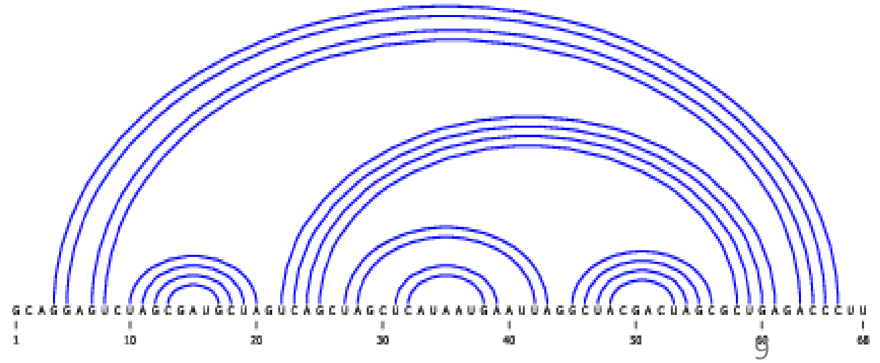
Definition: A secondary structure of a set of DNA strands is a set of base pairs among complementary bases. Formally, it is a *matching* on the graph $G=(V,E)$, where
V = { bases in each strand }
E = { $\{u,v\}$ | $\{u,v\}$ = {A,T} or $\{u,v\}$ = {G,C} }

Definition: A secondary structure is unpseudoknotted (with respect to a particular circular ordering of the strands) if, drawing strands in 5'-3' order in a *circle* and connecting the base pairs by *straight lines*, **no lines cross**.

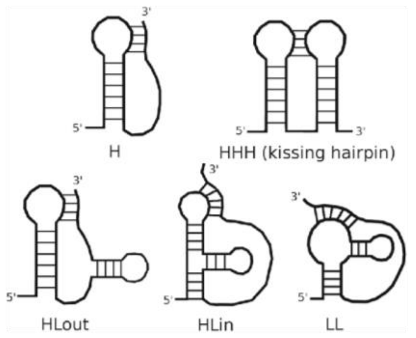
unpseudoknotted:



sometimes drawn with strands straight and base pairs as curved arcs:

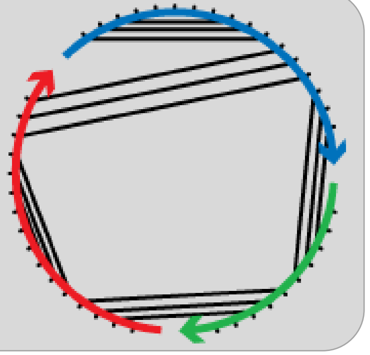

G C A G G A G U C U A G C G A U G C U A G U C A G C U A G C U C A U A A U G A A U U A G G C U A C G A C U A G C G C U G A G A C C C U U
1
10
20
30
40
50
60
68

pseudoknots:



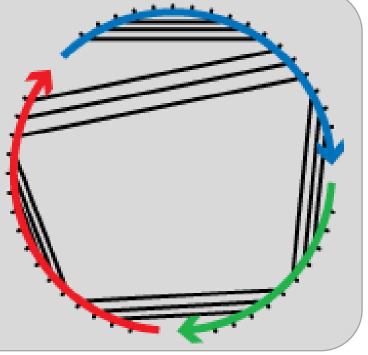
Equivalent definitions of unpseudoknotted

Definition 1: Drawing strands in 5’-3’ order in a *circle* and connecting the base pairs by *straight lines*, no lines cross.



Equivalent definitions of unpseudoknotted

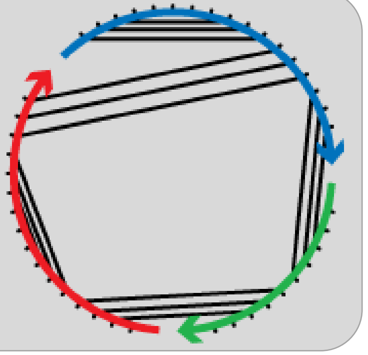
Definition 1: Drawing strands in 5'-3' order in a *circle* and connecting the base pairs by *straight lines*, no lines cross.



Definition 2: Base pair indices obey the **nesting property**: there are *no* base pairs $(a,b) \in \mathbb{N}^2$ and $(x,y) \in \mathbb{N}^2$ such that $a < x < b < y$ (e.g., it can be $a < b < x < y$ or $a < x < y < b$)

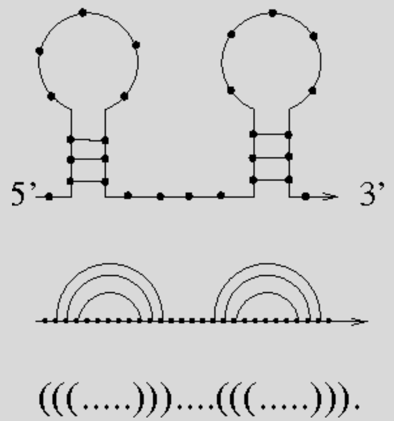
Equivalent definitions of unpseudoknotted

Definition 1: Drawing strands in 5'-3' order in a *circle* and connecting the base pairs by *straight lines*, no lines cross.



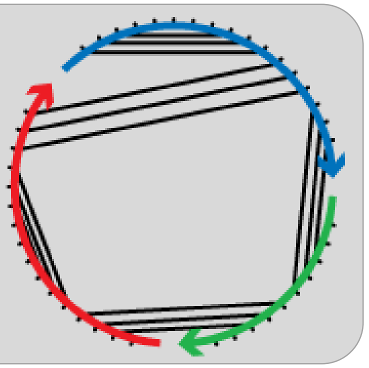
Definition 2: Base pair indices obey the **nesting property**: there are *no* base pairs (a,b) ∈ $\mathbb{N}^2$ and (x,y) ∈ $\mathbb{N}^2$ such that $a < x < b < y$ (e.g., it can be $a < b < x < y$ or $a < x < y < b$)

Definition 3: Balanced parentheses describe base pairs in **dot-parens** (a.k.a., **dot-bracket**) notation.



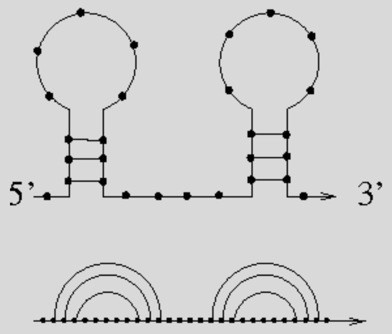
Equivalent definitions of unpseudoknotted

Definition 1: Drawing strands in 5'-3' order in a *circle* and connecting the base pairs by *straight lines*, no lines cross.

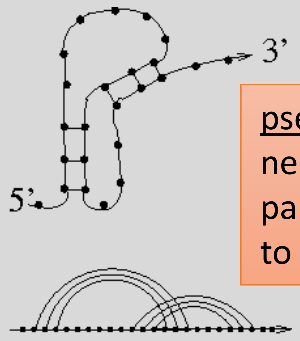


Definition 2: Base pair indices obey the **nesting property**: there are *no* base pairs (a,b) ∈ $\mathbb{N}^2$ and (x,y) ∈ $\mathbb{N}^2$ such that $a < x < b < y$ (e.g., it can be $a < b < x < y$ or $a < x < y < b$)

Definition 3: Balanced parentheses describe base pairs in **dot-parens** (a.k.a., **dot-bracket**) notation.



(((.....)))....(((.....))).

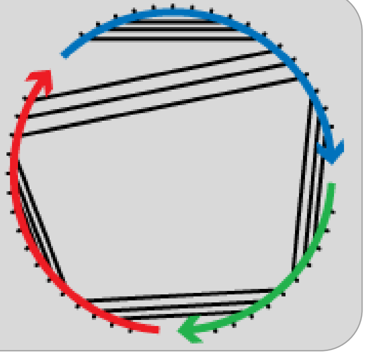


(((......[[[))...]]]..

pseudoknotted: need multiple parenthesis types to describe

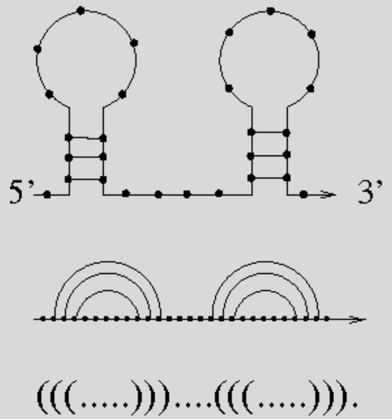
Equivalent definitions of unpseudoknotted

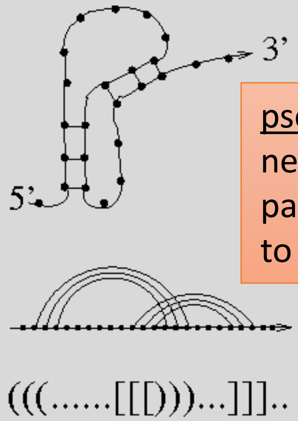
Definition 1: Drawing strands in 5'-3' order in a *circle* and connecting the base pairs by *straight lines*, no lines cross.



Definition 2: Base pair indices obey the **nesting property**: there are *no* base pairs $(a,b) \in \mathbb{N}^2$ and $(x,y) \in \mathbb{N}^2$ such that $a < x < b < y$ (e.g., it can be $a < b < x < y$ or $a < x < y < b$)

Definition 3: Balanced parentheses describe base pairs in **dot-parens** (a.k.a., **dot-bracket**) notation.

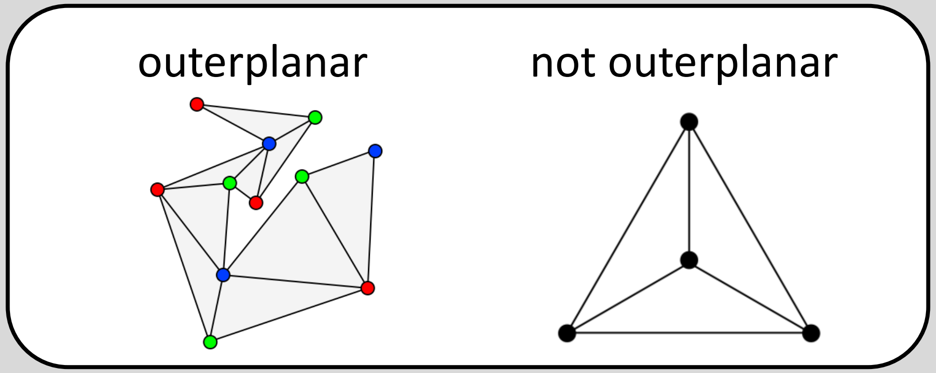




<u>pseudoknotted</u>: need multiple parenthesis types to describe

Definition 4: The graph $G=(V,E)$ is **outerplanar**, where
V = { bases in each strand }
E = { $\{u,v\}$ | $\{u,v\}$ are a paired base pair,
<u>or</u> $\{u,v\}$ are adjacent }

outerplanar = can be drawn with no edges crossing (planar), **and** all vertices incident to the outer face



Back to first approximation of energy model

- (For now, consider only one strand.)
- Given a DNA sequence *S*, what is the maximum number of base pairs that can be formed in any unpseudoknotted secondary structure?

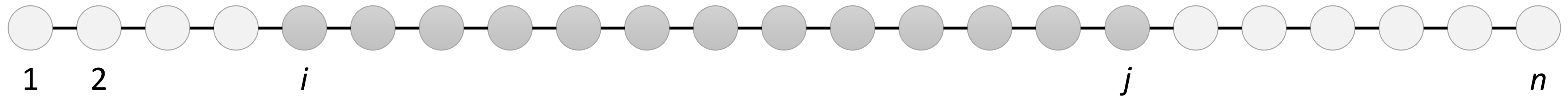
Back to first approximation of energy model

- (For now, consider only one strand.)
- Given a DNA sequence *S*, what is the maximum number of base pairs that can be formed in any unpseudoknotted secondary structure?
    - Without unpseudoknotted constraint, this is trivial: min(#C,#G) + min(#A,#T)

Back to first approximation of energy model

- (For now, consider only one strand.)
- Given a DNA sequence *S*, what is the maximum number of base pairs that can be formed in any unpseudoknotted secondary structure?
  - Without unpseudoknotted constraint, this is trivial: min(#C,#G) + min(#A,#T)
- Can be taken as a rough approximation of the minimum free energy structure of *S*, i.e., the most probable structure "at thermodynamic equilibrium" (*what you'd see if you heat it up and slowly cool it*).

