

# Idury-Waterman Algorithm (1995)

CS1820/2820: Algorithmic Foundations of Computational Biology  
Spring 2022

Prof. Sorin Istrail

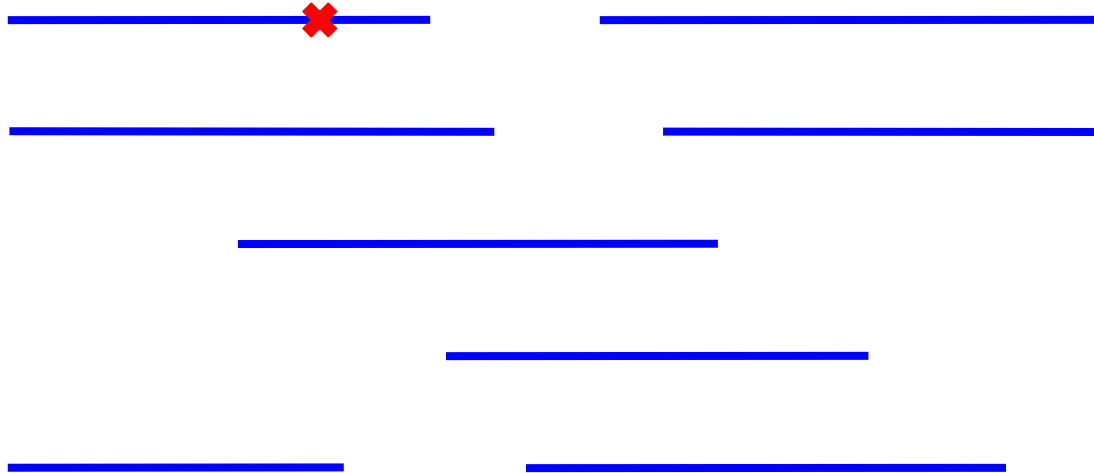
# The Algorithm

Inputs: Reads (fragments)  $f_1, f_2, \dots, f_N$  and length  $k$

1. Obtain the union of the **spectrum** of all reads
2. Construct the **sequence graph** with  $(k-1)$ -tuples as nodes from spectra in (1)
3. Perform a variant of **Eulerian tour** to infer sequence(s)
4. Align the reads to the inferred sequence(s) in (3)

An example:

TTCATGGACATCGAC



An example:

TTCATGGACATCGAC

TTCAGG CATCGAC

TTCATGG TCGAC

ATGGACA

GACATC

TTCAT ACATCGA

An example:

$f_1$  TTCAGG

$f_2$  TTCATGG

$f_3$  ATGGACA

$f_4$  TTCAT

$f_5$  CATCGAC

$f_6$  TCGAC

$f_7$  GACATC

$f_8$  ACATCGA

An example:

$f_1$	TTCAGG
$f_2$	TTCATGG
$f_3$	ATGGACA
$f_4$	TTCAT
$f_5$	CATCGAC
$f_6$	TCGAC
$f_7$	GACATC
$f_8$	ACATCGA









# 1) Obtain the union of spectra for all reads

$f_1$	TTCAGG
$f_2$	TTCATGG
$f_3$	ATGGACA
$f_4$	TTCAT
$f_5$	CATCGAC
$f_6$	TCGAC
$f_7$	GACATC
$f_8$	ACATCGA

First, we identify the set of all k-mers (substrings of length k) present in the data.

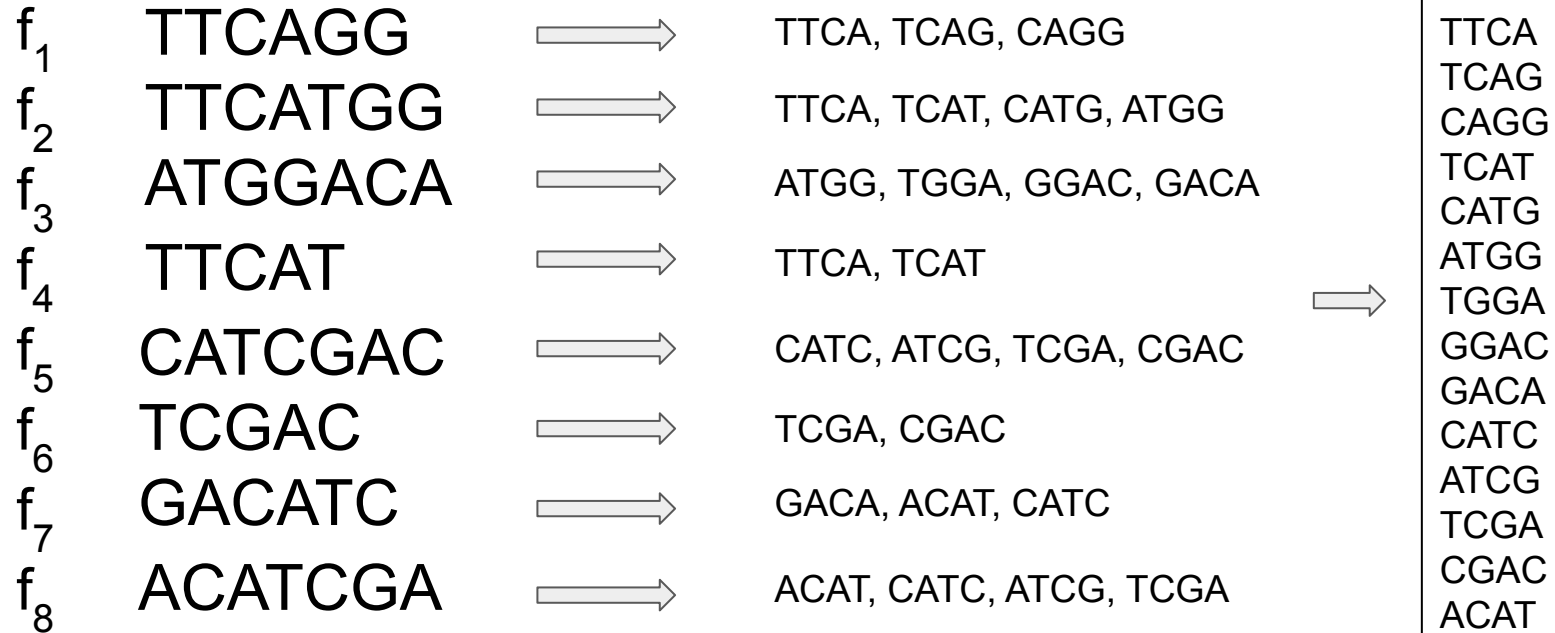
Let's choose  $k = 4$ .

# 1) Obtain the union of spectra for all reads

$f_1$	TTCAGG		TTCA, TCAG, CAGG
$f_2$	TTCATGG		TTCA, TCAT, CATG, ATGG
$f_3$	ATGGACA		ATGG, TGGA, GGAC, GACA
$f_4$	TTCAT		TTCA, TCAT
$f_5$	CATCGAC		CATC, ATCG, TCGA, CGAC
$f_6$	TCGAC		TCGA, CGAC
$f_7$	GACATC		GACA, ACAT, CATC
$f_8$	ACATCGA		ACAT, CATC, ATCG, TCGA



# 1) Obtain the union of spectra for all reads



## 2) Construct the sequence graph on (k-1)-mers

$f_1$	TTCAGG
$f_2$	TTCATGG
$f_3$	ATGGACA
$f_4$	TTCAT
$f_5$	CATCGAC
$f_6$	TCGAC
$f_7$	GACATC
$f_8$	ACATCGA



TTCA
TCAG
CAGG
TCAT
CATG
ATGG
TGGA
GGAC
GACA
CATC
ATCG
TCGA
CGAC
ACAT

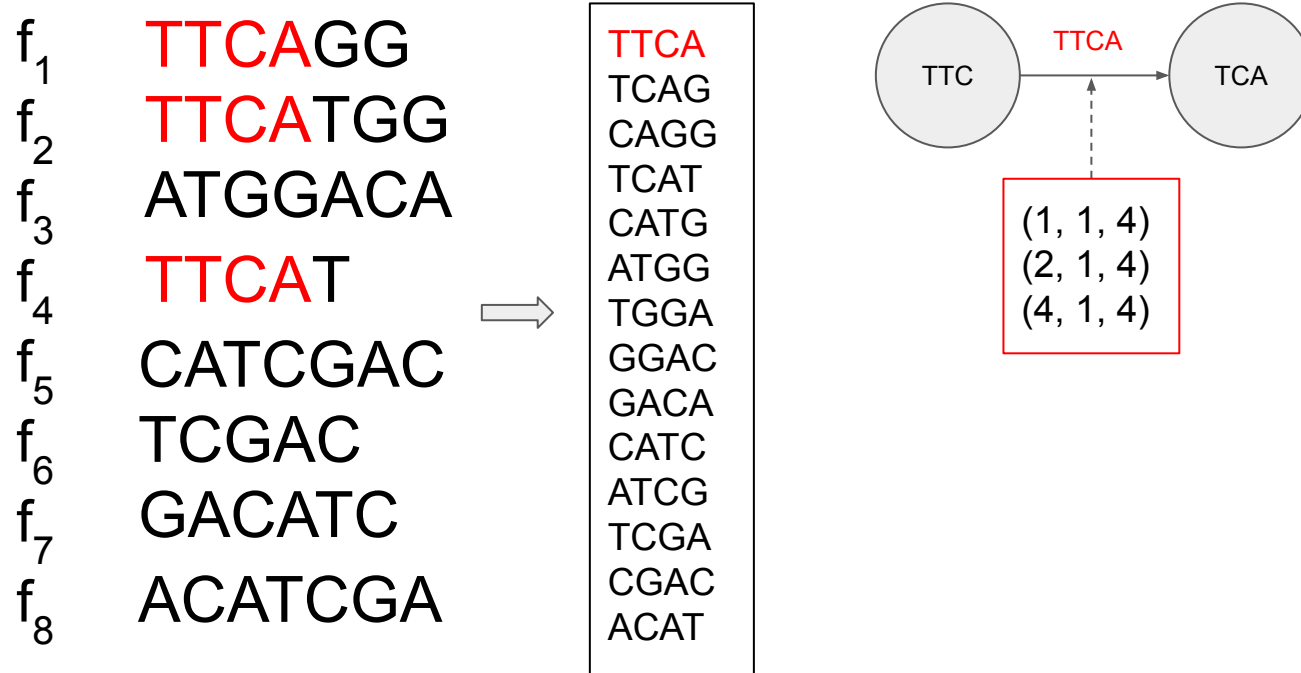
For each k-mer ( $a_1 \dots a_k$ ), we create an edge between nodes labeled  $a_1 \dots a_{k-1}$  and  $a_2 \dots a_k$ .

If those nodes do not exist yet, we add them to the graph.

We label the edge by its k-mer,  $a_1 \dots a_k$ .

We also store the set of position values  $(f, i, j)$  in each edge, which identify all occurrences of that k-mer by (fragment index, start position, end position)\*

## 2) Construct the sequence graph on (k-1)-mers

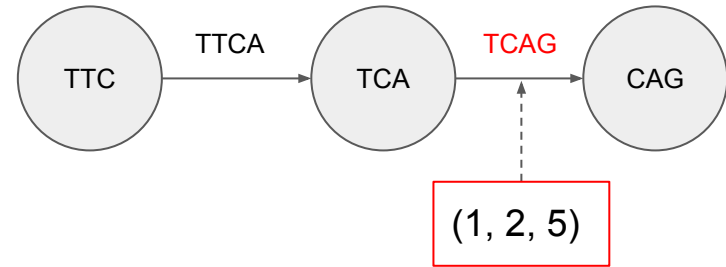


## 2) Construct the sequence graph on (k-1)-mers

$f_1$  TTCAGG  
 $f_2$  TTCATGG  
 $f_3$  ATGGACA  
 $f_4$  TTCAT  
 $f_5$  CATCGAC  
 $f_6$  TCGAC  
 $f_7$  GACATC  
 $f_8$  ACATCGA



TTCA  
TCAG  
CAGG  
TCAT  
CATG  
ATGG  
TGGA  
GGAC  
GACA  
CATC  
ATCG  
TCGA  
CGAC  
ACAT

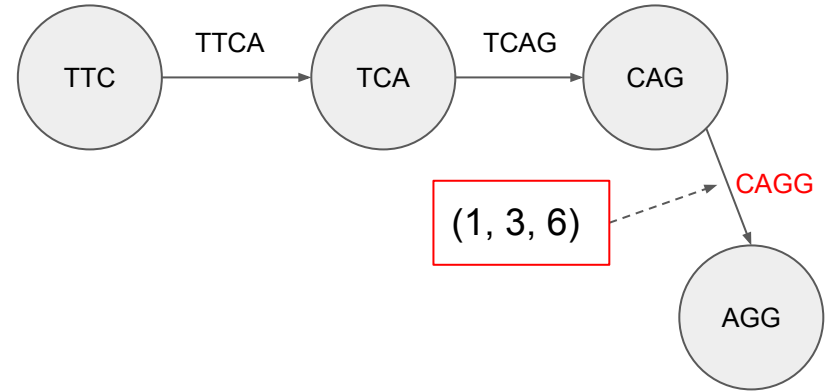


## 2) Construct the sequence graph on (k-1)-mers

$f_1$  TT**CAGG**  
 $f_2$  TTCATGG  
 $f_3$  ATGGACA  
 $f_4$  TTCAT  
 $f_5$  CATCGAC  
 $f_6$  TCGAC  
 $f_7$  GACATC  
 $f_8$  ACATCGA



TTCA  
TCAG  
**CAGG**  
TCAT  
CATG  
ATGG  
TGGA  
GGAC  
GACA  
CATC  
ATCG  
TCGA  
CGAC  
ACAT

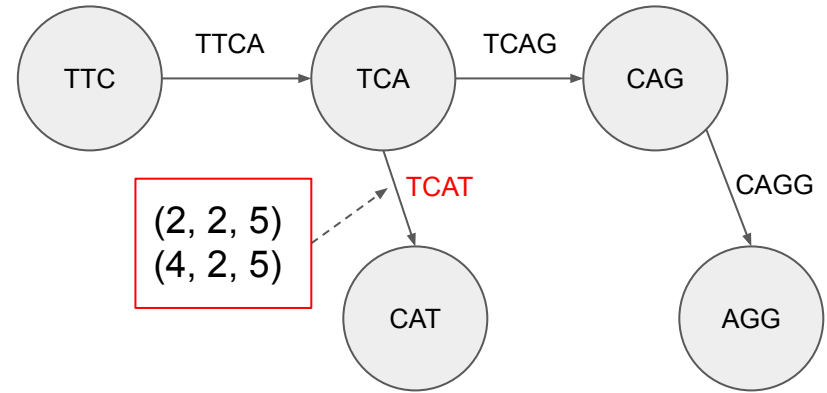


## 2) Construct the sequence graph on (k-1)-mers

$f_1$  TTCAGG  
 $f_2$  TTCATGG  
 $f_3$  ATGGACA  
 $f_4$  TTCAT  
 $f_5$  CATCGAC  
 $f_6$  TCGAC  
 $f_7$  GACATC  
 $f_8$  ACATCGA