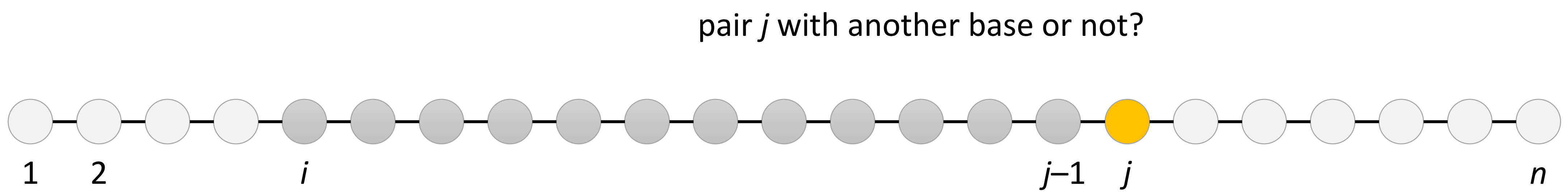
Computing maximally bound unpseudoknotted secondary structure in polynomial time



Recursive solution:

- Strand length is *n*.
- For 1 ≤ *i* ≤ *j* ≤ *n*, let OPT(*i*,*j*) = max base pairs possible using **only** bases *i* through *j*.

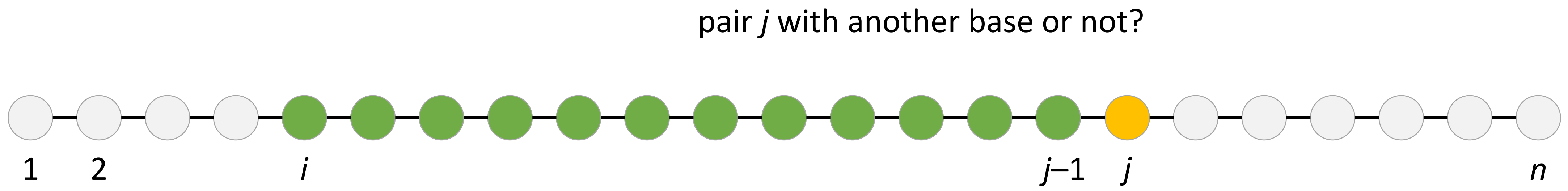
Computing maximally bound unpseudoknotted secondary structure in polynomial time



Recursive solution:

- Strand length is *n*.
- For 1 ≤ *i* ≤ *j* ≤ *n*, let OPT(*i*,*j*) = max base pairs possible using **only** bases *i* through *j*.
- Question: do we pair base *j* with some other base between *i* and *j*–1?

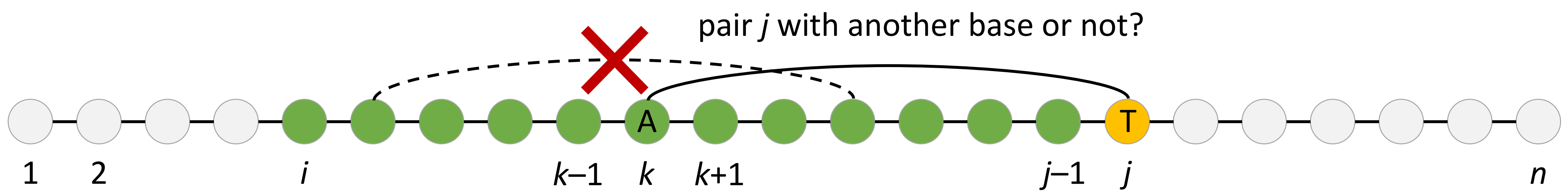
Computing maximally bound unpseudoknotted secondary structure in polynomial time



Recursive solution:

- Strand length is *n*.
- For 1 ≤ *i* ≤ *j* ≤ *n*, let OPT(*i*,*j*) = max base pairs possible using **only** bases *i* through *j*.
- <u>Question</u>: do we pair base *j* with some other base between *i* and *j*–1?
- If *not*, recursively, the optimal value is:
    - OPT(*i*,*j*) = OPT(*i*,*j*–1)

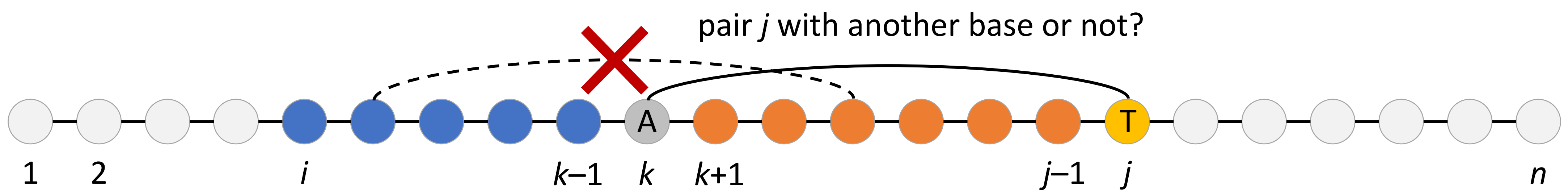
Computing maximally bound unpseudoknotted secondary structure in polynomial time



Recursive solution:

- Strand length is *n*.
- For 1 ≤ *i* ≤ *j* ≤ *n*, let OPT(*i*,*j*) = max base pairs possible using **only** bases *i* through *j*.
- Question: do we pair base *j* with some other base between *i* and *j*–1?
- If *not*, recursively, the optimal value is:
    - OPT(*i*,*j*) = OPT(*i*,*j*–1)
- If we pair *j* with *k*, **nesting property** implies no base pair can form between any base in [*i*,… *k*–1] and any base in [*k*+1,*j*–1]

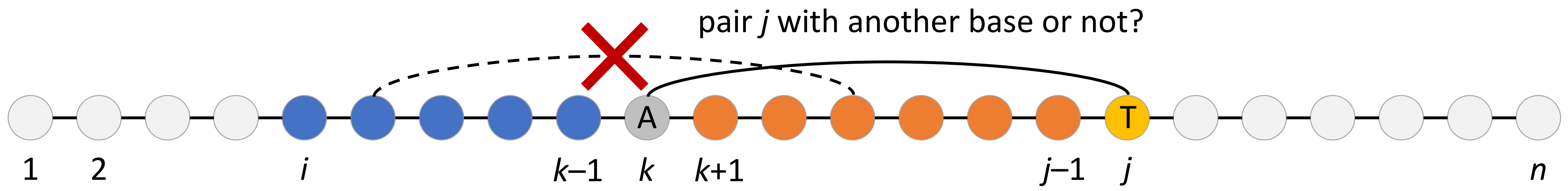
Computing maximally bound unpseudoknotted secondary structure in polynomial time



Recursive solution:

- Strand length is *n*.
- For 1 ≤ *i* ≤ *j* ≤ *n*, let OPT(*i*,*j*) = max base pairs possible using **only** bases *i* through *j*.
- Question: do we pair base *j* with some other base between *i* and *j*–1?
- If *not*, recursively, the optimal value is:
  - OPT(*i*,*j*) = OPT(*i*,*j*–1)
- If we pair *j* with *k*, **nesting property** implies no base pair can form between any base in [*i*,… *k*–1] and any base in [*k*+1,*j*–1]
- Recursively, optimal value depends on:
  - OPT(*i*,*k*–1) and OPT(*k*+1,*j*–1)

Computing maximally bound unpseudoknotted secondary structure in polynomial time



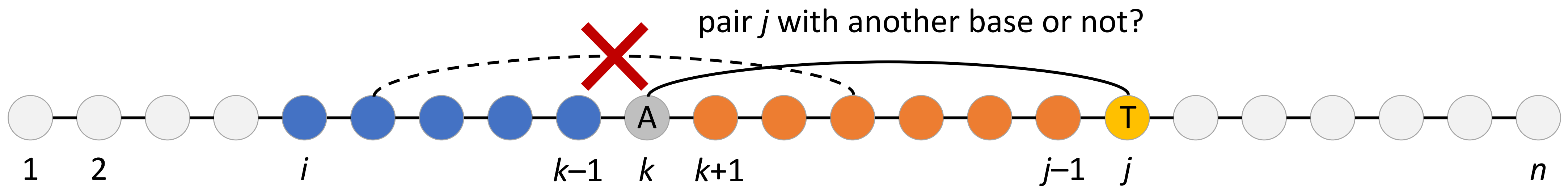
Recursive solution:

- Strand length is *n*.
- For $1 \le i \le j \le n$, let OPT(*i*,*j*) = max base pairs possible using **only** bases *i* through *j*.
- Question: do we pair base *j* with some other base between *i* and *j*–1?
- If *not*, recursively, the optimal value is:
  - OPT(*i*,*j*) = OPT(*i*,*j*–1)
- If we pair *j* with *k*, **nesting property** implies no base pair can form between any base in [*i*,… *k*–1] and any base in [*k*+1,*j*–1]
- Recursively, optimal value depends on:
  - OPT(*i*,*k*–1) and OPT(*k*+1,*j*–1)

Recursive algorithm (implement w/ dynamic programming):

OPT(*i*,*j*) = max of:

Computing maximally bound unpseudoknotted secondary structure in polynomial time



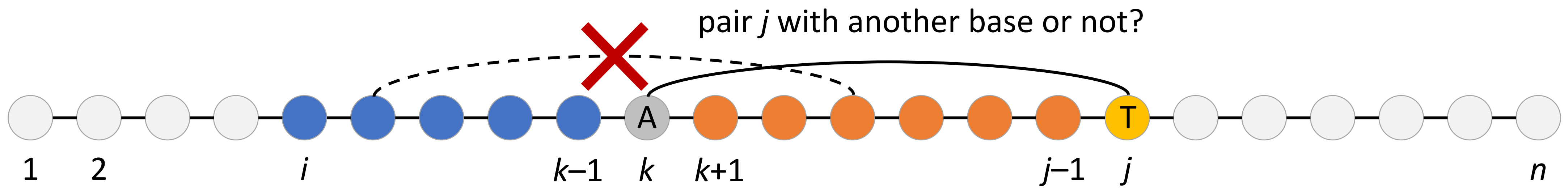
Recursive solution:

- Strand length is *n*.
- For 1 ≤ *i* ≤ *j* ≤ *n*, let OPT(*i*,*j*) = max base pairs possible using **only** bases *i* through *j*.
- Question: do we pair base *j* with some other base between *i* and *j*–1?
- If *not*, recursively, the optimal value is:
    - OPT(*i*,*j*) = OPT(*i*,*j*–1)
- If we pair *j* with *k*, **nesting property** implies no base pair can form between any base in [*i*,… *k*–1] and any base in [*k*+1,*j*–1]
- Recursively, optimal value depends on:
    - OPT(*i*,*k*–1) and OPT(*k*+1,*j*–1)

Recursive algorithm (implement w/ dynamic programming):

```
OPT(i,j) = max of:
   OPT(i,j–1),                    // don't form base pair with j
```

Computing maximally bound unpseudoknotted secondary structure in polynomial time



Recursive solution:

- Strand length is *n*.
- For $1 \leq i \leq j \leq n$, let OPT(*i*,*j*) = max base pairs possible using **only** bases *i* through *j*.
- Question: do we pair base *j* with some other base between *i* and *j*–1?
- If *not*, recursively, the optimal value is:
  - OPT(*i*,*j*) = OPT(*i*,*j*–1)
- If we pair *j* with *k*, **nesting property** implies no base pair can form between any base in [*i*,… *k*–1] and any base in [*k*+1,*j*–1]
- Recursively, optimal value depends on:
  - OPT(*i*,*k*–1) and OPT(*k*+1,*j*–1)

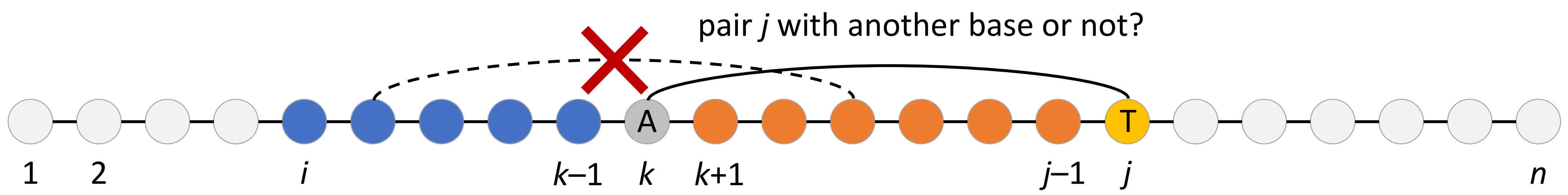
Recursive algorithm (implement w/ dynamic programming):

OPT(*i*,*j*) = max of:

  OPT(*i*,*j*−1), *// don't form base pair with j*

  $\max_{i \leq k < j}$ 1 + OPT(*i*,*k*−1) + OPT(*k*+1,*j*−1) *// form k,j base pair*

Computing maximally bound unpseudoknotted secondary structure in polynomial time



Recursive solution:

- Strand length is *n*.
- For 1 ≤ *i* ≤ *j* ≤ *n*, let OPT(*i*,*j*) = max base pairs possible using **only** bases *i* through *j*.
- Question: do we pair base *j* with some other base between *i* and *j*–1?
- If *not*, recursively, the optimal value is:
    - OPT(*i*,*j*) = OPT(*i*,*j*–1)
- If we pair *j* with *k*, **nesting property** implies no base pair can form between any base in [*i*,… *k*–1] and any base in [*k*+1,*j*–1]
- Recursively, optimal value depends on:
    - OPT(*i*,*k*–1) and OPT(*k*+1,*j*–1)

Recursive algorithm (implement w/ dynamic programming):

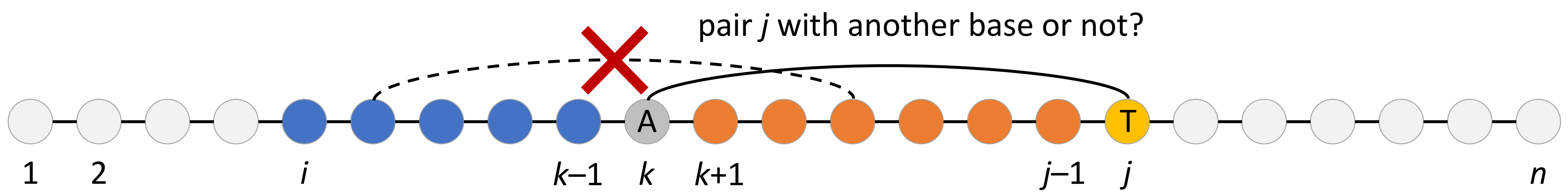
OPT(*i*,*j*) = max of:

only if *k* and *j* are complementary bases

$$\text{OPT}(i,j-1), \quad \textit{// don't form base pair with } j$$

$$\max_{i \le k < j} \; 1 + \text{OPT}(i,k-1) + \text{OPT}(k+1,j-1) \quad \textit{// form } k,j \textit{ base pair}$$

Computing maximally bound unpseudoknotted secondary structure in polynomial time



Recursive solution:

- Strand length is *n*.
- For $1 \leq i \leq j \leq n$, let OPT(*i*,*j*) = max base pairs possible using **only** bases *i* through *j*.
- Question: do we pair base *j* with some other base between *i* and *j*–1?
- If *not*, recursively, the optimal value is:
  - OPT(*i*,*j*) = OPT(*i*,*j*–1)
- If we pair *j* with *k*, **nesting property** implies no base pair can form between any base in [*i*,… *k*–1] and any base in [*k*+1,*j*–1]
- Recursively, optimal value depends on:
  - OPT(*i*,*k*–1) and OPT(*k*+1,*j*–1)

Recursive algorithm (implement w/ dynamic programming):

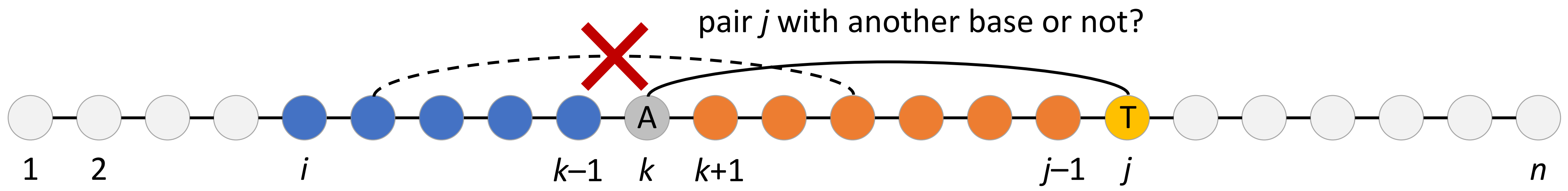
OPT(*i*,*j*) = max of: only if *k* and *j* are complementary bases

OPT(*i*,*j*−1), *// don't form base pair with j*

$\max_{i \leq k < j}$ 1 + OPT(*i*,*k*−1) + OPT(*k*+1,*j*−1) *// form k,j base pair*

base case: OPT(*i*,*i*) = 0

Computing maximally bound unpseudoknotted secondary structure in polynomial time



Recursive solution:

- Strand length is n.
- For $1 \leq i \leq j \leq n$, let OPT(i,j) = max base pairs possible using **only** bases i through j.
- Question: do we pair base j with some other base between i and j–1?
- If *not*, recursively, the optimal value is:
  - OPT(i,j) = OPT(i,j–1)
- If we pair j with k, **nesting property** implies no base pair can form between any base in [i,… k–1] and any base in [k+1,j–1]
- Recursively, optimal value depends on:
  - OPT(i,k–1) and OPT(k+1,j–1)

Recursive algorithm (implement w/ dynamic programming):

OPT(i,j) = max of: only if k and j are complementary bases

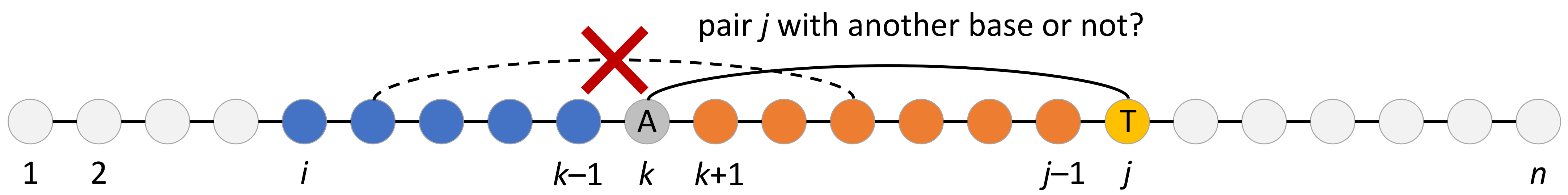
OPT(i,j–1), *// don't form base pair with j*

$\max_{i \leq k < j}$ 1 + OPT(i,k–1) + OPT(k+1,j–1) *// form k,j base pair*

base case: OPT(i,i) = 0

optimal value for whole strand = OPT(1,n)

Computing maximally bound unpseudoknotted secondary structure in polynomial time



Recursive solution:

- Strand length is *n*.
- For $1 \leq i \leq j \leq n$, let OPT(*i*,*j*) = max base pairs possible using **only** bases *i* through *j*.
- Question: do we pair base *j* with some other base between *i* and *j*–1?
- If *not*, recursively, the optimal value is:
  - OPT(*i*,*j*) = OPT(*i*,*j*–1)
- If we pair *j* with *k*, **nesting property** implies no base pair can form between any base in [*i*,… *k*–1] and any base in [*k*+1,*j*–1]
- Recursively, optimal value depends on:
  - OPT(*i*,*k*–1) and OPT(*k*+1,*j*–1)

Recursive algorithm (implement w/ dynamic programming):

OPT(*i*,*j*) = max of: only if *k* and *j* are complementary bases

  OPT(*i*,*j*−1), *// don't form base pair with j*

  $\max_{i \leq k < j}$ 1 + OPT(*i*,*k*−1) + OPT(*k*+1,*j*−1) *// form k,j base pair*

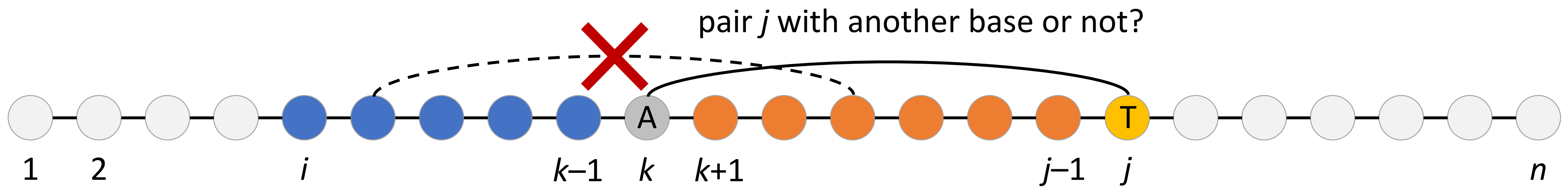
base case: OPT(*i*,*i*) = 0

optimal value for whole strand = OPT(1,*n*)

Running time:

There are $O(n^2)$ subproblems: choices *i*,*j* with $1 \leq i < j \leq n$.

Computing maximally bound unpseudoknotted secondary structure in polynomial time



Recursive solution:

- Strand length is *n*.
- For $1 \le i \le j \le n$, let OPT(*i*,*j*) = max base pairs possible using **only** bases *i* through *j*.
- Question: do we pair base *j* with some other base between *i* and *j*–1?
- If *not*, recursively, the optimal value is:
  - OPT(*i*,*j*) = OPT(*i*,*j*–1)
- If we pair *j* with *k*, **nesting property** implies no base pair can form between any base in [*i*,… *k*–1] and any base in [*k*+1,*j*–1]
- Recursively, optimal value depends on:
  - OPT(*i*,*k*–1) and OPT(*k*+1,*j*–1)

Recursive algorithm (implement w/ dynamic programming):

OPT(*i*,*j*) = max of:

only if *k* and *j* are complementary bases

$$\mathrm{OPT}(i,j-1), \quad \textit{// don't form base pair with } j$$

$$\max_{i \le k < j} \; 1 + \mathrm{OPT}(i,k-1) + \mathrm{OPT}(k+1,j-1) \quad \textit{// form } k,j \textit{ base pair}$$

base case: OPT(*i*,*i*) = 0

optimal value for whole strand = OPT(1,*n*)

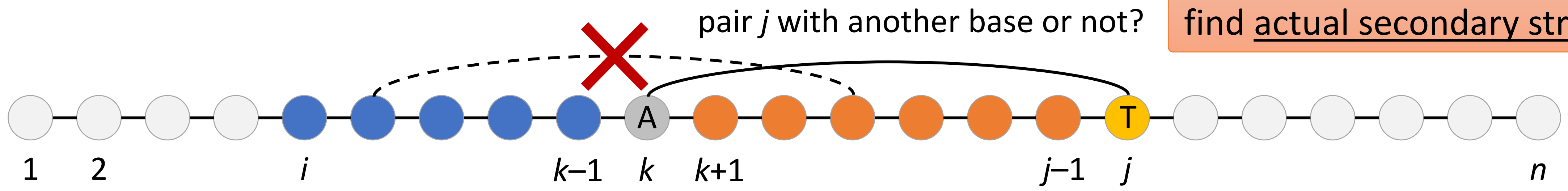
Running time:

There are $O(n^2)$ subproblems: choices *i*,*j* with $1 \le i < j \le n$.

Each takes time $O(n)$ to search all values of *k*, so $O(n^3)$ total.

Computing maximally bound unpseudoknotted secondary structure in polynomial time

This gives optimal *value*: how to find actual secondary structure?



Recursive solution:

- Strand length is *n*.
- For $1 \le i \le j \le n$, let OPT(*i*,*j*) = max base pairs possible using **only** bases *i* through *j*.
- Question: do we pair base *j* with some other base between *i* and *j*–1?
- If *not*, recursively, the optimal value is:
  - OPT(*i*,*j*) = OPT(*i*,*j*–1)
- If we pair *j* with *k*, **nesting property** implies no base pair can form between any base in [*i*,… *k*–1] and any base in [*k*+1,*j*–1]
- Recursively, optimal value depends on:
  - OPT(*i*,*k*–1) and OPT(*k*+1,*j*–1)

Recursive algorithm (implement w/ dynamic programming):

OPT(*i*,*j*) = max of:

OPT(*i*,*j*–1), // *don't form base pair with j*

$\max_{i \le k < j}$ 1 + OPT(*i*,*k*–1) + OPT(*k*+1,*j*–1) // *form k,j base pair*

only if *k* and *j* are complementary bases

base case: OPT(*i*,*i*) = 0

optimal value for whole strand = OPT(1,*n*)

Running time:

There are $O(n^2)$ subproblems: choices *i*,*j* with $1 \le i < j \le n$.

Each takes time $O(n)$ to search all values of *k*, so $O(n^3)$ total.