TTCA  
TCAG  
CAGG  
TCAT  
CATG  
ATGG  
TGGG  
GGAC  
GACA  
CATC  
ATCG  
TCGA  
CGAC  
ACAT

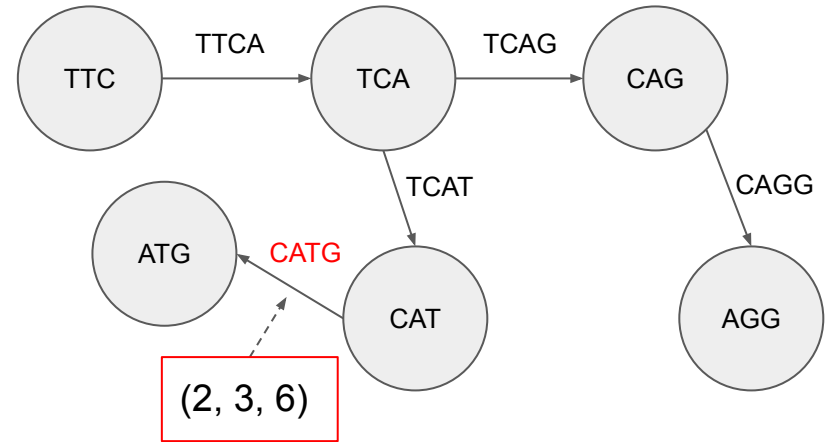


## 2) Construct the sequence graph on (k-1)-mers

$f_1$  TTCAGG  
 $f_2$  TT**CAT**GG  
 $f_3$  ATGGACA  
 $f_4$  TTCAT  
 $f_5$  CATCGAC  
 $f_6$  TCGAC  
 $f_7$  GACATC  
 $f_8$  ACATCGA



TTCATCA  
TCAGCAGG  
TCATCATG  
**CATG**ATGG  
TGGAGGAC  
GACACATC  
ATCGTCGA  
CGACACAT

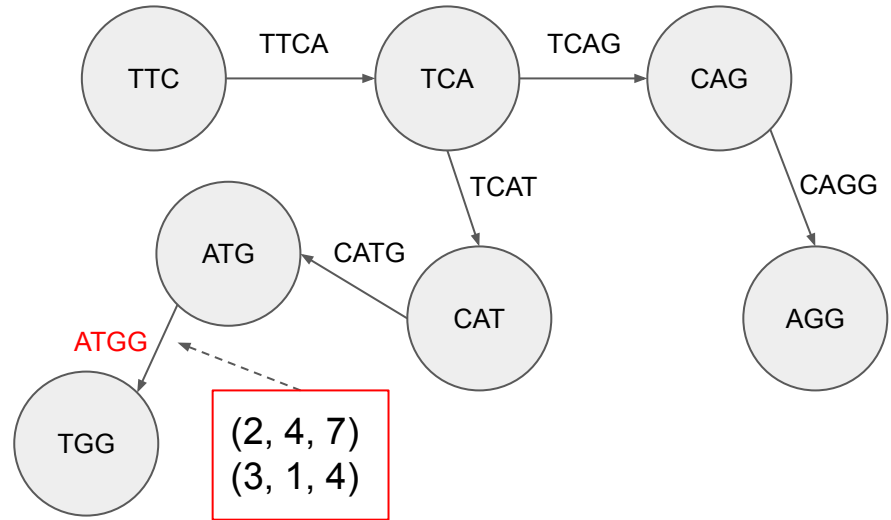


## 2) Construct the sequence graph on (k-1)-mers

$f_1$  TTCAGG  
 $f_2$  TTCATGG  
 $f_3$  ATGGACA  
 $f_4$  TTCAT  
 $f_5$  CATCGAC  
 $f_6$  TCGAC  
 $f_7$  GACATC  
 $f_8$  ACATCGA



TTCATCA  
TCAGCAGG  
TCATCATG  
CATGATGG  
TGGAGGAC  
GGACGACA  
GACATCATC  
ATCGTCGA  
TCGACGAC  
CGACACAT



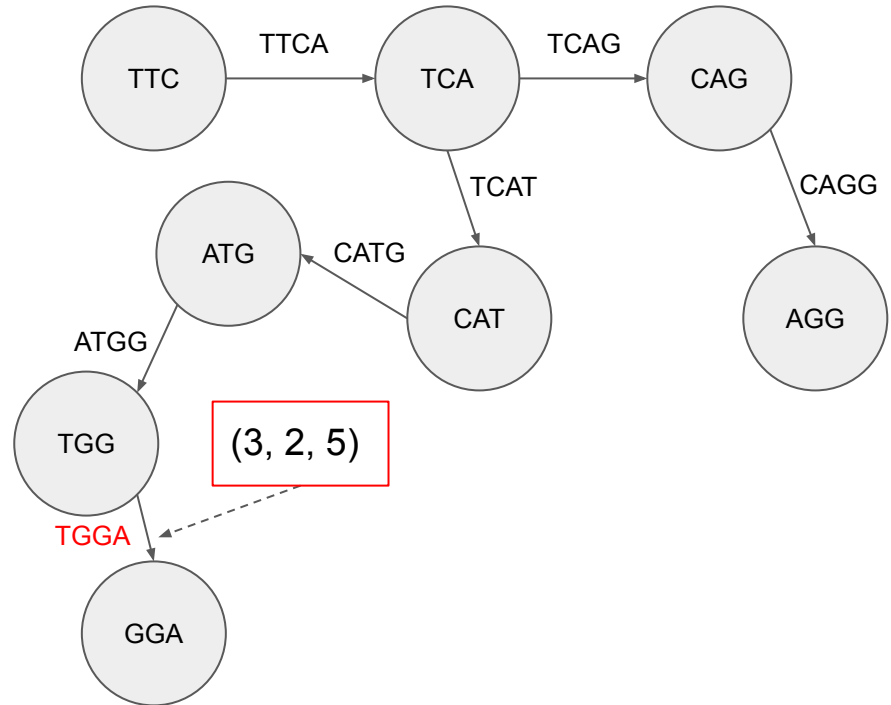


## 2) Construct the sequence graph on (k-1)-mers

$f_1$  TTCAGG  
 $f_2$  TTCATGG  
 $f_3$  ATGGACA  
 $f_4$  TTCAT  
 $f_5$  CATCGAC  
 $f_6$  TCGAC  
 $f_7$  GACATC  
 $f_8$  ACATCGA



TTCATCA  
TCAGCAGG  
TCATCATG  
CATGATGG  
TGGAGGAC  
GACAGCATC  
ATCGTCGA  
TCGACGAC  
CGACACAT

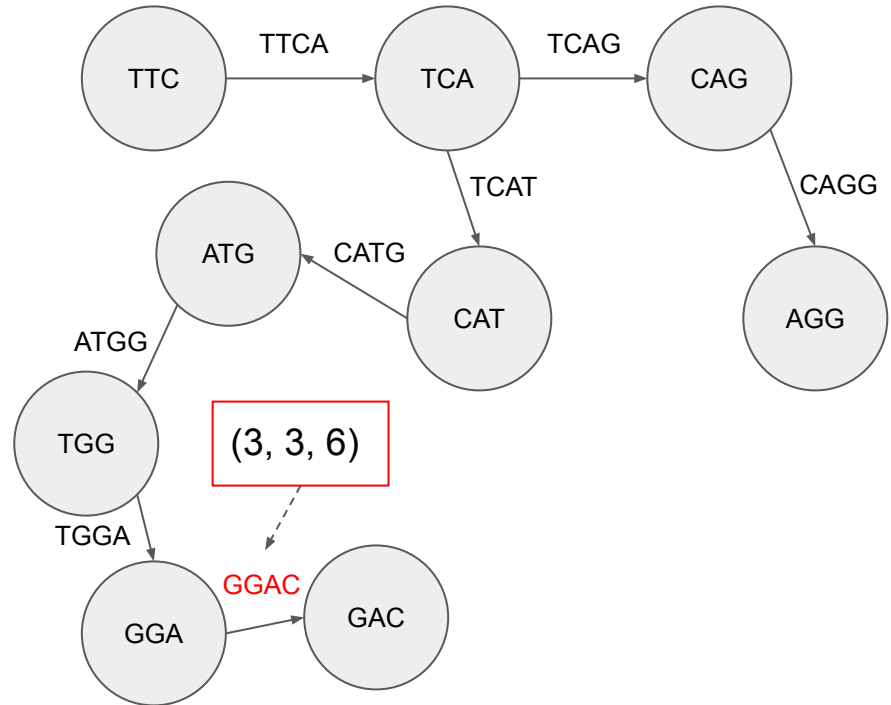


## 2) Construct the sequence graph on (k-1)-mers

$f_1$  TTCAGG  
 $f_2$  TTCATGG  
 $f_3$  ATGGACA  
 $f_4$  TTCAT  
 $f_5$  CATCGAC  
 $f_6$  TCGAC  
 $f_7$  GACATC  
 $f_8$  ACATCGA



TTCATCA  
TCAGCAGG  
TCATCATG  
CATGATGG  
TGGAGGAC  
GACACATC  
ATCGTCGA  
TCGACGAC  
ACAT

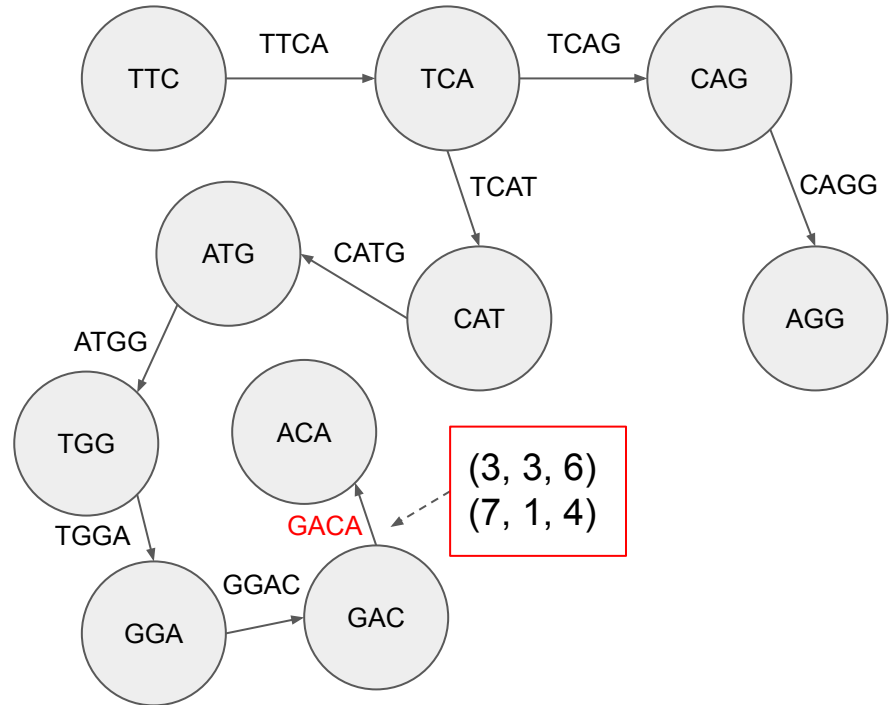


## 2) Construct the sequence graph on (k-1)-mers

$f_1$  TTCAGG  
 $f_2$  TTCATGG  
 $f_3$  ATG**GACA**  
 $f_4$  TTCAT  
 $f_5$  CATCGAC  
 $f_6$  TCGAC  
 $f_7$  **GAC**ATC  
 $f_8$  ACATCGA



TTCATCA  
TCAGCAGG  
TCATCATG  
CATGATGG  
TGGAGGAC  
**GACACATC**  
ATCGTCGA  
CGACACAT

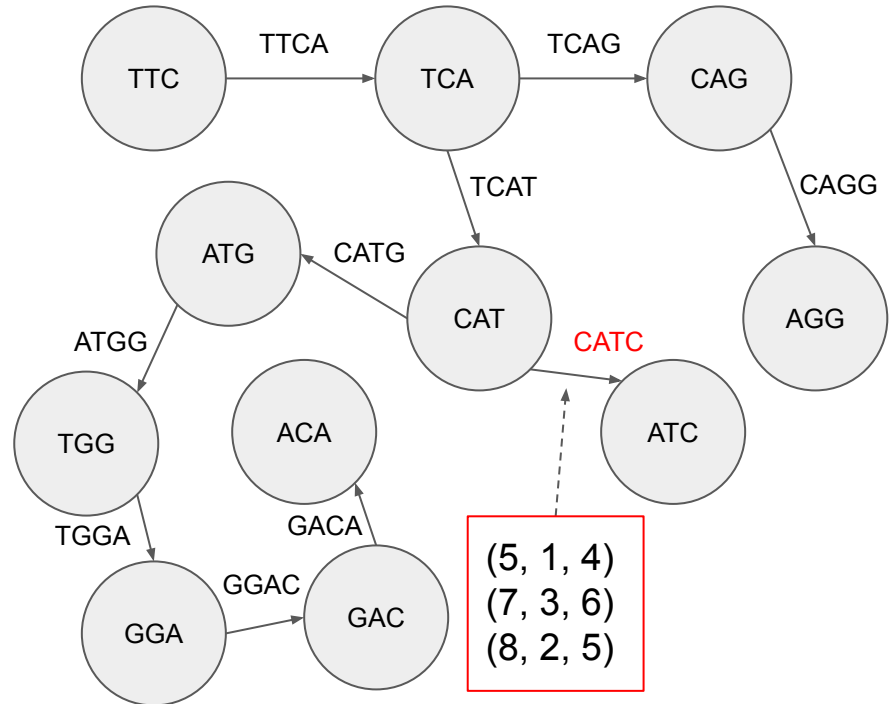


## 2) Construct the sequence graph on (k-1)-mers

$f_1$  TTCAGG  
 $f_2$  TTCATGG  
 $f_3$  ATGGACA  
 $f_4$  TTCAT  
 $f_5$  **CATC**GAC  
 $f_6$  TCGAC  
 $f_7$  GAC**CATC**  
 $f_8$  AC**CATC**GA