Example of dynamic programming algorithm

strand sequence =

ATTGATC

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1							
T	2							
T	3							
G	4							
A	5							
T	6							
C	7							

strand sequence =

ATTGATC

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1							
T	2	✗						
T	3	✗	✗					
G	4	✗	✗	✗				
A	5	✗	✗	✗	✗			
T	6	✗	✗	✗	✗	✗		
C	7	✗	✗	✗	✗	✗	✗	

strand sequence =

ATTGATC

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1							
T	2	✗						
T	3	✗	✗					
G	4	✗	✗	✗				
A	5	✗	✗	✗	✗			
T	6	✗	✗	✗	✗	✗		
C	7	✗	✗	✗	✗	✗	✗	

strand sequence =

ATTGATC

🟩 base cases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0						
T	2	✗						
T	3	✗	✗					
G	4	✗	✗	✗				
A	5	✗	✗	✗	✗			
T	6	✗	✗	✗	✗	✗		
C	7	✗	✗	✗	✗	✗	✗	

strand sequence =

ATTGATC

base cases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0						
T	2	✗	0					
T	3	✗	✗					
G	4	✗	✗	✗				
A	5	✗	✗	✗	✗			
T	6	✗	✗	✗	✗	✗		
C	7	✗	✗	✗	✗	✗	✗	

strand sequence =

A$\underline{\text{T}}$TGATC

base cases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0						
T	2	✗	0					
T	3	✗	✗	0				
G	4	✗	✗	✗				
A	5	✗	✗	✗	✗			
T	6	✗	✗	✗	✗	✗		
C	7	✗	✗	✗	✗	✗	✗	

strand sequence =

ATTGATC

base cases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0						
T	2	✗	0					
T	3	✗	✗	0				
G	4	✗	✗	✗	0			
A	5	✗	✗	✗	✗			
T	6	✗	✗	✗	✗	✗		
C	7	✗	✗	✗	✗	✗	✗	

strand sequence =

ATTGATC

base cases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0						
T	2	✗	0					
T	3	✗	✗	0				
G	4	✗	✗	✗	0			
A	5	✗	✗	✗	✗	0		
T	6	✗	✗	✗	✗	✗		
C	7	✗	✗	✗	✗	✗	✗	

strand sequence =

ATTGATC

base cases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0						
T	2	✗	0					
T	3	✗	✗	0				
G	4	✗	✗	✗	0			
A	5	✗	✗	✗	✗	0		
T	6	✗	✗	✗	✗	✗	0	
C	7	✗	✗	✗	✗	✗	✗	

strand sequence =

ATTGATC

base cases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0						
T	2	✗	0					
T	3	✗	✗	0				
G	4	✗	✗	✗	0			
A	5	✗	✗	✗	✗	0		
T	6	✗	✗	✗	✗	✗	0	
C	7	✗	✗	✗	✗	✗	✗	0

strand sequence =

ATTGATC

base cases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0	●					
T	2	✗	0					
T	3	✗	✗	0				
G	4	✗	✗	✗	0			
A	5	✗	✗	✗	✗	0		
T	6	✗	✗	✗	✗	✗	0	
C	7	✗	✗	✗	✗	✗	✗	0

strand sequence =

ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0	1					
T	2	✗	0					
T	3	✗	✗	0				
G	4	✗	✗	✗	0			
A	5	✗	✗	✗	✗	0		
T	6	✗	✗	✗	✗	✗	0	
C	7	✗	✗	✗	✗	✗	✗	0

strand sequence =

ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0	1					
T	2	✗	0	●				
T	3	✗	✗	0				
G	4	✗	✗	✗	0			
A	5	✗	✗	✗	✗	0		
T	6	✗	✗	✗	✗	✗	0	
C	7	✗	✗	✗	✗	✗	✗	0

strand sequence =

A$\underline{\text{TT}}$GATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0	1					
T	2	✗	0	0				
T	3	✗	✗	0				
G	4	✗	✗	✗	0			
A	5	✗	✗	✗	✗	0		
T	6	✗	✗	✗	✗	✗	0	
C	7	✗	✗	✗	✗	✗	✗	0

strand sequence =

$$\mathrm{A\underline{TT}GATC}$$

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0	1					
T	2	✗	0	0				
T	3	✗	✗	0	0			
G	4	✗	✗	✗	0			
A	5	✗	✗	✗	✗	0		
T	6	✗	✗	✗	✗	✗	0	
C	7	✗	✗	✗	✗	✗	✗	0

strand sequence =

$$\text{AT}\underline{\text{TG}}\text{ATC}$$

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0	1					
T	2	✗	0	0				
T	3	✗	✗	0	0			
G	4	✗	✗	✗	0	0		
A	5	✗	✗	✗	✗	0		
T	6	✗	✗	✗	✗	✗	0	
C	7	✗	✗	✗	✗	✗	✗	0

strand sequence =

ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0	1					
T	2	✗	0	0				
T	3	✗	✗	0	0			
G	4	✗	✗	✗	0	0		
A	5	✗	✗	✗	✗	0	●	
T	6	✗	✗	✗	✗	✗	0	
C	7	✗	✗	✗	✗	✗	✗	0

strand sequence =

ATTGATC

base cases

recursive cases with complementary bases

recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	🟢 0	🔵 1					
T	2	✗	🟢 0	🟠 0				
T	3	✗	✗	🟢 0	🟠 0			
G	4	✗	✗	✗	🟢 0	🟠 0		
A	5	✗	✗	✗	✗	🟢 0	🔵 1	
T	6	✗	✗	✗	✗	✗	🟢 0	
C	7	✗	✗	✗	✗	✗	✗	🟢 0

strand sequence =

ATTG$\underline{\text{AT}}$C

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	🟩 0	🟦 1					
T	2	✗	🟩 0	🟧 0				
T	3	✗	✗	🟩 0	🟧 0			
G	4	✗	✗	✗	🟩 0	🟧 0		
A	5	✗	✗	✗	✗	🟩 0	🟦 1	
T	6	✗	✗	✗	✗	✗	🟩 0	🟧 0
C	7	✗	✗	✗	✗	✗	✗	🟩 0

strand sequence =

ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	🟩 0	🟦 1	🟦				
T	2	✗	🟩 0	🟧 0				
T	3	✗	✗	🟩 0	🟧 0			
G	4	✗	✗	✗	🟩 0	🟧 0		
A	5	✗	✗	✗	✗	🟩 0	🟦 1	
T	6	✗	✗	✗	✗	✗	🟩 0	🟧 0
C	7	✗	✗	✗	✗	✗	✗	🟩 0

strand sequence =

ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	🟢 0	🔵 1	🔵 1				
T	2	✗	🟢 0	🟠 0				
T	3	✗	✗	🟢 0	🟠 0			
G	4	✗	✗	✗	🟢 0	🟠 0		
A	5	✗	✗	✗	✗	🟢 0	🔵 1	
T	6	✗	✗	✗	✗	✗	🟢 0	🟠 0
C	7	✗	✗	✗	✗	✗	✗	🟢 0

strand sequence =

ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	🟢 0	🔵 1	🔵 1				
T	2	✗	🟢 0	🟠 0	🟠 0			
T	3	✗	✗	🟢 0	🟠 0			
G	4	✗	✗	✗	🟢 0	🟠 0		
A	5	✗	✗	✗	✗	🟢 0	🔵 1	
T	6	✗	✗	✗	✗	✗	🟢 0	🟠 0
C	7	✗	✗	✗	✗	✗	✗	🟢 0

strand sequence =

ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0	1	1				
T	2	✗	0	0	0			
T	3	✗	✗	0	0	●		
G	4	✗	✗	✗	0	0		
A	5	✗	✗	✗	✗	0	1	
T	6	✗	✗	✗	✗	✗	0	0
C	7	✗	✗	✗	✗	✗	✗	0

strand sequence =

ATTGATC

base cases

recursive cases with complementary bases

recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	🟢 0	🔵 1	🔵 1				
T	2	✗	🟢 0	🟠 0	🟠 0			
T	3	✗	✗	🟢 0	🟠 0	🔵 1		
G	4	✗	✗	✗	🟢 0	🟠 0		
A	5	✗	✗	✗	✗	🟢 0	🔵 1	
T	6	✗	✗	✗	✗	✗	🟢 0	🟠 0
C	7	✗	✗	✗	✗	✗	✗	🟢 0

strand sequence =

ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	🟢 0	🔵 1	🔵 1				
T	2	✗	🟢 0	🟠 0	🟠 0			
T	3	✗	✗	🟢 0	🟠 0	🔵 1		
G	4	✗	✗	✗	🟢 0	🟠 0	🔵	
A	5	✗	✗	✗	✗	🟢 0	🔵 1	
T	6	✗	✗	✗	✗	✗	🟢 0	🟠 0
C	7	✗	✗	✗	✗	✗	✗	🟢 0

strand sequence =

ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	🟢 0	🔵 1	🔵 1				
T	2	✗	🟢 0	🟠 0	🟠 0			
T	3	✗	✗	🟢 0	🟠 0	🔵 1		
G	4	✗	✗	✗	🟢 0	🟠 0	🔵 1	
A	5	✗	✗	✗	✗	🟢 0	🔵 1	
T	6	✗	✗	✗	✗	✗	🟢 0	🟠 0
C	7	✗	✗	✗	✗	✗	✗	🟢 0

strand sequence =

ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	🟩 0	🟦 1	🟦 1				
T	2	✗	🟩 0	🟧 0	🟧 0			
T	3	✗	✗	🟩 0	🟧 0	🟦 1		
G	4	✗	✗	✗	🟩 0	🟧 0	🟦 1	
A	5	✗	✗	✗	✗	🟩 0	🟦 1	🟧 0
T	6	✗	✗	✗	✗	✗	🟩 0	🟧 0
C	7	✗	✗	✗	✗	✗	✗	🟩 0

strand sequence =

ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm

		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	🟢 0	🔵 1	🔵 1	🟠 1	🔵 2	🔵 2	🔵
T	2	✗	🟢 0	🟠 0	🟠 0	🔵 1	🔵 1	🔵 2
T	3	✗	✗	🟢 0	🟠 0	🔵 1	🔵 1	🔵 2
G	4	✗	✗	✗	🟢 0	🟠 0	🔵 1	🔵 1
A	5	✗	✗	✗	✗	🟢 0	🔵 1	🟠 0
T	6	✗	✗	✗	✗	✗	🟢 0	🟠 0
C	7	✗	✗	✗	✗	✗	✗	🟢 0

strand sequence =

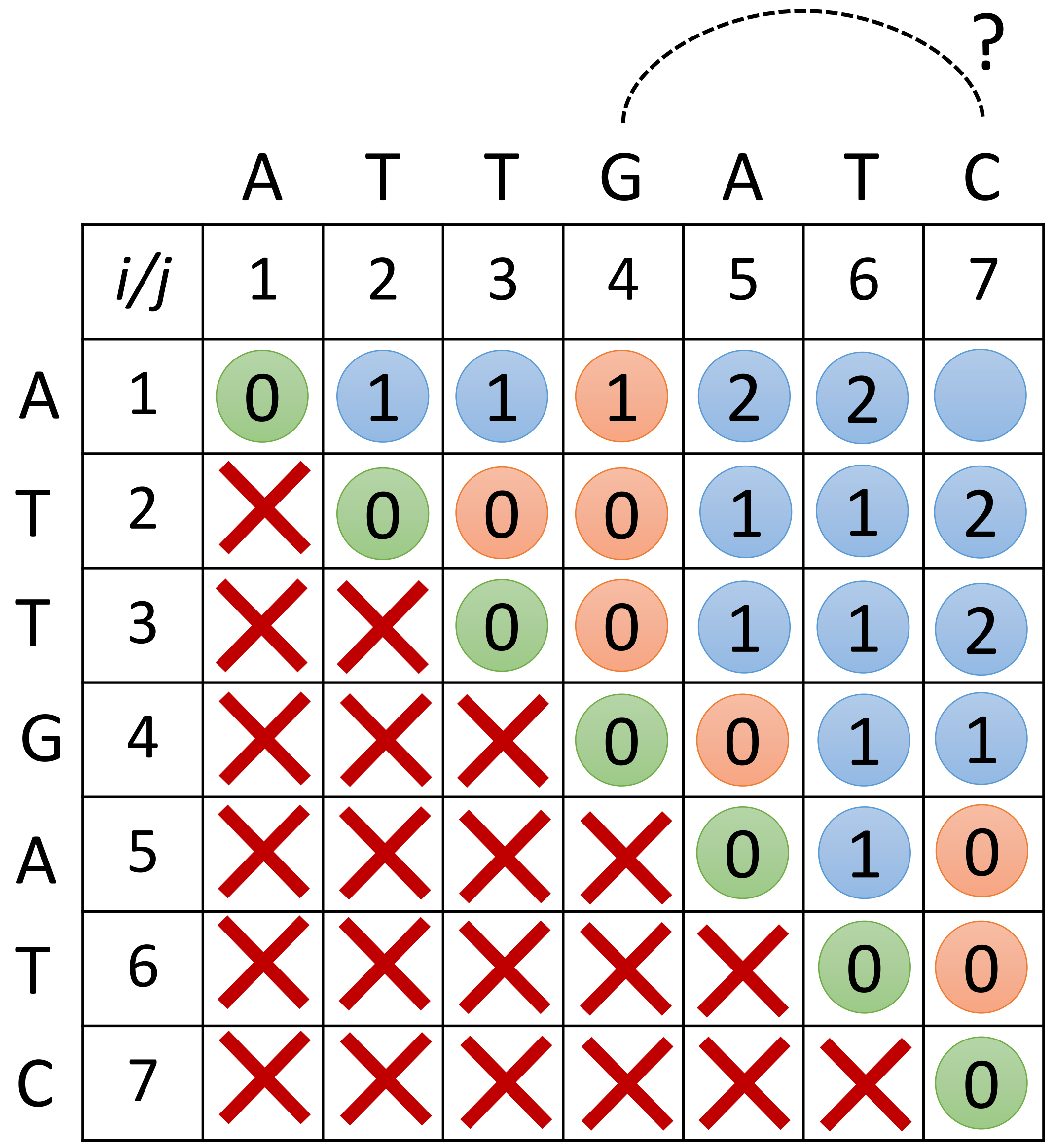
ATTGATC

🟢 base cases

🔵 recursive cases with complementary bases

🟠 recursive cases without complementary bases

Example of dynamic programming algorithm



		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	🟢 0	🔵 1	🔵 1	🟠 1	🔵 2	🔵 2	🔵
T	2	✗	🟢 0	🟠 0	🟠 0	🔵 1	🔵 1	🔵 2
T	3	✗	✗	🟢 0	🟠 0	🔵 1	🔵 1	🔵 2
G	4	✗	✗	✗	🟢 0	🟠 0	🔵 1	🔵 1
A	5	✗	✗	✗	✗	🟢 0	🔵 1	🟠 0
T	6	✗	✗	✗	✗	✗	🟢 0	🟠 0
C	7	✗	✗	✗	✗	✗	✗	🟢 0