TTCAT  
TCAGG  
CAGG  
TCATG  
CATGG  
ATGGG  
TGGGA  
GGACG  
GACAT  
**CATC**  
ATCGA  
TCGAC  
CGACG  
ACATG

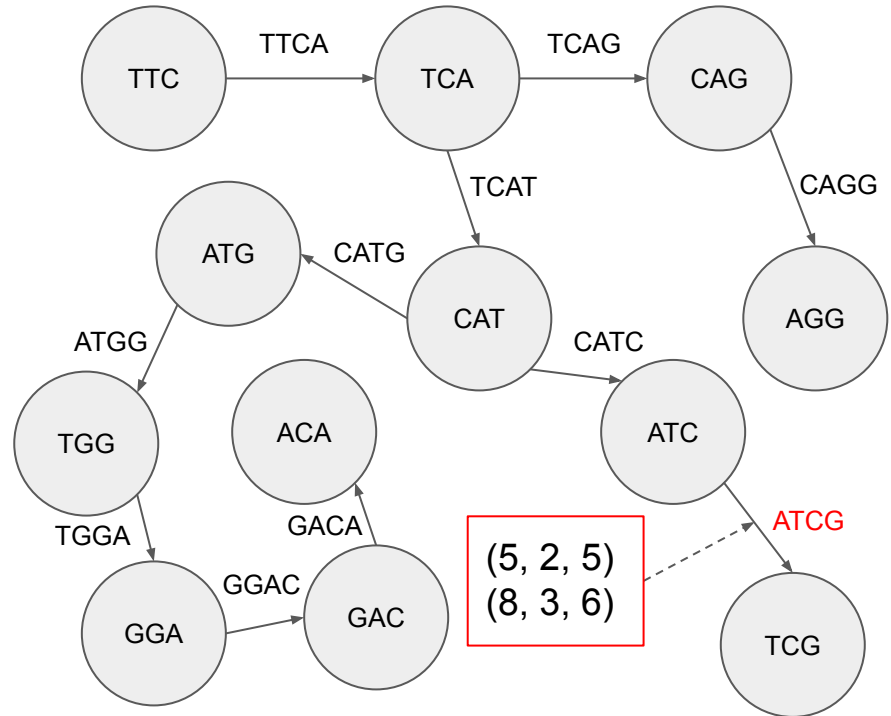


## 2) Construct the sequence graph on (k-1)-mers

$f_1$  TTCAGG  
 $f_2$  TTCATGG  
 $f_3$  ATGGACA  
 $f_4$  TTCAT  
 $f_5$  CATCGAC  
 $f_6$  TCGAC  
 $f_7$  GACATC  
 $f_8$  ACATCGA



TTCATCA  
TCAGCAGG  
TCATCATG  
CATGATGG  
TGGAGGAC  
GGACGACA  
GACATCATC  
ATCGTCGA  
TCGACGAC  
CGACACAT

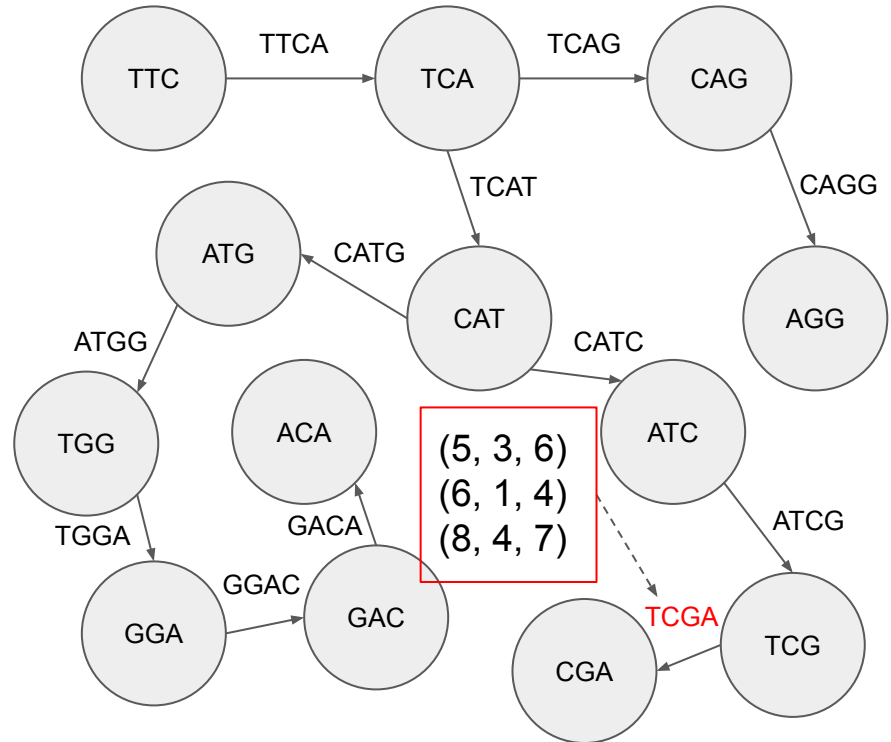


## 2) Construct the sequence graph on (k-1)-mers

$f_1$  TTCAGG  
 $f_2$  TTCATGG  
 $f_3$  ATGGACA  
 $f_4$  TTCAT  
 $f_5$  CATCGAC  
 $f_6$  TCGAC  
 $f_7$  GACATC  
 $f_8$  ACATCGA



TTCATCA  
TCAGCAGG  
TCATCATG  
CATGATGG  
TGGATGGA  
GGACGACA  
GACATCATC  
ATCGTCGA  
CGACACAT

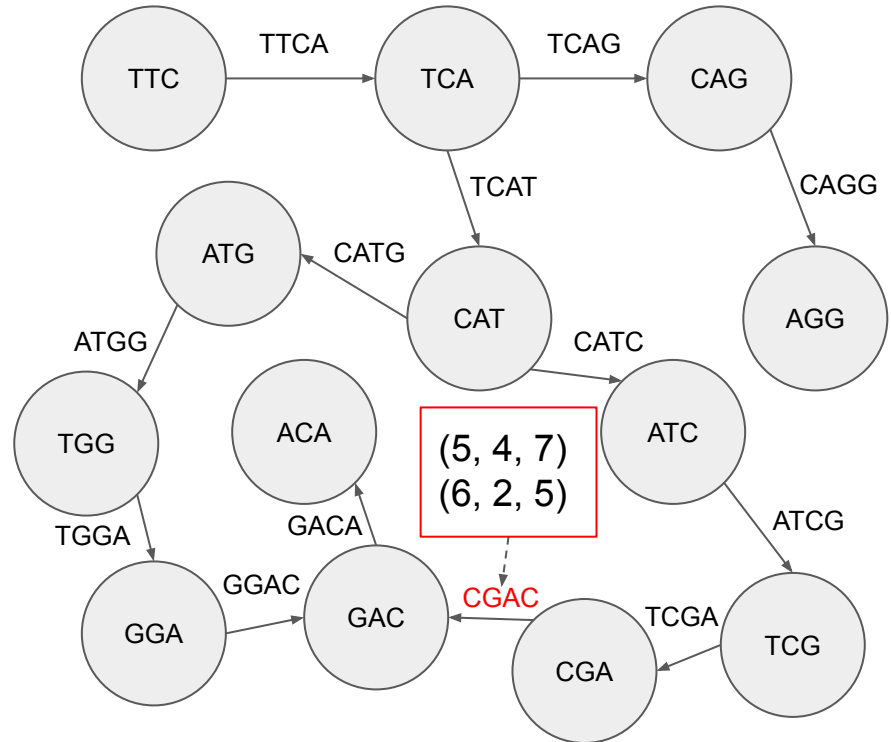


## 2) Construct the sequence graph on (k-1)-mers

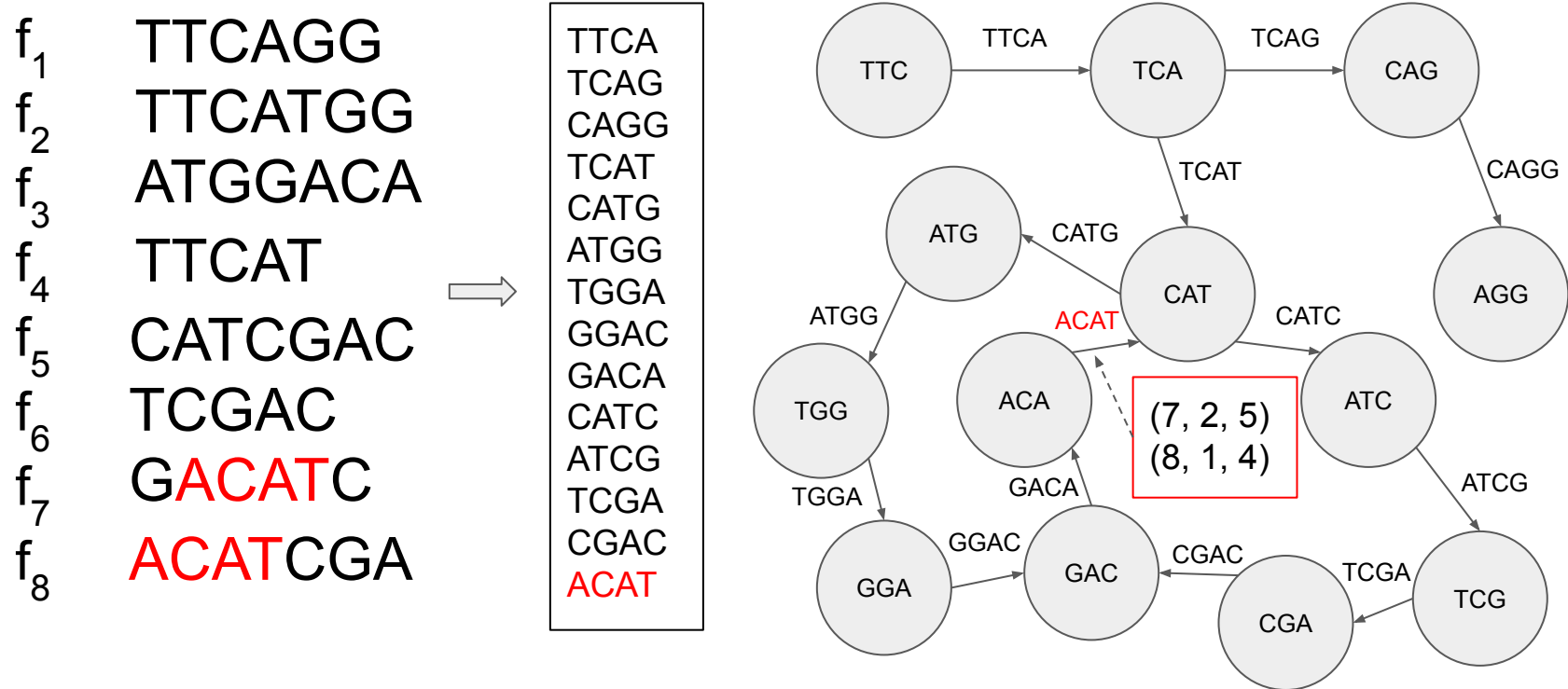
$f_1$  TTCAGG  
 $f_2$  TTCATGG  
 $f_3$  ATGGACA  
 $f_4$  TTCAT  
 $f_5$  CATCGAC  
 $f_6$  TCGAC  
 $f_7$  GACATC  
 $f_8$  ACATCGA



TTCATCA  
TCAGTCA  
CAGGCAG  
TCATCAG  
CATGCAT  
ATGGATG  
TGGATGG  
GGACGGA  
GACATGGA  
CATCGAC  
ATCGGAC  
TCGACGA  
CGACGGA  
ACATGGA



## 2) Construct the sequence graph on (k-1)-mers



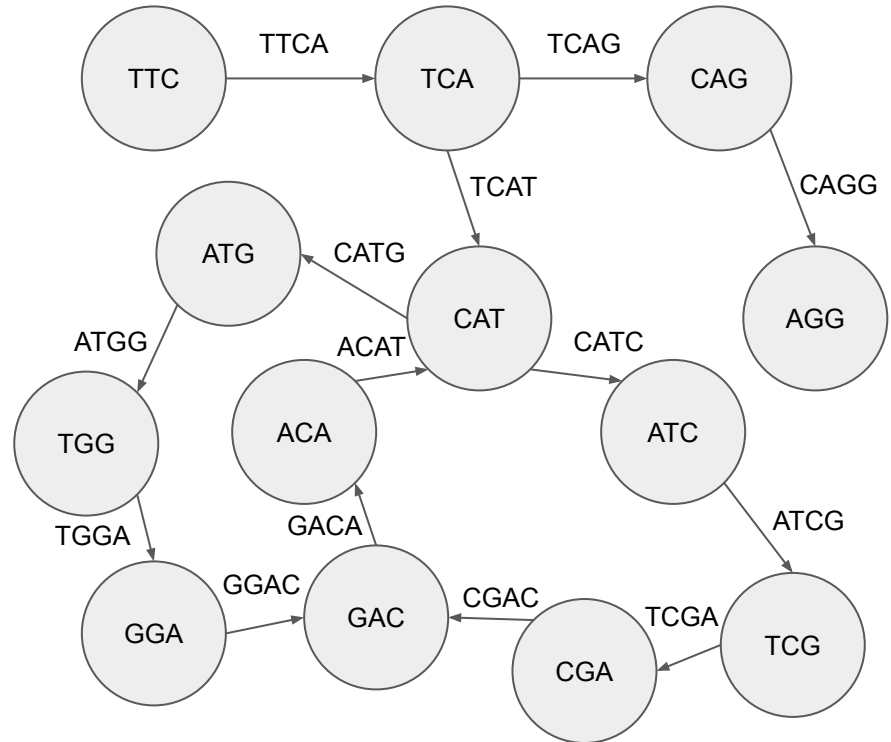


## 2) Construct the sequence graph on (k-1)-mers

$f_1$  TTCAGG  
 $f_2$  TTCATGG  
 $f_3$  ATGGACA  
 $f_4$  TTCAT  
 $f_5$  CATCGAC  
 $f_6$  TCGAC  
 $f_7$  GACATC  
 $f_8$  ACATCGA

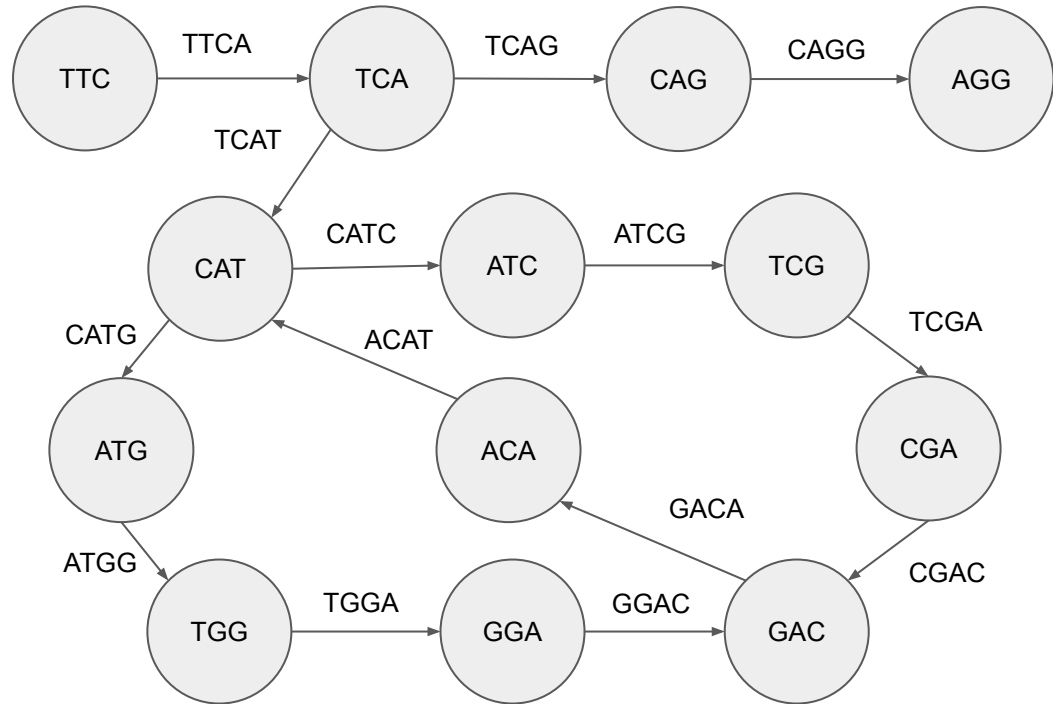


TTCATCA  
TCAGCAGG  
TCATCATG  
CATGATGG  
TGGATGGA  
GGACGGAC  
GACACATC  
ATCGTCGA  
CGACCGAC  
ACATACAT



## 2) Construct the sequence graph on (k-1)-mers

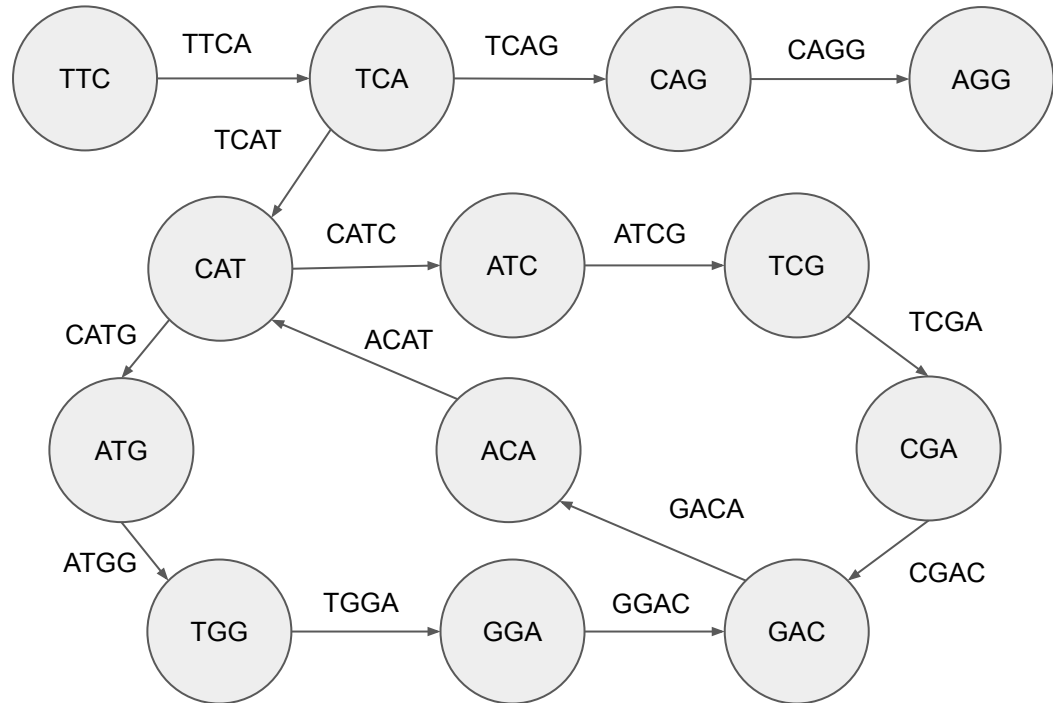
$f_1$  TTCAGG  
 $f_2$  TTCATGG  
 $f_3$  ATGGACA  
 $f_4$  TTCAT  
 $f_5$  CATCGAC  
 $f_6$  TCGAC  
 $f_7$  GACATC  
 $f_8$  ACATCGA



## 2) Construct the sequence graph on (k-1)-mers

This graph is sufficient to begin searching for an Eulerian path, but we can simplify the graph beforehand to make inference simpler.

There are three types of graph reductions that are possible, which we will illustrate next.



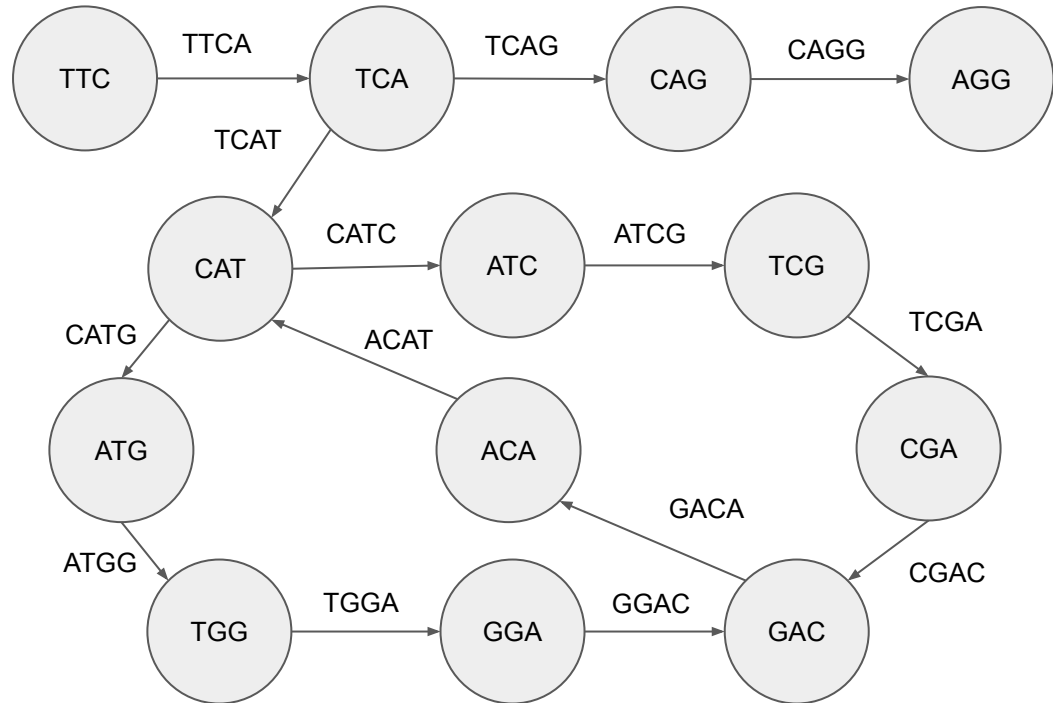
## 2) Construct the sequence graph on (k-1)-mers

This graph is sufficient to begin searching for an Eulerian path, but we can simplify the graph beforehand to make inference simpler.

There are three types of graph reductions that are possible, which we will illustrate next.

Why did we bother storing positional information,  $(f, i, j)$ ?

These are useful for establishing continuity when performing reductions and the Eulerian tour.



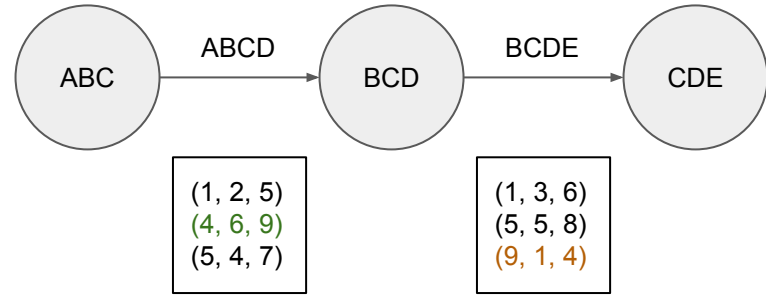
## 2a) Graph reductions: singletons

Singletons are nodes with indegree = 1 and outdegree = 1 (i.e., a node  $v$  with exactly one incoming edge  $[u, v]$  and exactly one outgoing edge  $[v, w]$ ).

We simplify this feature by removing the node  $v$  and its incident edges, replacing it with a new edge  $[u, w]$ .

This new edge has a label which merges the two labels of the previous edges.

Likewise, it stores the position tuples of the previous edges, merged where possible.\*



Formally, this is:

$$\{(f, i, m) \mid (f, i, j+k-2) \in [u, v] \text{ and } (f, j+k-2, m) \in [v, w]\} \\ \cup \{\text{fragment ends of } [u, v]\} \cup \{\text{fragment starts of } [v, w]\}$$

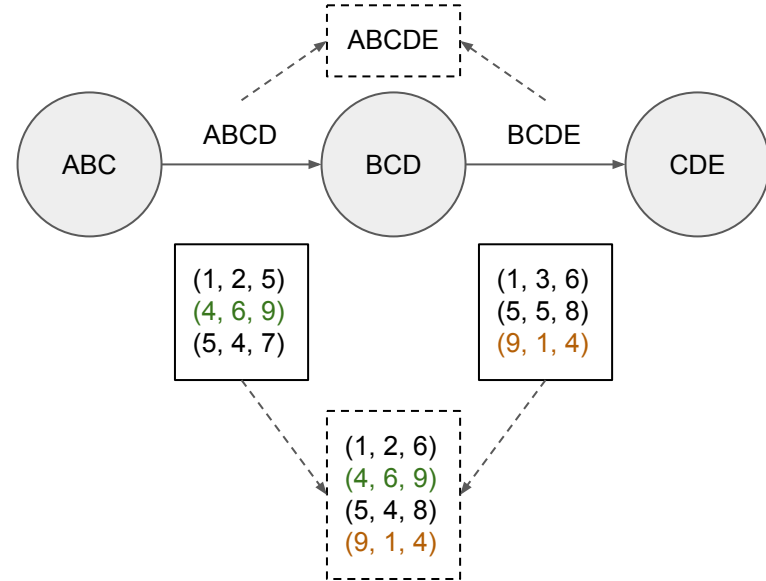
## 2a) Graph reductions: singletons

Singletons are nodes with indegree = 1 and outdegree = 1 (i.e., a node  $v$  with exactly one incoming edge  $[u, v]$  and exactly one outgoing edge  $[v, w]$ ).

We simplify this feature by removing the node  $v$  and its incident edges, replacing it with a new edge  $[u, w]$ .

This new edge has a label which merges the two labels of the previous edges.

Likewise, it stores the position tuples of the previous edges, merged where possible.\*



Formally, this is:

$$\{(f, i, m) \mid (f, i, j+k-2) \in [u, v] \text{ and } (f, j+k-2, m) \in [v, w]\} \\ \cup \{\text{fragment ends of } [u, v]\} \cup \{\text{fragment starts of } [v, w]\}$$

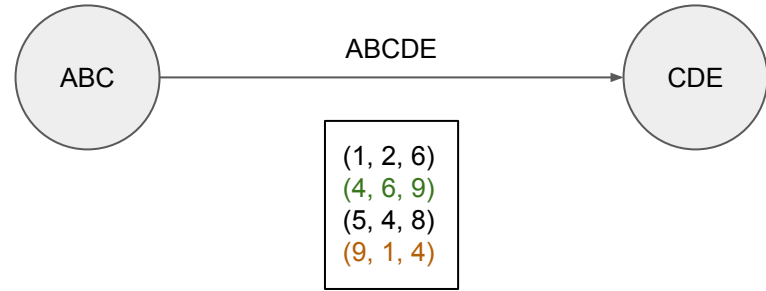
## 2a) Graph reductions: singletons

Singletons are nodes with indegree = 1 and outdegree = 1 (i.e., a node  $v$  with exactly one incoming edge  $[u, v]$  and exactly one outgoing edge  $[v, w]$ ).

We simplify this feature by removing the node  $v$  and its incident edges, replacing it with a new edge  $[u, w]$ .

This new edge has a label which merges the two labels of the previous edges.