strand sequence =

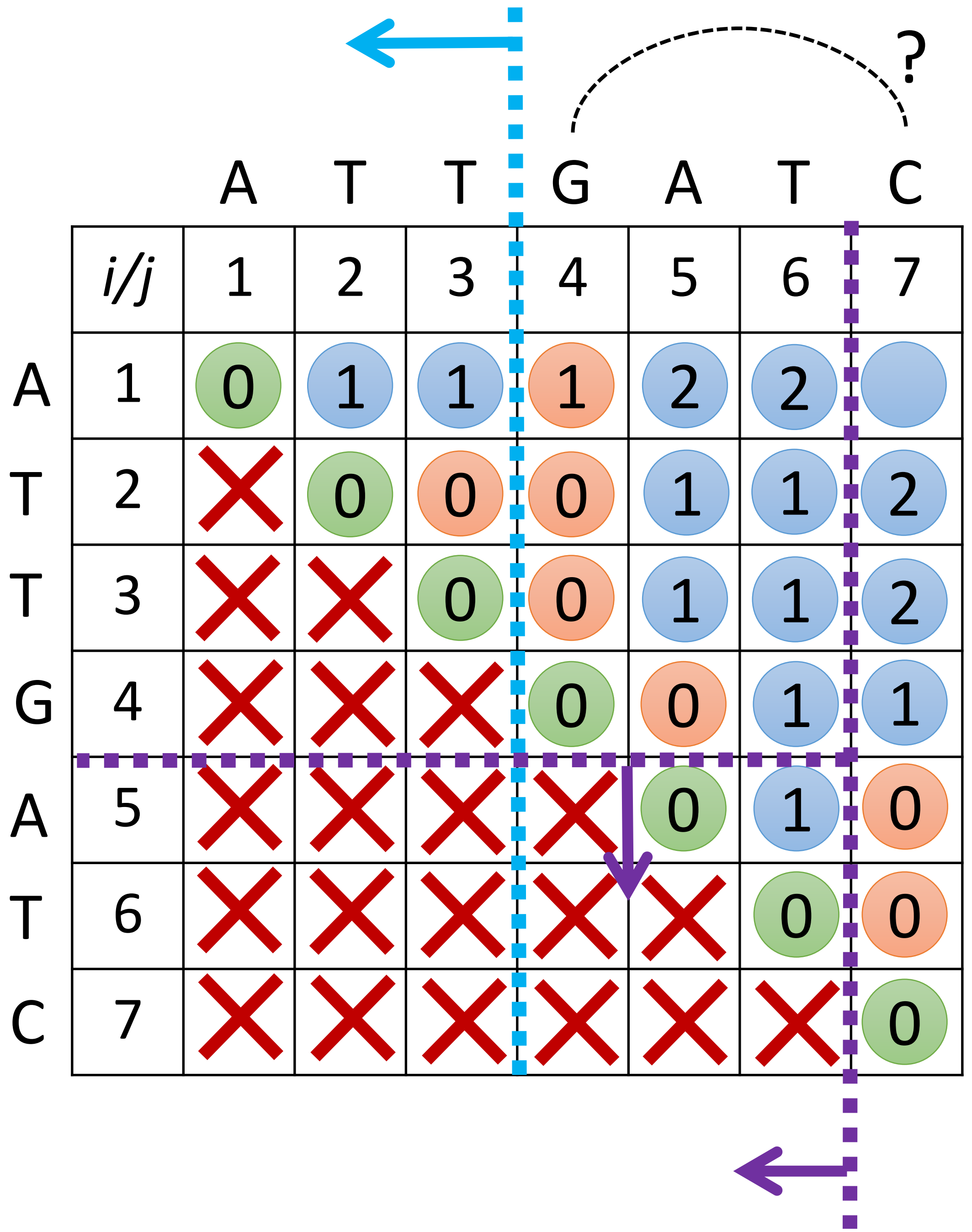
ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm



		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0	1	1	1	2	2	●
T	2	✗	0	0	0	1	1	2
T	3	✗	✗	0	0	1	1	2
G	4	✗	✗	✗	0	0	1	1
A	5	✗	✗	✗	✗	0	1	0
T	6	✗	✗	✗	✗	✗	0	0
C	7	✗	✗	✗	✗	✗	✗	0

strand sequence =

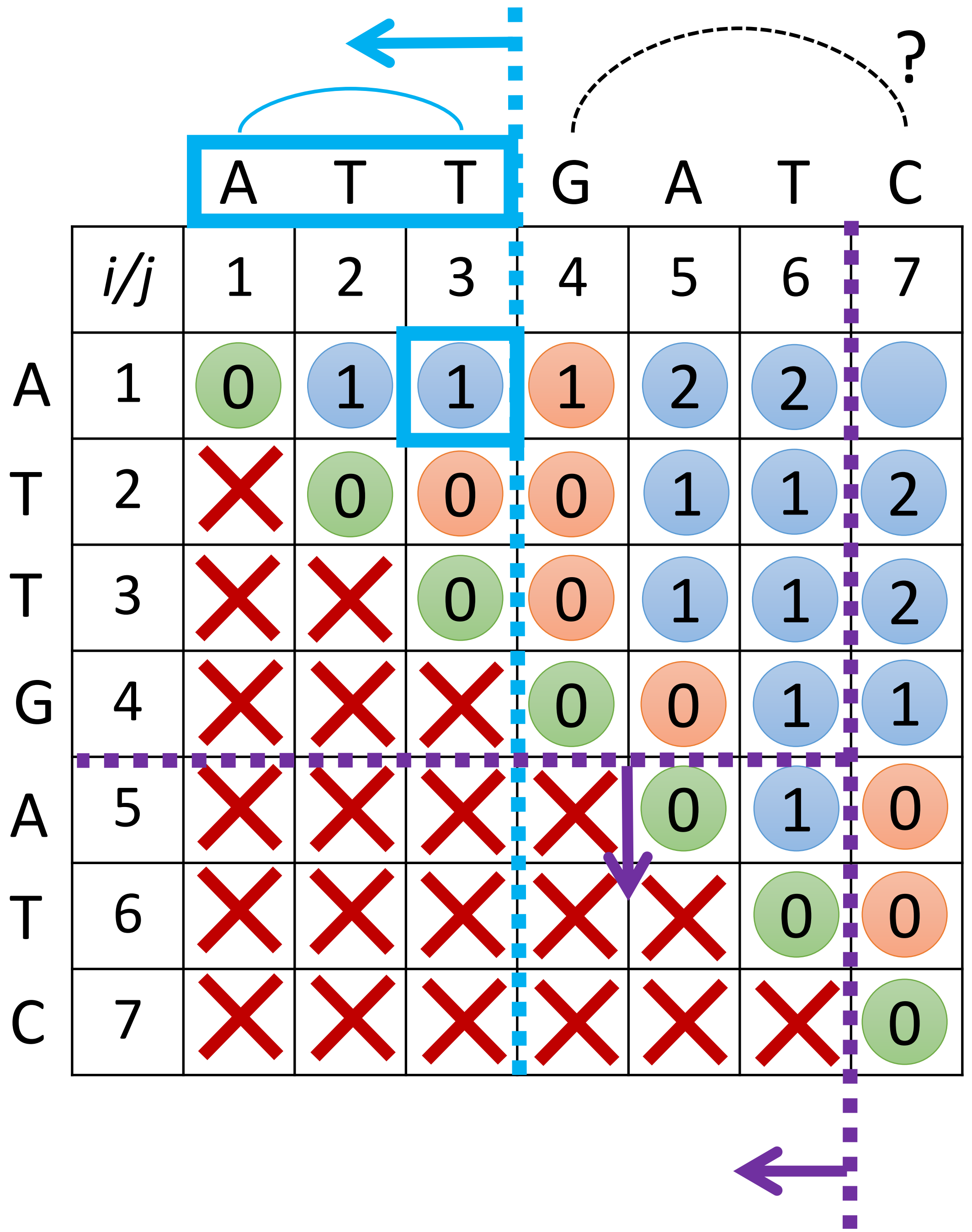
ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm



		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0	1	1	1	2	2	
T	2	✗	0	0	0	1	1	2
T	3	✗	✗	0	0	1	1	2
G	4	✗	✗	✗	0	0	1	1
A	5	✗	✗	✗	✗	0	1	0
T	6	✗	✗	✗	✗	✗	0	0
C	7	✗	✗	✗	✗	✗	✗	0

strand sequence =

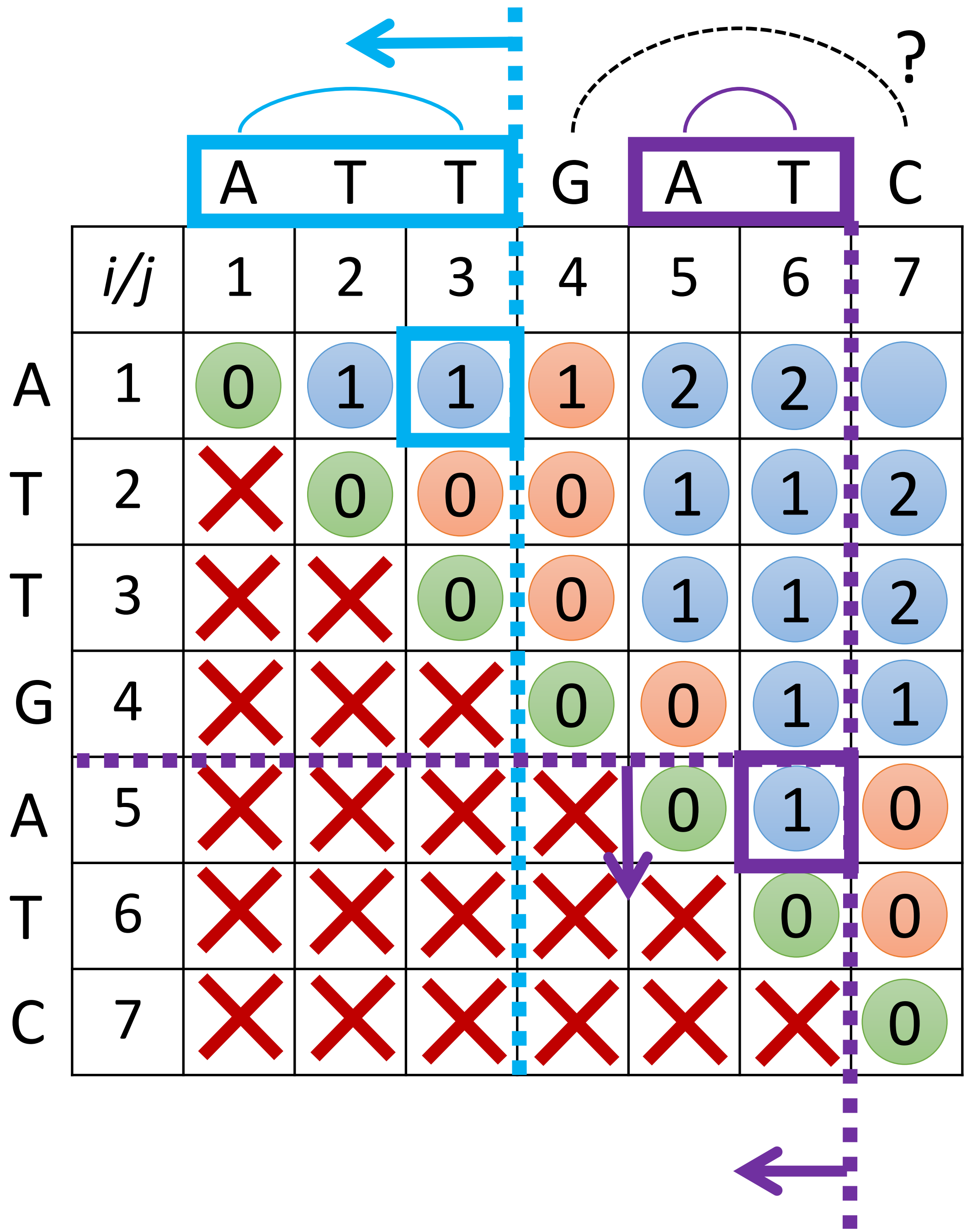
ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm



	i/j	1	2	3	4	5	6	7
A	1	0	1	1	1	2	2	
T	2	✗	0	0	0	1	1	2
T	3	✗	✗	0	0	1	1	2
G	4	✗	✗	✗	0	0	1	1
A	5	✗	✗	✗	✗	0	1	0
T	6	✗	✗	✗	✗	✗	0	0
C	7	✗	✗	✗	✗	✗	✗	0