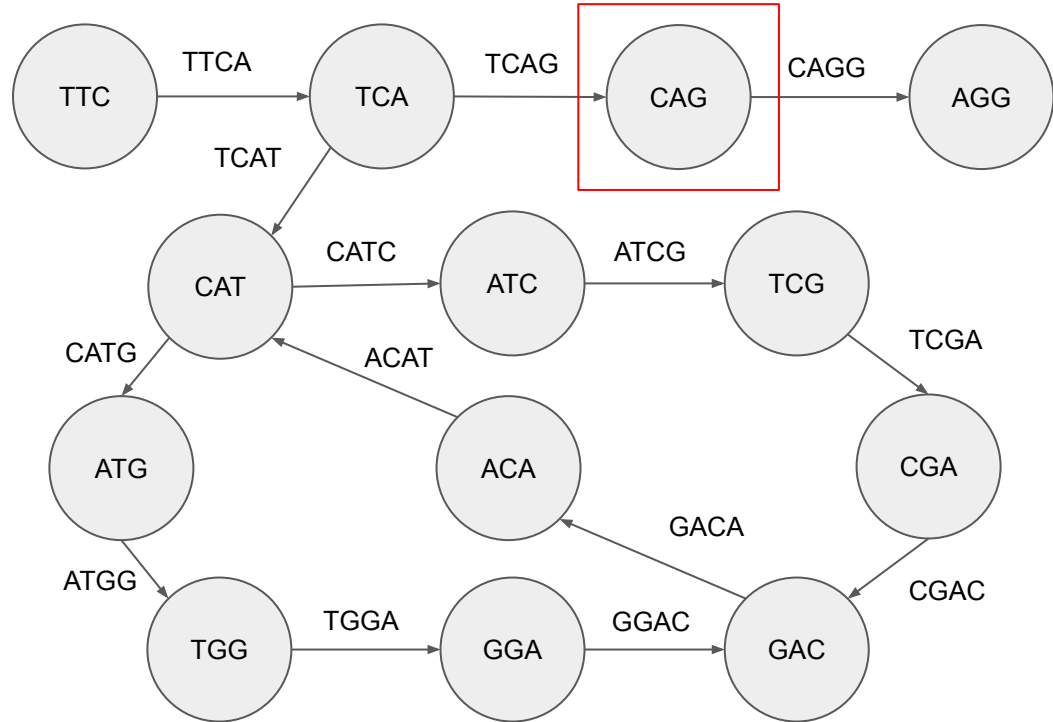
Likewise, it stores the position tuples of the previous edges, merged where possible.\*



Formally, this is:

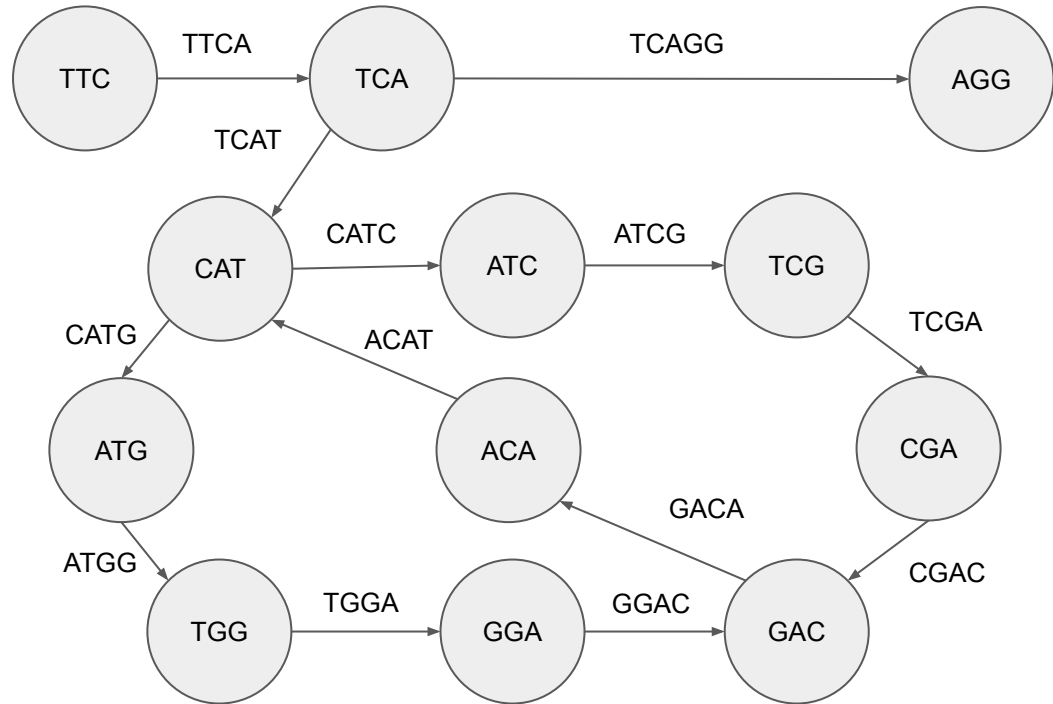
$$\{(f, i, m) \mid (f, i, j+k-2) \in [u, v] \text{ and } (f, j+k-2, m) \in [v, w]\} \\ \cup \{\text{fragment ends of } [u, v]\} \cup \{\text{fragment starts of } [v, w]\}$$