strand sequence =

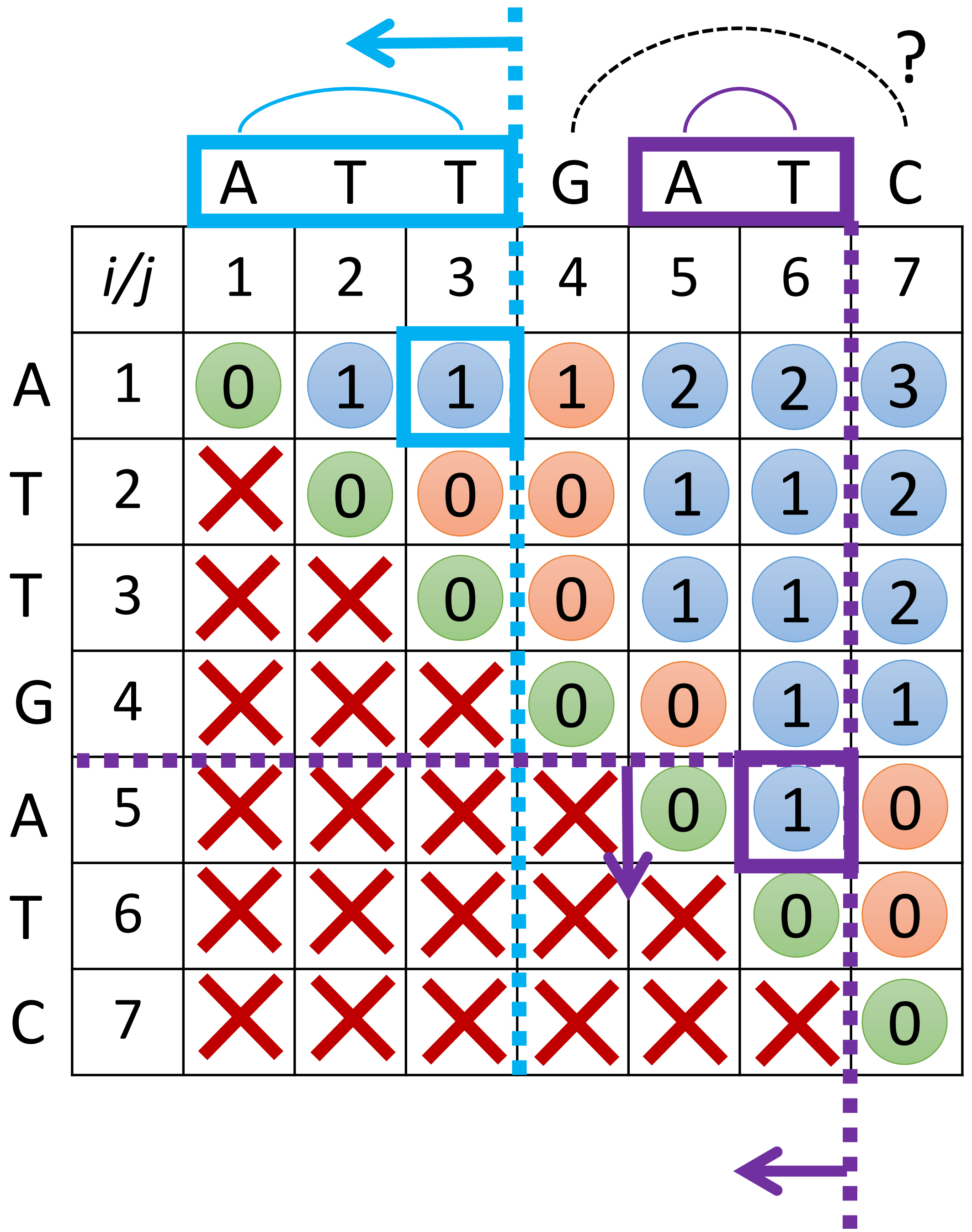
ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Example of dynamic programming algorithm



		A	T	T	G	A	T	C
	i/j	1	2	3	4	5	6	7
A	1	0	1	1	1	2	2	3
T	2	✗	0	0	0	1	1	2
T	3	✗	✗	0	0	1	1	2
G	4	✗	✗	✗	0	0	1	1
A	5	✗	✗	✗	✗	0	1	0
T	6	✗	✗	✗	✗	✗	0	0
C	7	✗	✗	✗	✗	✗	✗	0

strand sequence =

ATTGATC

🟩 base cases

🟦 recursive cases with complementary bases

🟧 recursive cases without complementary bases

Extensions to more realistic energy models

- base pairs on one strand must be separated by at least 4 other bases

Extensions to more realistic energy models

- base pairs on one strand must be separated by at least 4 other bases
    - base case switches from OPT(i,i) = 0 to OPT(i,j)=0 if $j–i \leq 4$

Extensions to more realistic energy models

- base pairs on one strand must be separated by at least 4 other bases
    - base case switches from OPT(i,i) = 0 to OPT(i,j)=0 if $j-i \leq 4$
- G/C twice as strong as A/T?

Extensions to more realistic energy models

- base pairs on one strand must be separated by at least 4 other bases
  - base case switches from OPT(i,i) = 0 to OPT(i,j)=0 if $j-i \leq 4$
- G/C twice as strong as A/T?
  - $\max_{i \leq k < j}$ (1 if k,j is A/T base pair, else 2) + OPT(i,$k-1$) + OPT($k+1$,$j-1$)

Extensions to more realistic energy models

- base pairs on one strand must be separated by at least 4 other bases
  - base case switches from OPT(i,i) = 0 to OPT(i,j)=0 if $j–i \le 4$
- G/C twice as strong as A/T?
  - $\max_{i \le k < j}$ (1 if k,j is A/T base pair, else 2) + OPT(i,k−1) + OPT(k+1,j−1)
- nearest-neighbor interaction?
  - $\max_{i \le k < j}$ (more complex lookup here) + OPT(i,k−1) + OPT(k+1,j−1)

Extensions to more realistic energy models

- base pairs on one strand must be separated by at least 4 other bases
    - base case switches from OPT(i,i) = 0 to OPT(i,j)=0 if $j – i \leq 4$
- G/C twice as strong as A/T?
    - $\max_{i \leq k < j}$ (1 if k,j is A/T base pair, else 2) + OPT(i,$k-1$) + OPT($k+1$,$j-1$)
- nearest-neighbor interaction?
    - $\max_{i \leq k < j}$ (more complex lookup here) + OPT(i,$k-1$) + OPT($k+1$,$j-1$)
- multiple strands?
    - a ΔG_{assoc} term for each strand beyond the first one

Extensions to more realistic energy models

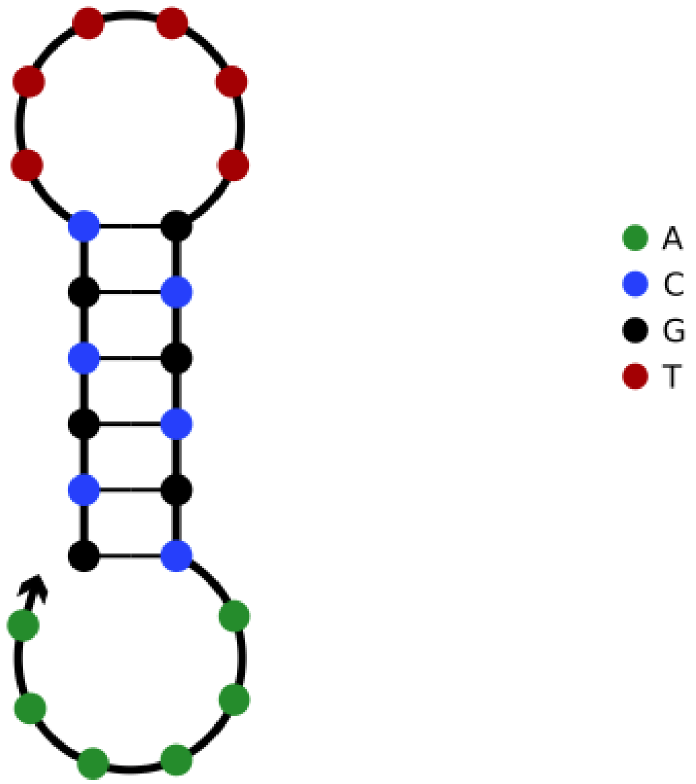
- base pairs on one strand must be separated by at least 4 other bases
  - base case switches from OPT(i,i) = 0 to OPT(i,j)=0 if $j–i \le 4$
- G/C twice as strong as A/T?
  - $\max_{i \le k < j}$ (1 if k,j is A/T base pair, else 2) + OPT(i,$k-1$) + OPT($k+1$,$j-1$)
- nearest-neighbor interaction?
  - $\max_{i \le k < j}$ (more complex lookup here) + OPT(i,$k-1$) + OPT($k+1$,$j-1$)
- multiple strands?
  - a ΔG_{assoc} term for each strand beyond the first one
- https://piercelab-caltech.github.io/nupack-docs/definitions/

Software to compute minimum free energy DNA structures

NUPACK
http://www.nupack.org/

ViennaRNA
https://www.tbi.univie.ac.at/RNA/

MFE structure at 37.0 C



Free energy of secondary structure: -8.78 kcal/mol

What is “free energy”?

A way to express probability of seeing a structure, in units of energy (kcal/mol).

Energy and probability are *exponentially* related.

What is "free energy"?

A way to express probability of seeing a structure, in units of energy (kcal/mol).

Energy and probability are *exponentially* related.

- If *S* is a secondary structure, let Pr[*S*] denote probability of seeing it ("*at equilibrium*").

What is “free energy”?

A way to express probability of seeing a structure, in units of energy (kcal/mol).

Energy and probability are *exponentially* related.

- If S is a secondary structure, let $\Pr[S]$ denote probability of seeing it (“*at equilibrium*”).
- At fixed temperature, $\ln(\Pr[S]) \approx \Delta G(S)$ (*recall free energy $\Delta G(S)$ is negative*)

What is “free energy”?

A way to express probability of seeing a structure, in units of energy (kcal/mol).

Energy and probability are *exponentially* related.

- If S is a secondary structure, let $\Pr[S]$ denote probability of seeing it (“*at equilibrium*”).
- At fixed temperature, $\ln(\Pr[S]) \approx \Delta G(S)$ (*recall free energy ΔG(S) is negative*)
- Some constants: $\ln(\Pr[S]) \approx \Delta G(S)/(RT)$, usually expressed as $\Pr[S] \propto e^{-\Delta G(S)/(RT)}$
  T = temperature in K (Kelvin), R = Boltzmann's constant ≈ 0.001987204 kcal/mol/K

What is “free energy”?

A way to express probability of seeing a structure, in units of energy (kcal/mol).

Energy and probability are *exponentially* related.

- If S is a secondary structure, let $\Pr[S]$ denote probability of seeing it (“*at equilibrium*”).
- At fixed temperature, $\ln(\Pr[S]) \approx \Delta G(S)$ (*recall free energy ΔG(S) is negative*)
- Some constants: $\ln(\Pr[S]) \approx \Delta G(S)/(RT)$, usually expressed as $\Pr[S] \propto e^{-\Delta G(S)/(RT)}$

  T = temperature in K (Kelvin), R = Boltzmann's constant ≈ 0.001987204 kcal/mol/K
- To convert $e^{-\Delta G(S)/(RT)}$ to a probability, need to normalize so they sum to 1.

What is "free energy"?

A way to express probability of seeing a structure, in units of energy (kcal/mol).

Energy and probability are *exponentially* related.

- If S is a secondary structure, let $\Pr[S]$ denote probability of seeing it ("*at equilibrium*").
- At fixed temperature, $\ln(\Pr[S]) \approx \Delta G(S)$ (*recall free energy ΔG(S) is negative*)
- Some constants: $\ln(\Pr[S]) \approx \Delta G(S)/(RT)$, usually expressed as $\Pr[S] \propto e^{-\Delta G(S)/(RT)}$

  T = temperature in K (Kelvin), R = Boltzmann's constant ≈ 0.001987204 kcal/mol/K
- To convert $e^{-\Delta G(S)/(RT)}$ to a probability, need to normalize so they sum to 1.
- For a DNA strand/set of DNA strands, let Ω denote set of all secondary structures.

What is “free energy”?

A way to express probability of seeing a structure, in units of energy (kcal/mol).

Energy and probability are *exponentially* related.

- If S is a secondary structure, let $\Pr[S]$ denote probability of seeing it (“*at equilibrium*”).
- At fixed temperature, $\ln(\Pr[S]) \approx \Delta G(S)$ (*recall free energy ΔG(S) is negative*)
- Some constants: $\ln(\Pr[S]) \approx \Delta G(S)/(RT)$, usually expressed as $\Pr[S] \propto e^{-\Delta G(S)/(RT)}$
  T = temperature in K (Kelvin), R = Boltzmann's constant ≈ 0.001987204 kcal/mol/K
- To convert $e^{-\Delta G(S)/(RT)}$ to a probability, need to normalize so they sum to 1.
- For a DNA strand/set of DNA strands, let Ω denote set of all secondary structures.

Definition: The partition function of Ω is $Q = \Sigma_{S\in\Omega}\, e^{-\Delta G(S)/(RT)}$.

What is "free energy"?

A way to express probability of seeing a structure, in units of energy (kcal/mol).

Energy and probability are *exponentially* related.

- If S is a secondary structure, let $\Pr[S]$ denote probability of seeing it ("*at equilibrium*").
- At fixed temperature, $\ln(\Pr[S]) \approx \Delta G(S)$ (*recall free energy ΔG(S) is negative*)
- Some constants: $\ln(\Pr[S]) \approx \Delta G(S)/(RT)$, usually expressed as $\Pr[S] \propto e^{-\Delta G(S)/(RT)}$
  T = temperature in K (Kelvin), R = Boltzmann's constant ≈ 0.001987204 kcal/mol/K
- To convert $e^{-\Delta G(S)/(RT)}$ to a probability, need to normalize so they sum to 1.
- For a DNA strand/set of DNA strands, let Ω denote set of all secondary structures.

Definition: The partition function of Ω is $Q = \sum_{S \in \Omega} e^{-\Delta G(S)/(RT)}$.

For any secondary structure S, $\Pr[S] = (1/Q) \cdot e^{-\Delta G(S)/(RT)}$.

Minimum free energy versus complex free energy

Recall: For any secondary structure *S*,
$\Pr[S] = (1/Q) \cdot e^{-\Delta G(S)/(RT)}$

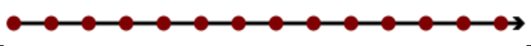
Minimum free energy structure *S* is the most likely structure.

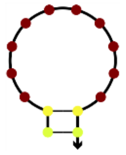
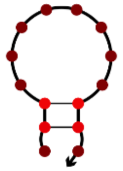
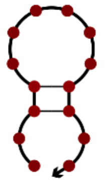
Minimum free energy versus complex free energy

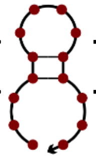
Recall: For any secondary structure *S*,
Pr[*S*] = (1/*Q*) · $e^{-\Delta G(S)/(RT)}$

Minimum free energy structure *S* is the most likely structure.

Problem: What if *most likely* structure *S* is *not very likely*?

Pr[] = **0.2**, but

Pr[] = Pr[] = Pr[]

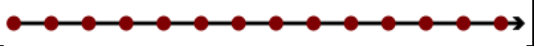
= Pr[] = **0.199**

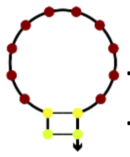
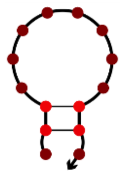
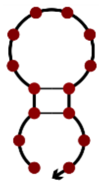
Minimum free energy versus complex free energy

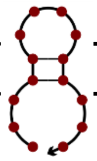
Recall: For any secondary structure *S*, $\Pr[S] = (1/Q) \cdot e^{-\Delta G(S)/(RT)}$

Minimum free energy structure *S* is the most likely structure.

Problem: What if *most likely* structure *S* is *not very likely*?

Pr[] = **0.2**, but

Pr[] = Pr[] = Pr[]

= Pr[] = **0.199**

This strand spends nearly 80% of its time bound.

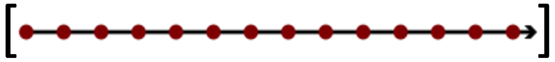
Minimum free energy versus complex free energy

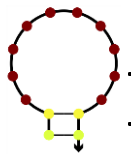
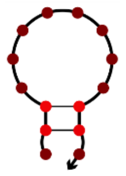
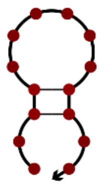
Recall: For any secondary structure *S*, $\Pr[S] = (1/Q) \cdot e^{-\Delta G(S)/(RT)}$

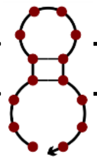
Minimum free energy structure *S* is the most likely structure.

Problem: What if *most likely* structure *S* is *not very likely*?

Solution: Consider energy of *all secondary structures at once*.

Pr[] = **0.2**, but

Pr[] = Pr[] = Pr[]

= Pr[] = **0.199**

This strand spends nearly 80% of its time bound.

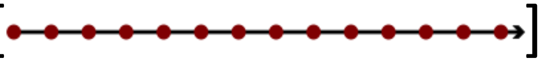
Minimum free energy versus complex free energy

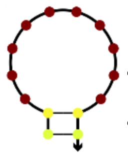
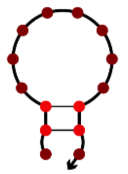
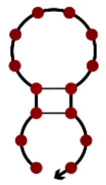
Recall: For any secondary structure *S*,
Pr[*S*] = (1/*Q*) · $e^{-\Delta G(S)/(RT)}$

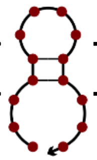
Minimum free energy structure *S* is the most likely structure.

Problem: What if *most likely* structure *S* is *not very likely*?

Solution: Consider energy of *all secondary structures at once*.

Pr[] = **0.2**, but

Pr[] = Pr[] = Pr[]

= Pr[] = **0.199**

This strand spends nearly 80% of its time bound.

Definition: The complex free energy of Ω is
$\Delta G = -RT \ln Q$.
Intuitively captures how much we expect strand to be bound/structured: higher (closer to 0) means more unstructured.

https://piercelab-caltech.github.io/nupack-docs/definitions/#complex-free-energy

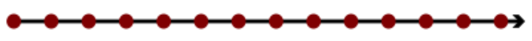
Minimum free energy versus complex free energy

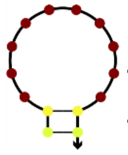
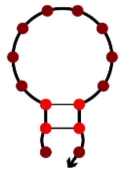
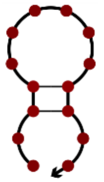
Recall: For any secondary structure S, $\Pr[S] = (1/Q) \cdot e^{-\Delta G(S)/(RT)}$

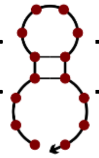
Minimum free energy structure S is the most likely structure.

Problem: What if *most likely* structure S is *not very likely*?

Solution: Consider energy of *all secondary structures at once*.

Pr[] = **0.2**, but

Pr[] = Pr[] = Pr[]

= Pr[] = **0.199**

This strand spends nearly 80% of its time bound.

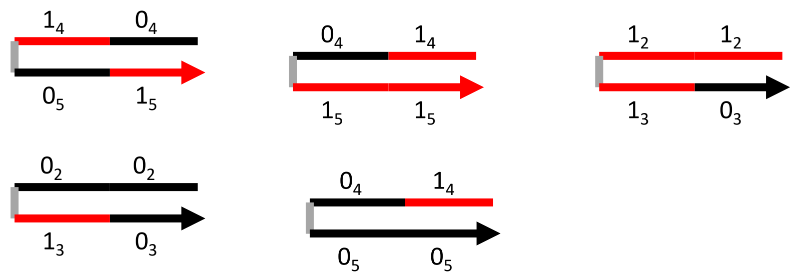
Definition: The complex free energy of Ω is $\Delta G = -RT \ln Q$.
Intuitively captures how much we expect strand to be bound/structured: higher (closer to 0) means more unstructured.

https://piercelab-caltech.github.io/nupack-docs/definitions/#complex-free-energy

ΔG can also be computed in time $O(n^3)$.

Example: DNA sequence design for single-stranded tiles

Given many single-stranded tiles with four domains each (lengths 10 and 11), assign DNA sequences to them satisfying:

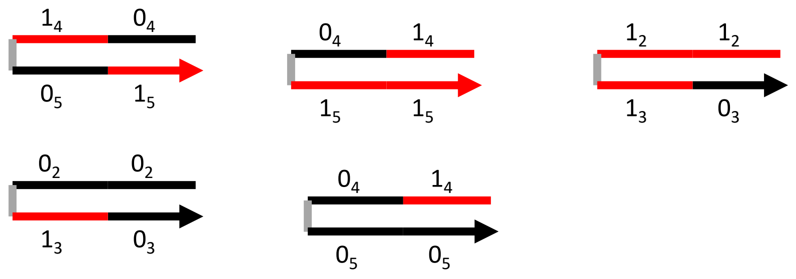


Abbreviated list of constraints similar to those used in [*Diverse and robust molecular algorithms using reprogrammable DNA self-assembly*. Woods, Doty, Myhrvold, Hui, Zhou, Yin, Winfree. Nature 2019.]

Example: DNA sequence design for single-stranded tiles

Given many single-stranded tiles with four domains each (lengths 10 and 11), assign DNA sequences to them satisfying:

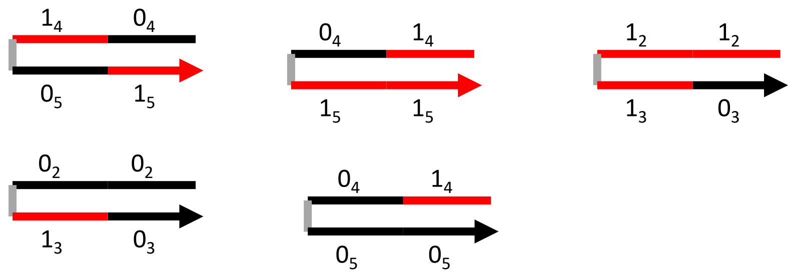
- ∀strands s, $\Delta G(s) \geq -1.65$ kcal/mol



Abbreviated list of constraints similar to those used in [*Diverse and robust molecular algorithms using reprogrammable DNA self-assembly*. Woods, Doty, Myhrvold, Hui, Zhou, Yin, Winfree. Nature 2019.]

Example: DNA sequence design for single-stranded tiles

Given many single-stranded tiles with four domains each (lengths 10 and 11), assign DNA sequences to them satisfying:

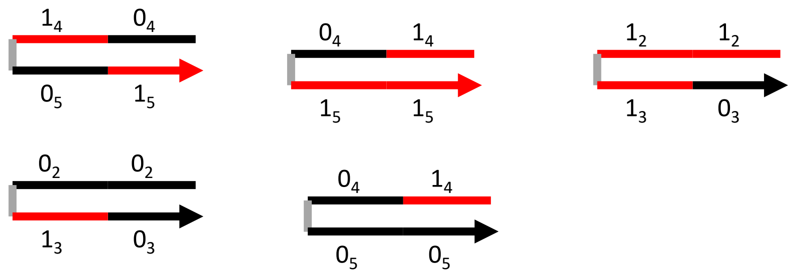


- ∀strands s, $\Delta G(s) \geq -1.65$ kcal/mol
- ∀strand pairs s,t, $\Delta G(s,t) \geq -5.4$ kcal/mol if no complementary domains, ≥ -7.4 kcal/mol otherwise

Abbreviated list of constraints similar to those used in [*Diverse and robust molecular algorithms using reprogrammable DNA self-assembly*. Woods, Doty, Myhrvold, Hui, Zhou, Yin, Winfree. Nature 2019.]

Example: DNA sequence design for single-stranded tiles

Given many single-stranded tiles with four domains each (lengths 10 and 11), assign DNA sequences to them satisfying:

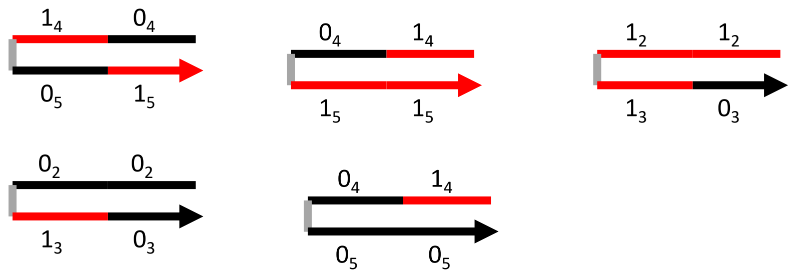


- ∀strands s, $\Delta G(s) \geq -1.65$ kcal/mol
- ∀strand pairs s,t, $\Delta G(s,t) \geq -5.4$ kcal/mol if no complementary domains, ≥ -7.4 kcal/mol otherwise
- all domains end with A or T

Abbreviated list of constraints similar to those used in [*Diverse and robust molecular algorithms using reprogrammable DNA self-assembly*. Woods, Doty, Myhrvold, Hui, Zhou, Yin, Winfree. Nature 2019.]

Example: DNA sequence design for single-stranded tiles

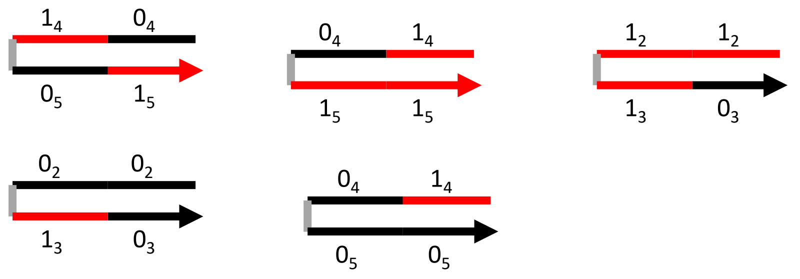
Given many single-stranded tiles with four domains each (lengths 10 and 11), assign DNA sequences to them satisfying:



- ∀strands s, $\Delta G(s) \geq -1.65$ kcal/mol
- ∀strand pairs s,t, $\Delta G(s,t) \geq -5.4$ kcal/mol if no complementary domains, ≥ -7.4 kcal/mol otherwise
- all domains end with A or T
- all domains have nearest-neighbor duplex energy between –9.2 and –8.9 kcal/mol

Abbreviated list of constraints similar to those used in [*Diverse and robust molecular algorithms using reprogrammable DNA self-assembly*. Woods, Doty, Myhrvold, Hui, Zhou, Yin, Winfree. Nature 2019.]