## 2a) Graph reductions: singletons

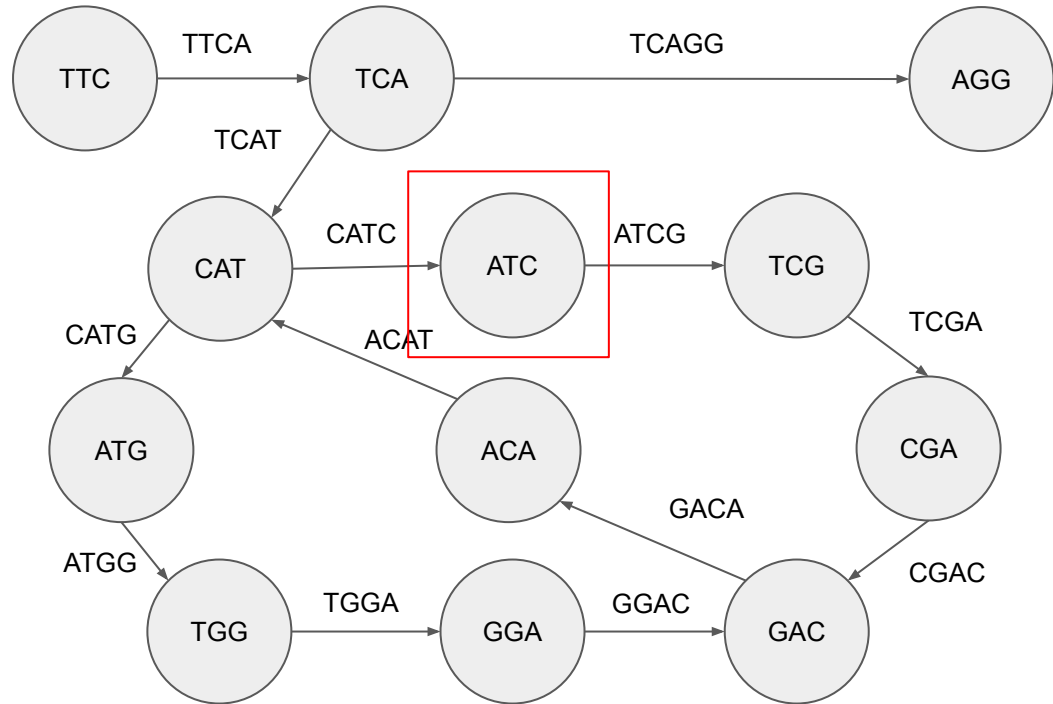




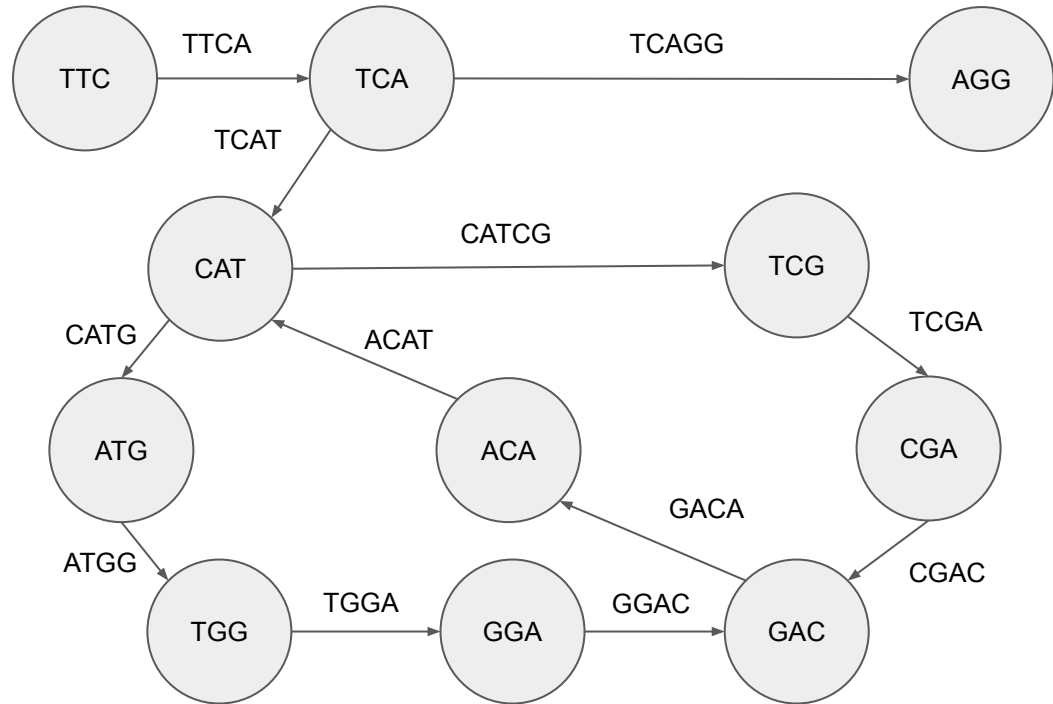
## 2a) Graph reductions: singletons



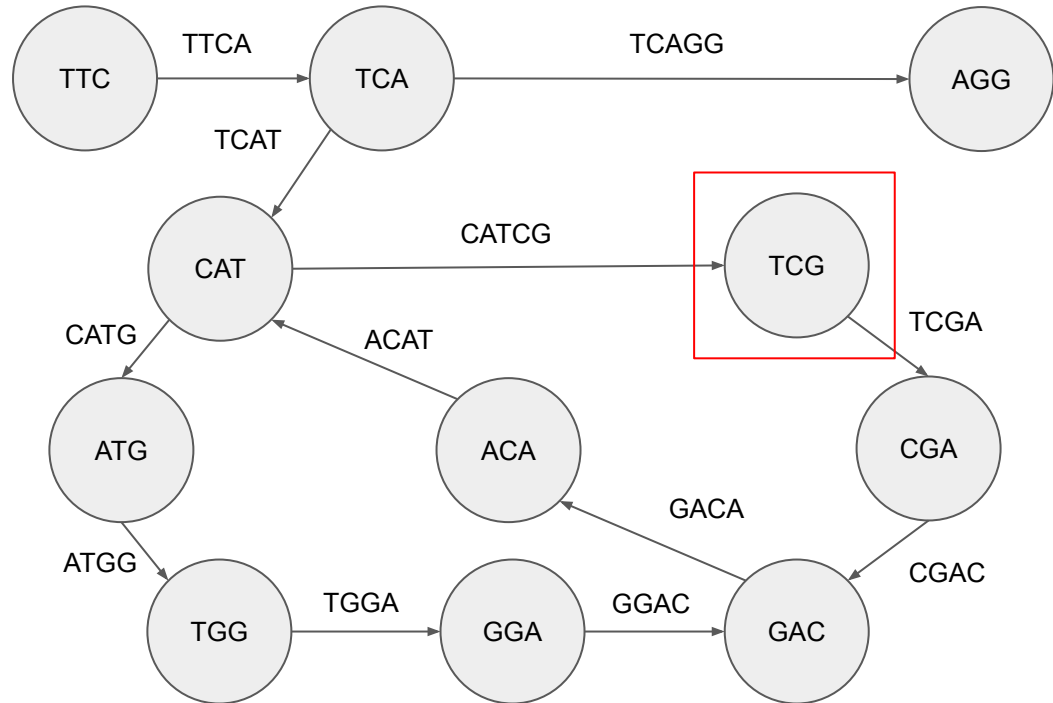
## 2a) Graph reductions: singletons



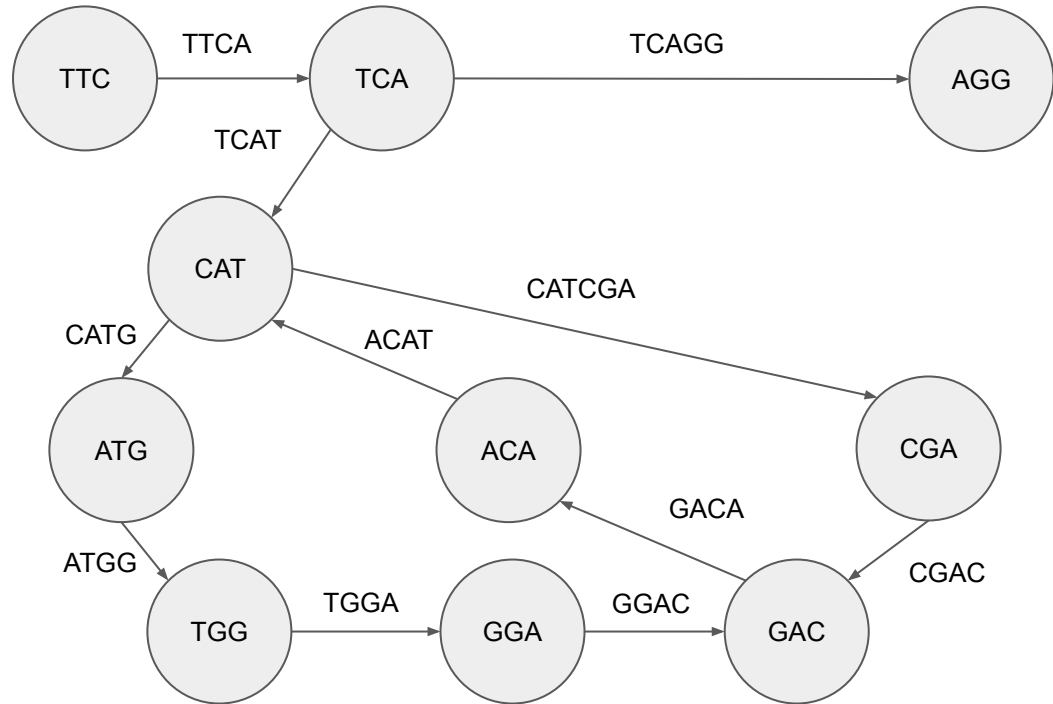
## 2a) Graph reductions: singletons



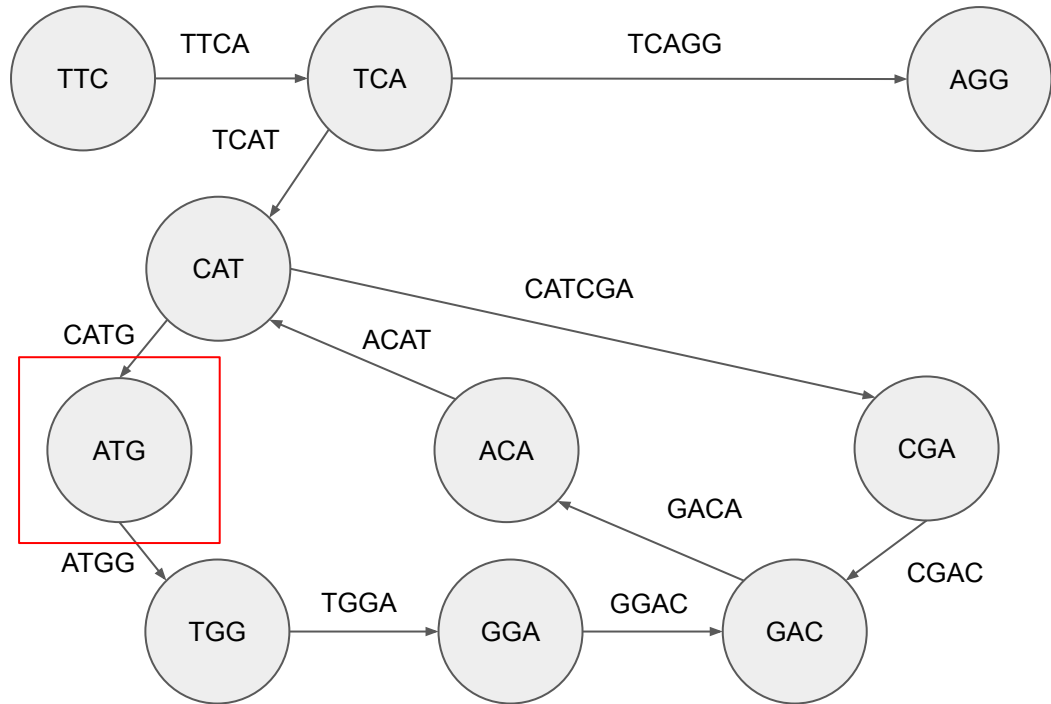
## 2a) Graph reductions: singletons



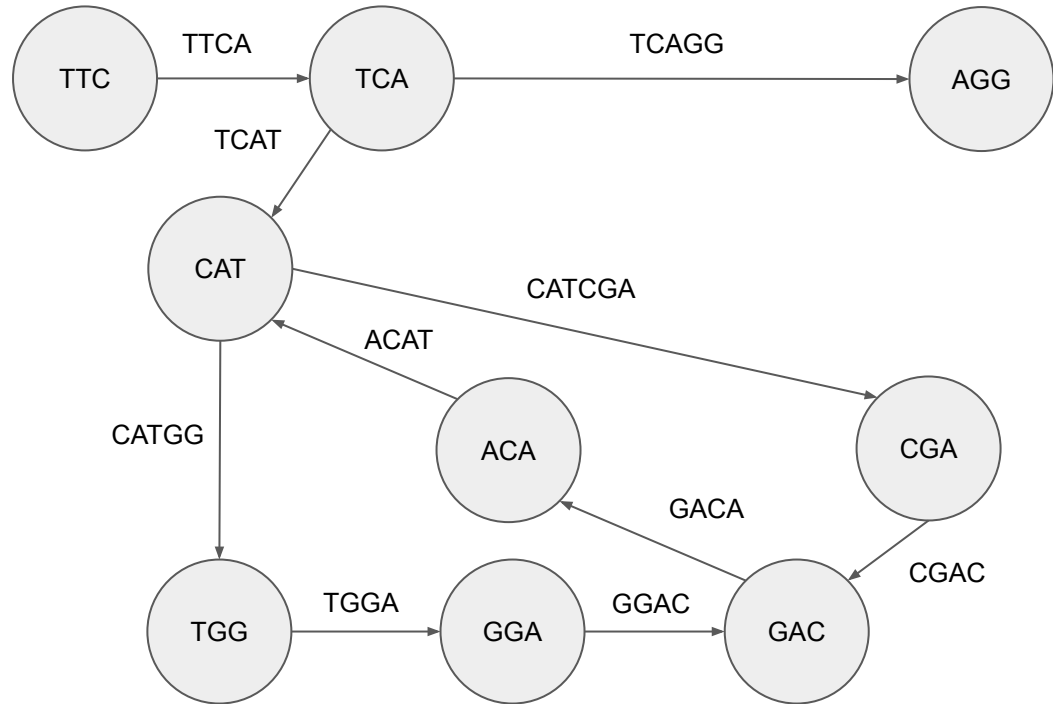
## 2a) Graph reductions: singletons



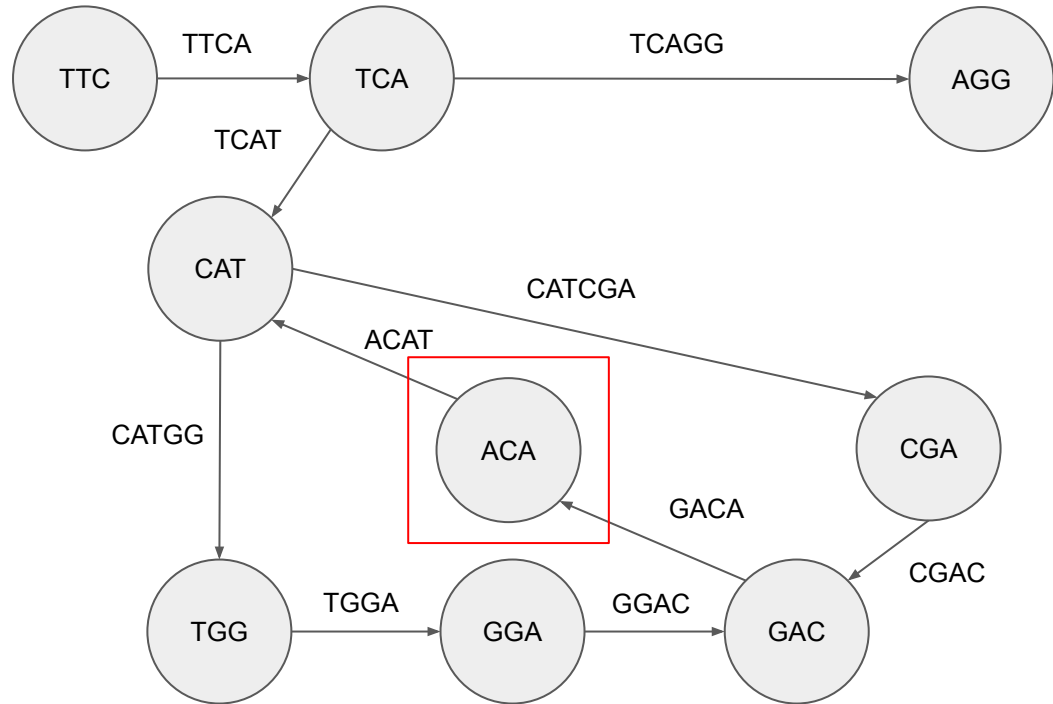
## 2a) Graph reductions: singletons



## 2a) Graph reductions: singletons

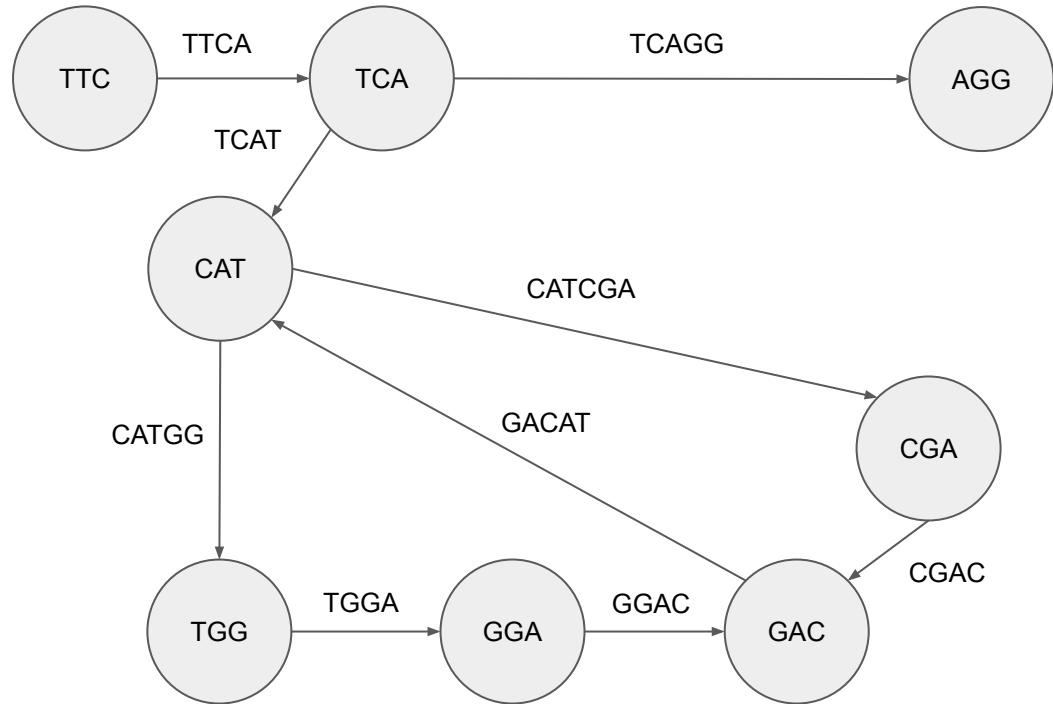


## 2a) Graph reductions: singletons

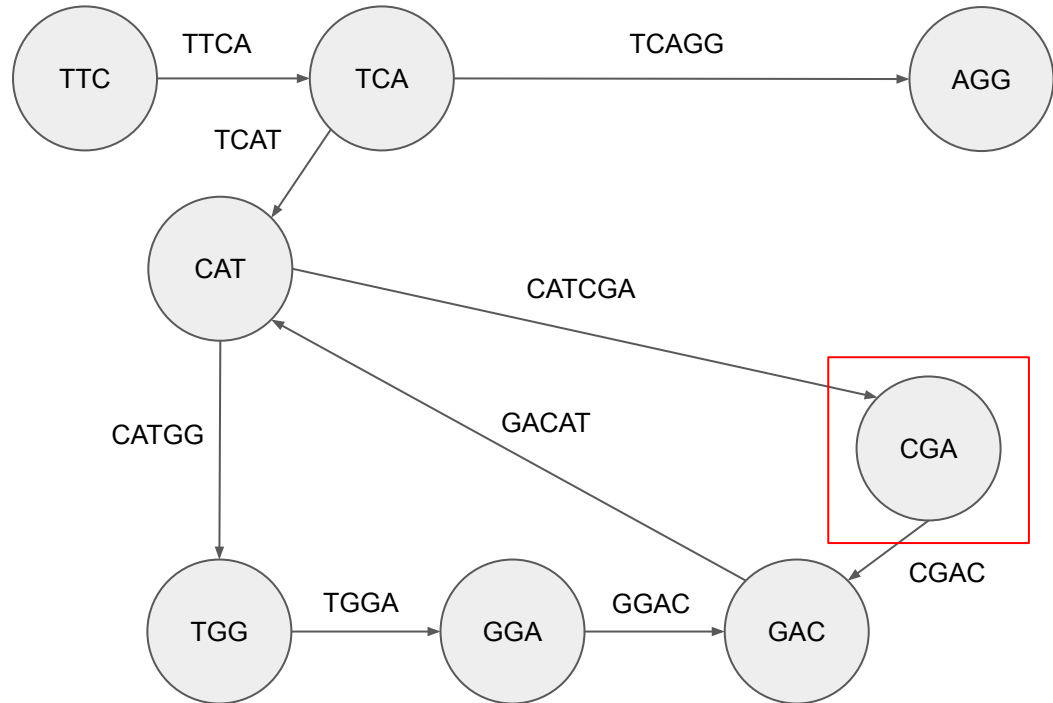




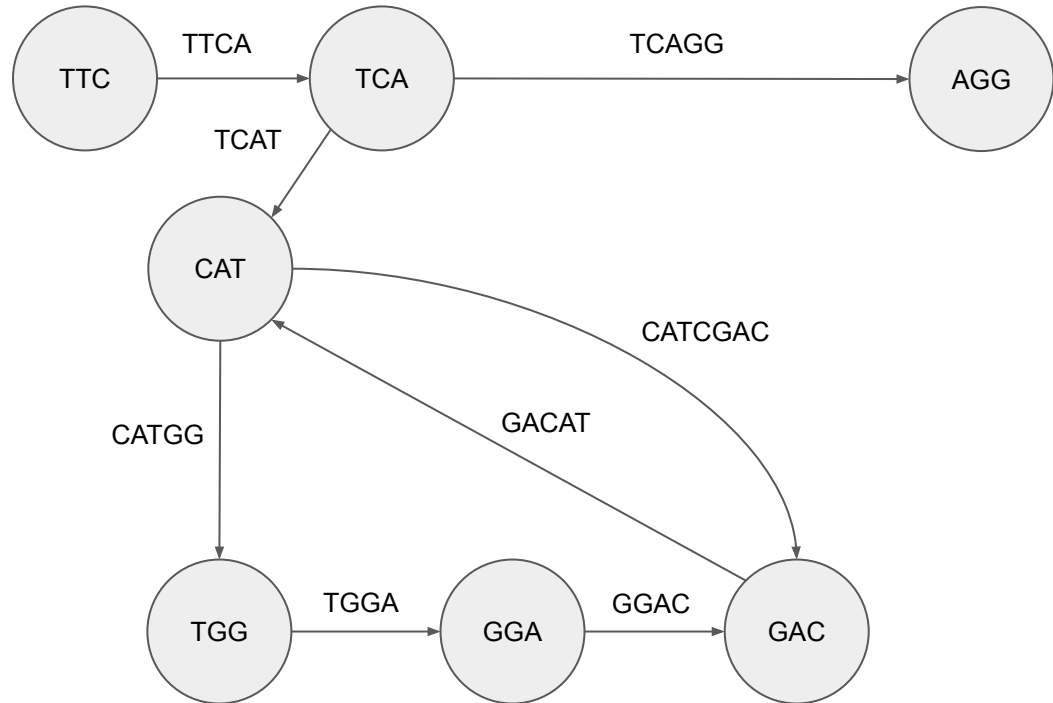
## 2a) Graph reductions: singletons



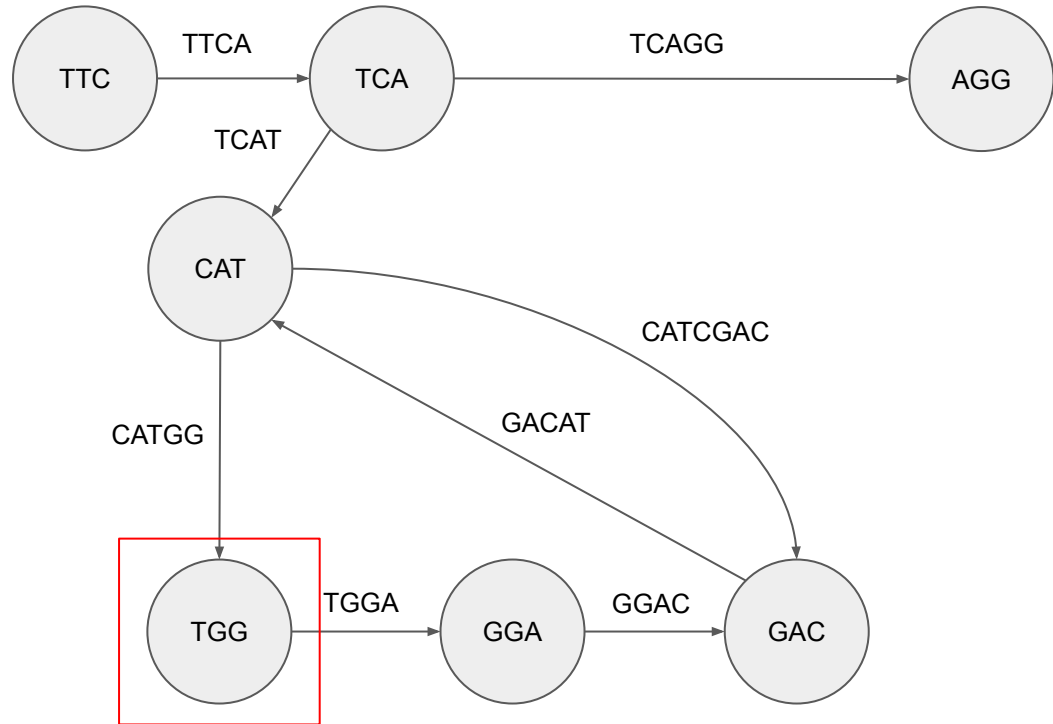
## 2a) Graph reductions: singletons



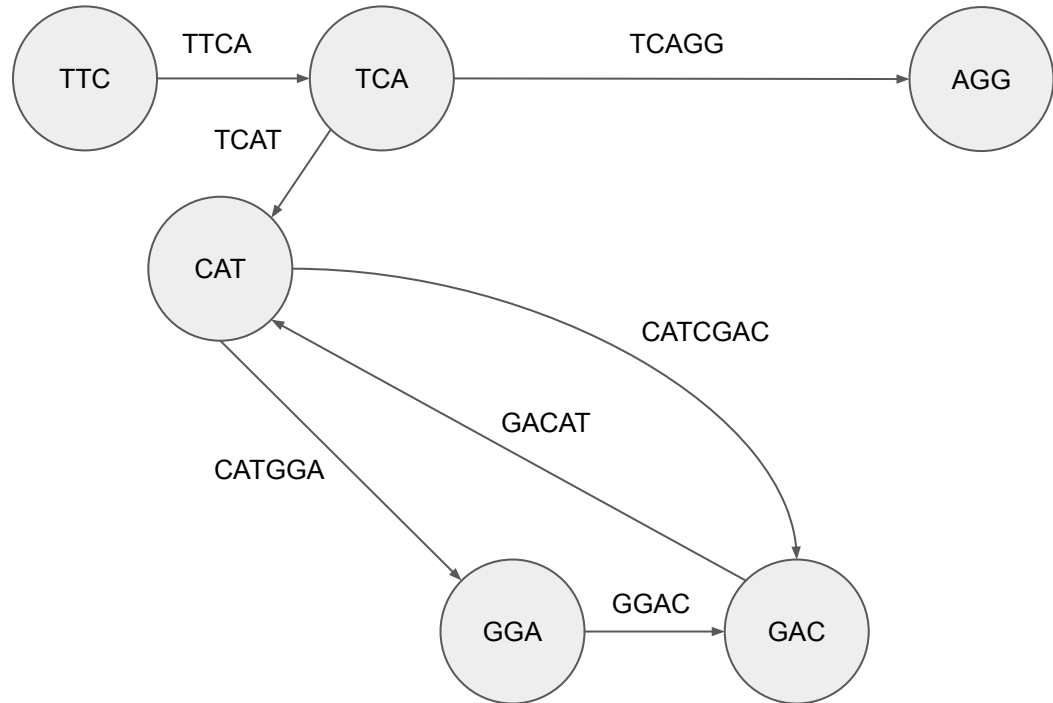
## 2a) Graph reductions: singletons



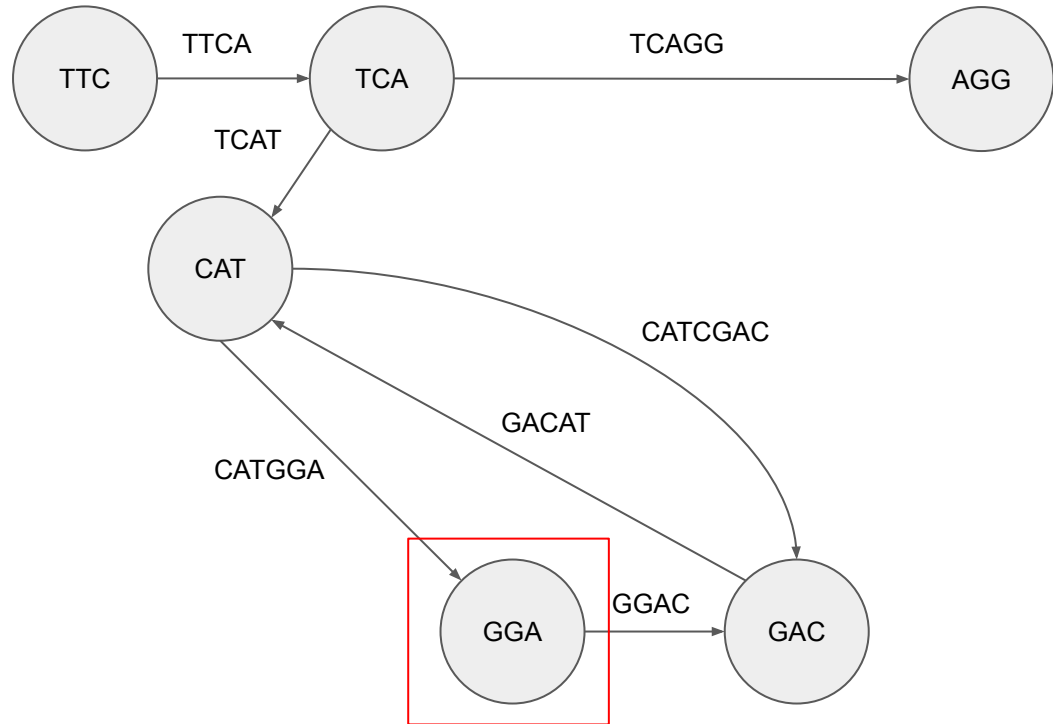
## 2a) Graph reductions: singletons



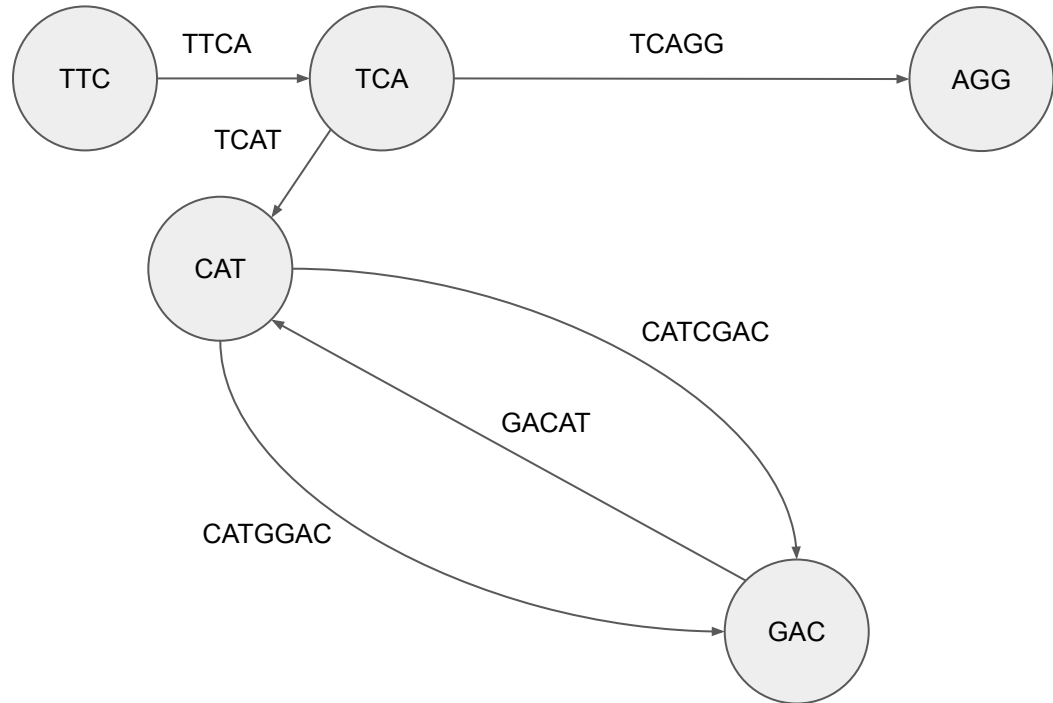
## 2a) Graph reductions: singletons



## 2a) Graph reductions: singletons



## 2a) Graph reductions: singletons

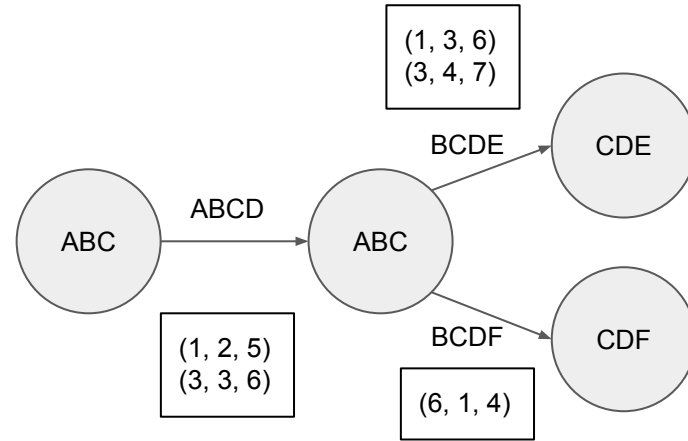


## 2b) Graph reductions: forks

Forks are nodes with indegree = 1 and outdegree > 1 (i.e., a node  $v$  with exactly one incoming edge  $[u, v]$  and outgoing edges  $[v, w_1], [v, w_2], \dots$ ).\*

A node with outdegree > 1 indicates either a sequencing error or a repetitive region. We apply a heuristic approach to resolve this feature.

Of the outgoing edges, keep only the one which has the most occurrences continuing from the incoming edge. If the number of such occurrences is roughly equal for all edges, leave the fork in the graph.\*



“Reverse forks” (i.e., nodes with indegree > 1 and outdegree = 1) can also be resolved analogously.

Idury and Waterman formalize this heuristic as the **overlap test**, but the explanation here is the rough intuition behind its behavior.

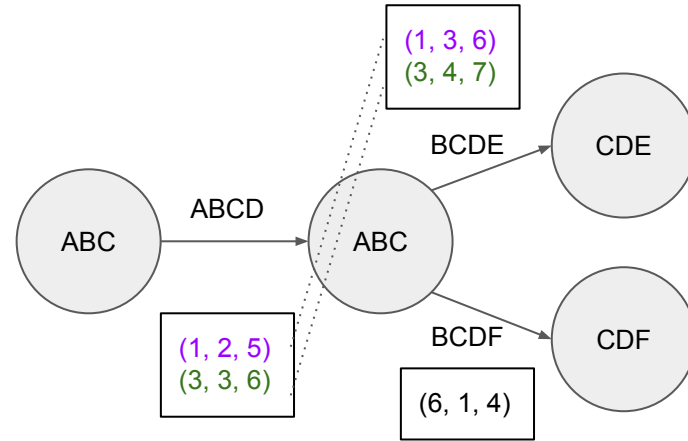


## 2b) Graph reductions: forks

Forks are nodes with indegree = 1 and outdegree > 1 (i.e., a node  $v$  with exactly one incoming edge  $[u, v]$  and outgoing edges  $[v, w_1], [v, w_2], \dots$ ).\*

A node with outdegree > 1 indicates either a sequencing error or a repetitive region. We apply a heuristic approach to resolve this feature.

Of the outgoing edges, keep only the one which has the most occurrences continuing from the incoming edge. If the number of such occurrences is roughly equal for all edges, leave the fork in the graph.\*



“Reverse forks” (i.e., nodes with indegree > 1 and outdegree = 1) can also be resolved analogously.

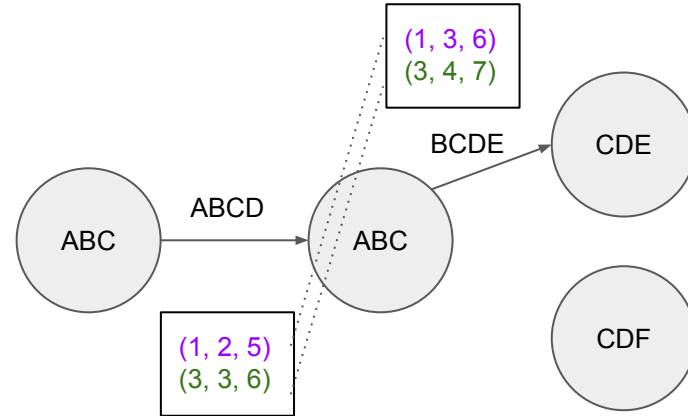
Idury and Waterman formalize this heuristic as the **overlap test**, but the explanation here is the rough intuition behind its behavior.

## 2b) Graph reductions: forks

Forks are nodes with indegree = 1 and outdegree > 1 (i.e., a node  $v$  with exactly one incoming edge  $[u, v]$  and outgoing edges  $[v, w_1], [v, w_2], \dots$ ).\*

A node with outdegree > 1 indicates either a sequencing error or a repetitive region. We apply a heuristic approach to resolve this feature.