Example: DNA sequence design for single-stranded tiles



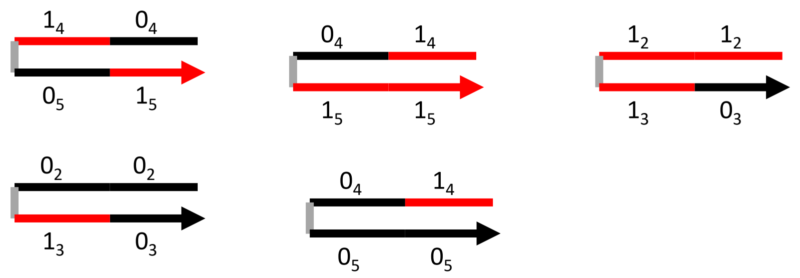
Given many single-stranded tiles with four domains each (lengths 10 and 11), assign DNA sequences to them satisfying:

- ∀strands s, $\Delta G(s) \geq -1.65$ kcal/mol
- ∀strand pairs s,t, $\Delta G(s,t) \geq -5.4$ kcal/mol if no complementary domains, ≥ -7.4 kcal/mol otherwise
- all domains end with A or T
- all domains have nearest-neighbor duplex energy between –9.2 and –8.9 kcal/mol
- tiles with even subscript domains on top have at most one G per domain (helps to satisfy first constraint)

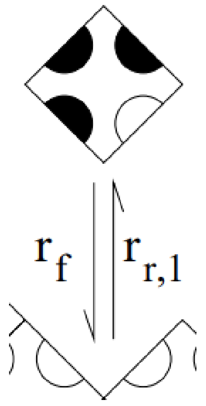
Abbreviated list of constraints similar to those used in [*Diverse and robust molecular algorithms using reprogrammable DNA self-assembly*. Woods, Doty, Myhrvold, Hui, Zhou, Yin, Winfree. Nature 2019.]

Example: DNA sequence design for single-stranded tiles

Given many single-stranded tiles with four domains each (lengths 10 and 11), assign DNA sequences to them satisfying:



- ∀strands s, $\Delta G(s) \geq -1.65$ kcal/mol
- ∀strand pairs s,t, $\Delta G(s,t) \geq -5.4$ kcal/mol if no complementary domains, ≥ -7.4 kcal/mol otherwise
- all domains end with A or T
- all domains have nearest-neighbor duplex energy between –9.2 and –8.9 kcal/mol
- tiles with even subscript domains on top have at most one G per domain (helps to satisfy first constraint)
- pairs of domains d_1,d_2 that could result in one-domain mismatches during tile binding have $\Delta G(d_1,d_2) \geq -1.6$ kcal/mol



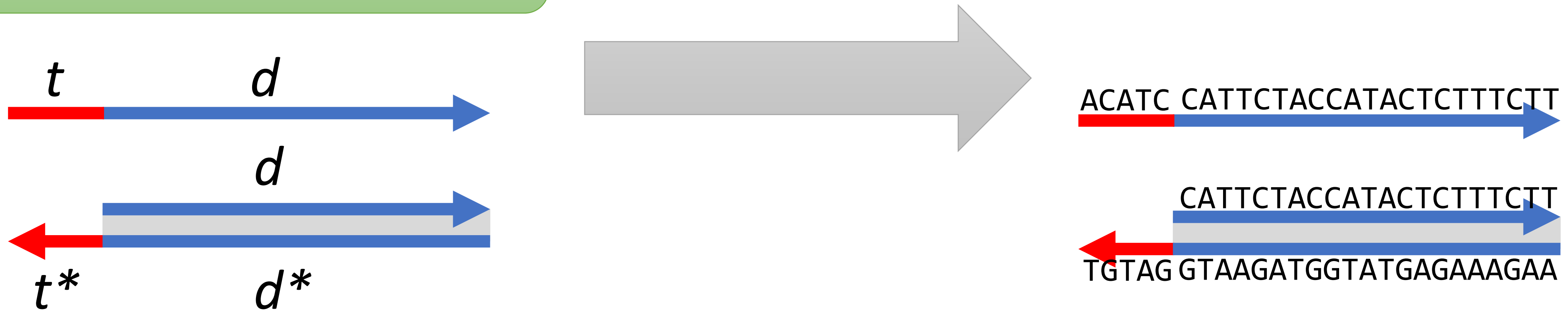
Abbreviated list of constraints similar to those used in [*Diverse and robust molecular algorithms using reprogrammable DNA self-assembly*. Woods, Doty, Myhrvold, Hui, Zhou, Yin, Winfree. Nature 2019.]

DNA sequence design

- If we have DNA sequences, we can compute MFE/complex free energies of individual strands, pairs of strands, etc. in polynomial time.
- DNA sequence design problem: given abstract strands with abstract domains, assign concrete DNA sequences to the domains to satisfy a list of (experiment-specific) constraints.
- This is almost certainly **NP**-hard for any "reasonable" choice of constraints.

DNA *strands* with abstract *domains*

DNA *sequences*



Stochastic local search for finding DNA sequences

https://github.com/UC-Davis-molecular-computing/nuad

Stochastic local search for finding DNA sequences

1. Assign DNA sequences randomly to domains.

https://github.com/UC-Davis-molecular-computing/nuad

Stochastic local search for finding DNA sequences

1. Assign DNA sequences randomly to domains.
    - Each domain has a fixed length.

https://github.com/UC-Davis-molecular-computing/nuad

Stochastic local search for finding DNA sequences

1. Assign DNA sequences randomly to domains.
   - Each domain has a fixed length.
   - Implicitly assign complement sequence to complement domains.

https://github.com/UC-Davis-molecular-computing/nuad

Stochastic local search for finding DNA sequences

1. Assign DNA sequences randomly to domains.
    - Each domain has a fixed length.
    - Implicitly assign complement sequence to complement domains.
    - “Easy” single-domain constraints such as [*no GGGG*] or [*domains have A or T at each end*] can be automatically satisfied at this step.

https://github.com/UC-Davis-molecular-computing/nuad

Stochastic local search for finding DNA sequences

1. Assign DNA sequences randomly to domains.
    - Each domain has a fixed length.
    - Implicitly assign complement sequence to complement domains.
    - “Easy” single-domain constraints such as [*no GGGG*] or [*domains have A or T at each end*] can be automatically satisfied at this step.
2. Check list of all constraints, tallying violations and “blaming” appropriate domains.

https://github.com/UC-Davis-molecular-computing/nuad

Stochastic local search for finding DNA sequences

1. Assign DNA sequences randomly to domains.
   - Each domain has a fixed length.
   - Implicitly assign complement sequence to complement domains.
   - “Easy” single-domain constraints such as [*no GGGG*] or [*domains have A or T at each end*] can be automatically satisfied at this step.
2. Check list of all constraints, tallying violations and “blaming” appropriate domains.
   - For example, if a strand *s* has too low $\Delta G(s)$, all domains on strand are blamed.

https://github.com/UC-Davis-molecular-computing/nuad

Stochastic local search for finding DNA sequences

1. Assign DNA sequences randomly to domains.
   - Each domain has a fixed length.
   - Implicitly assign complement sequence to complement domains.
   - “Easy” single-domain constraints such as [*no GGGG*] or [*domains have A or T at each end*] can be automatically satisfied at this step.
2. Check list of all constraints, tallying violations and “blaming” appropriate domains.
   - For example, if a strand *s* has too low $\Delta G(s)$, all domains on strand are blamed.
3. If no constraints violated, we’re done!

https://github.com/UC-Davis-molecular-computing/nuad

Stochastic local search for finding DNA sequences

1. Assign DNA sequences randomly to domains.
   - Each domain has a fixed length.
   - Implicitly assign complement sequence to complement domains.
   - "Easy" single-domain constraints such as [*no GGGG*] or [*domains have A or T at each end*] can be automatically satisfied at this step.
2. Check list of all constraints, tallying violations and "blaming" appropriate domains.
   - For example, if a strand *s* has too low $\Delta G(s)$, all domains on strand are blamed.
3. If no constraints violated, we're done!
4. Otherwise, pick a domain *d* at random in proportion to total "score" of violations it caused.

https://github.com/UC-Davis-molecular-computing/nuad

Stochastic local search for finding DNA sequences

1. Assign DNA sequences randomly to domains.
   - Each domain has a fixed length.
   - Implicitly assign complement sequence to complement domains.
   - "Easy" single-domain constraints such as [*no GGGG*] or [*domains have A or T at each end*] can be automatically satisfied at this step.
2. Check list of all constraints, tallying violations and "blaming" appropriate domains.
   - For example, if a strand *s* has too low $\Delta G(s)$, all domains on strand are blamed.
3. If no constraints violated, we're done!
4. Otherwise, pick a domain *d* at random in proportion to total "score" of violations it caused.
5. Assign new random DNA sequence to *d*.

https://github.com/UC-Davis-molecular-computing/nuad

Stochastic local search for finding DNA sequences

1. Assign DNA sequences randomly to domains.
   - Each domain has a fixed length.
   - Implicitly assign complement sequence to complement domains.
   - "Easy" single-domain constraints such as [*no GGGG*] or [*domains have A or T at each end*] can be automatically satisfied at this step.
2. Check list of all constraints, tallying violations and "blaming" appropriate domains.
   - For example, if a strand *s* has too low $\Delta G(s)$, all domains on strand are blamed.
3. If no constraints violated, we're done!
4. Otherwise, pick a domain *d* at random in proportion to total "score" of violations it caused.
5. Assign new random DNA sequence to *d*.
   - This change propagates through to all instances of *d* and *d** on all strands.

https://github.com/UC-Davis-molecular-computing/nuad

Stochastic local search for finding DNA sequences

1. Assign DNA sequences randomly to domains.
   - Each domain has a fixed length.
   - Implicitly assign complement sequence to complement domains.
   - “Easy” single-domain constraints such as [*no GGGG*] or [*domains have A or T at each end*] can be automatically satisfied at this step.
2. Check list of all constraints, tallying violations and “blaming” appropriate domains.
   - For example, if a strand *s* has too low Δ*G*(*s*), all domains on strand are blamed.
3. If no constraints violated, we’re done!
4. Otherwise, pick a domain *d* at random in proportion to total “score” of violations it caused.
5. Assign new random DNA sequence to *d*.
   - This change propagates through to all instances of *d* and *d** on all strands.
6. Repeat step 2; if the new DNA sequence for *d* results in lower score of violations, keep it, otherwise, ignore it and pick a new random domain at step 4.

https://github.com/UC-Davis-molecular-computing/nuad

Stochastic local search for finding DNA sequences

1. Assign DNA sequences randomly to domains.
   - Each domain has a fixed length.
   - Implicitly assign complement sequence to complement domains.
   - “Easy” single-domain constraints such as [*no GGGG*] or [*domains have A or T at each end*] can be automatically satisfied at this step.
2. Check list of all constraints, tallying violations and “blaming” appropriate domains.
   - For example, if a strand *s* has too low $\Delta G(s)$, all domains on strand are blamed.
3. If no constraints violated, we’re done!
4. Otherwise, pick a domain *d* at random in proportion to total “score” of violations it caused.
5. Assign new random DNA sequence to *d*.
   - This change propagates through to all instances of *d* and *d** on all strands.
6. Repeat step 2; if the new DNA sequence for *d* results in lower score of violations, keep it, otherwise, ignore it and pick a new random domain at step 4.
7. Repeat until no constraints are violated.

https://github.com/UC-Davis-molecular-computing/nuad

Stochastic local search for finding DNA sequences

1. Assign DNA sequences randomly to domains.
    - Each domain has a fixed length.
    - Implicitly assign complement sequence to complement domains.
    - “Easy” single-domain constraints such as [*no GGGG*] or [*domains have A or T at each end*] can be automatically satisfied at this step.
2. Check list of all constraints, tallying violations and “blaming” appropriate domains.
    - For example, if a strand *s* has too low Δ*G*(*s*), all domains on strand are blamed.
3. If no constraints violated, we’re done!
4. Otherwise, pick a domain *d* at random in proportion to total “score” of violations it caused.
5. Assign new random DNA sequence to *d*.
    - This change propagates through to all instances of *d* and *d** on all strands.
6. Repeat step 2; if the new DNA sequence for *d* results in lower score of violations, keep it, otherwise, ignore it and pick a new random domain at step 4.
7. Repeat until no constraints are violated.

https://github.com/UC-Davis-molecular-computing/nuad

Slow and unclever, but it works for any set of constraints.