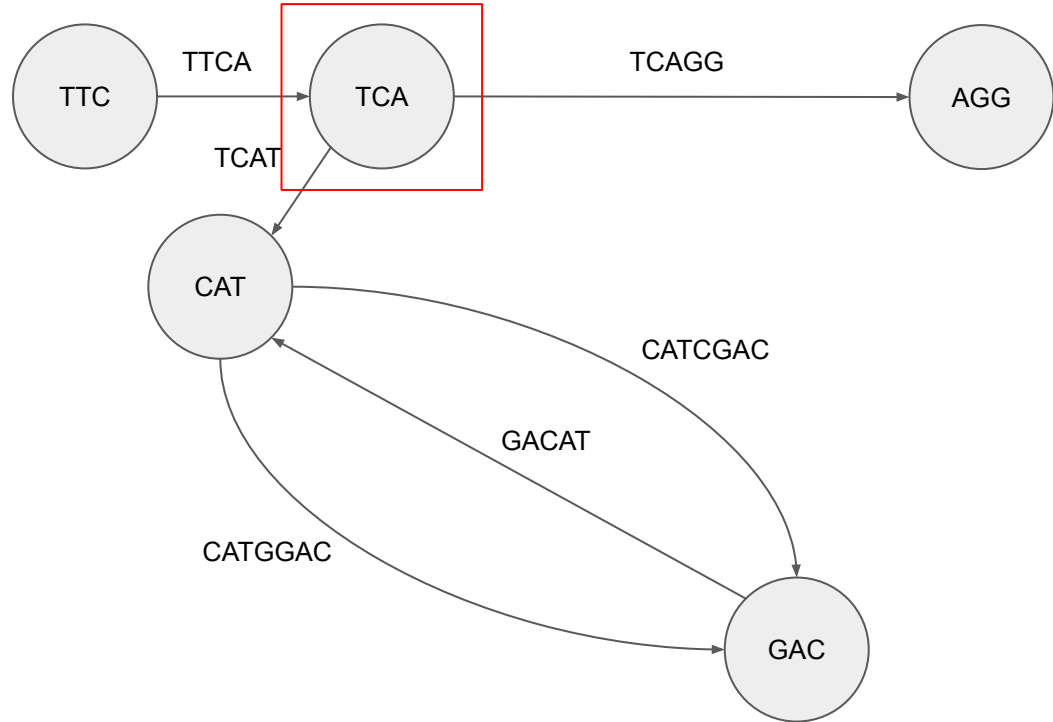
Of the outgoing edges, keep only the one which has the most occurrences continuing from the incoming edge. If the number of such occurrences is roughly equal for all edges, leave the fork in the graph.\*



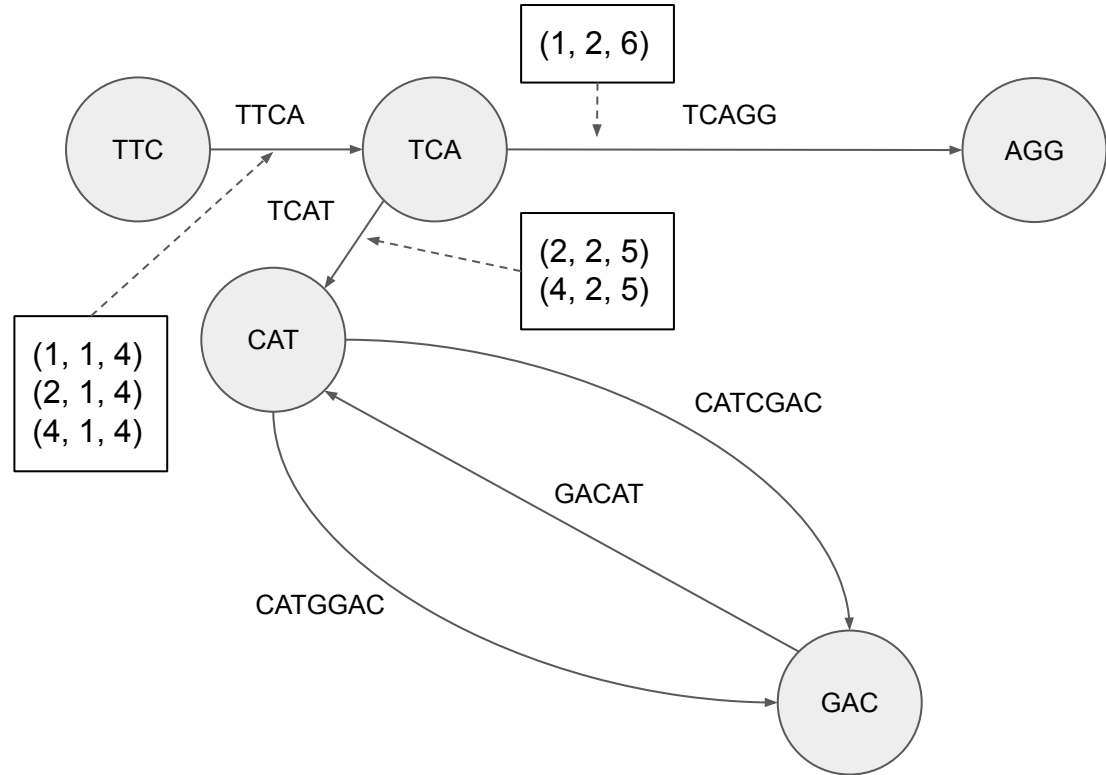
“Reverse forks” (i.e., nodes with indegree > 1 and outdegree = 1) can also be resolved analogously.

Idury and Waterman formalize this heuristic as the **overlap test**, but the explanation here is the rough intuition behind its behavior.

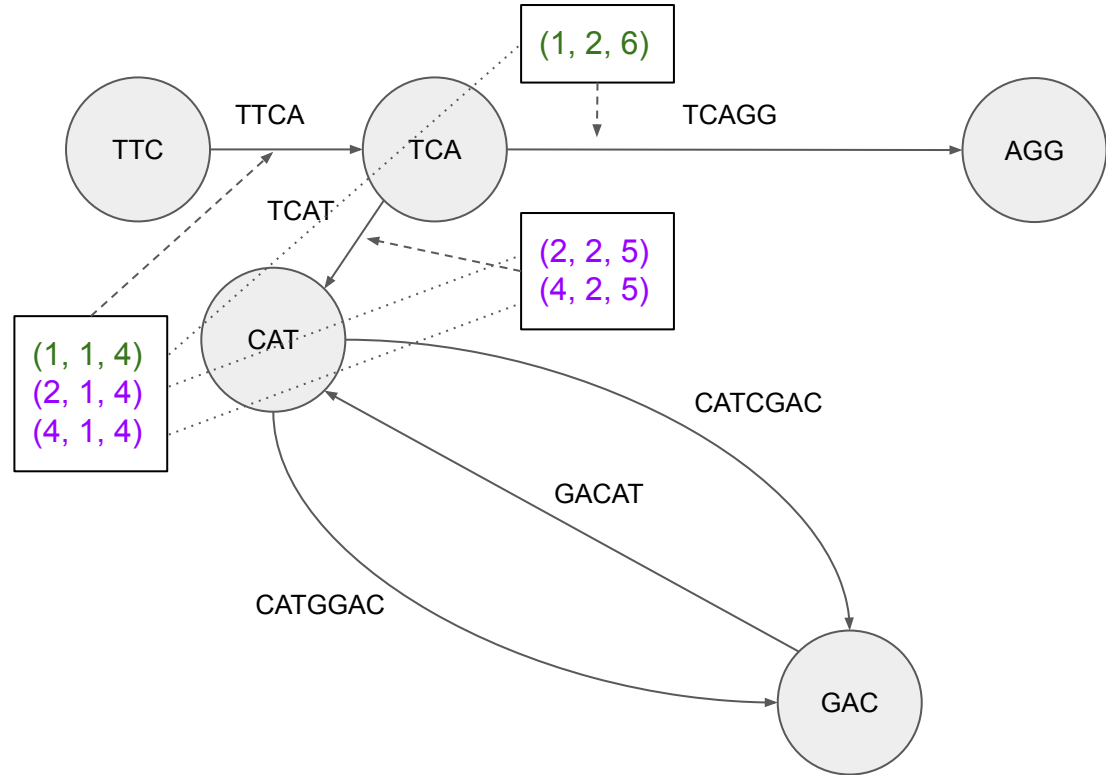
## 2b) Graph reductions: forks



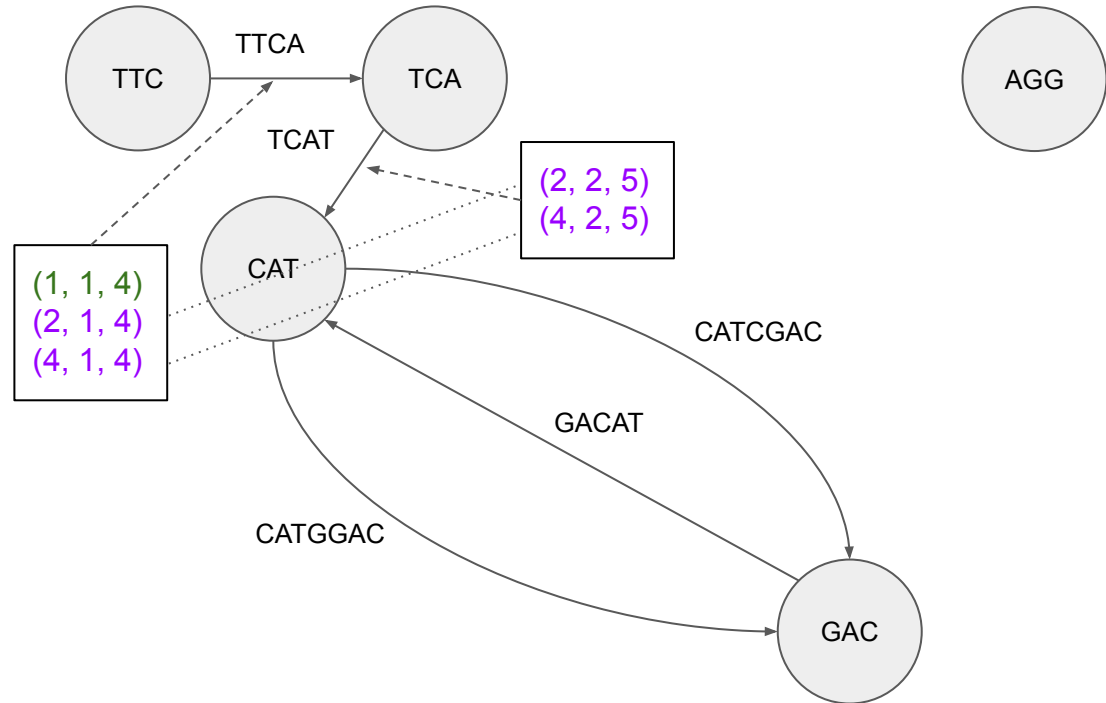
## 2b) Graph reductions: forks



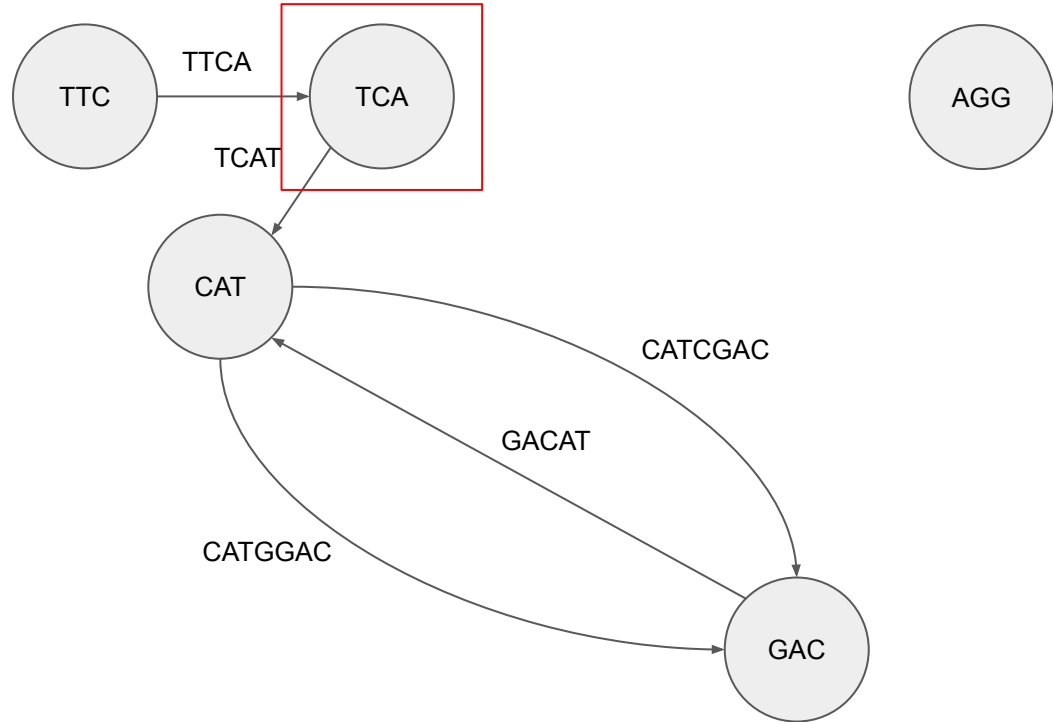
## 2b) Graph reductions: forks



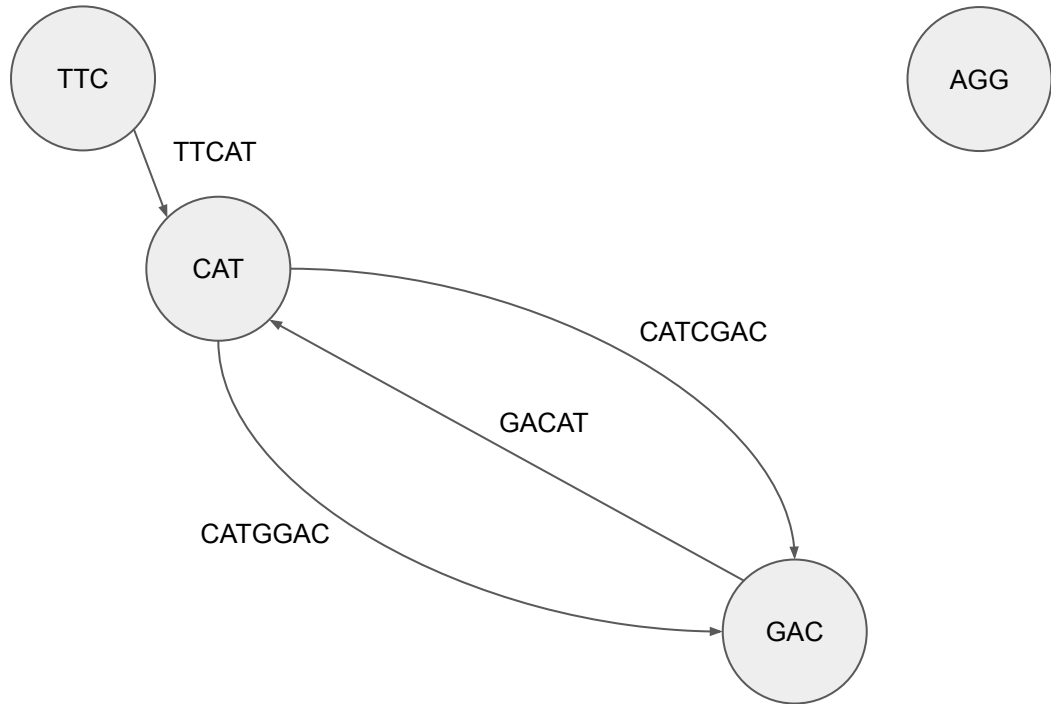
## 2b) Graph reductions: forks



## 2a) Graph reductions: singletons



## 2a) Graph reductions: singletons



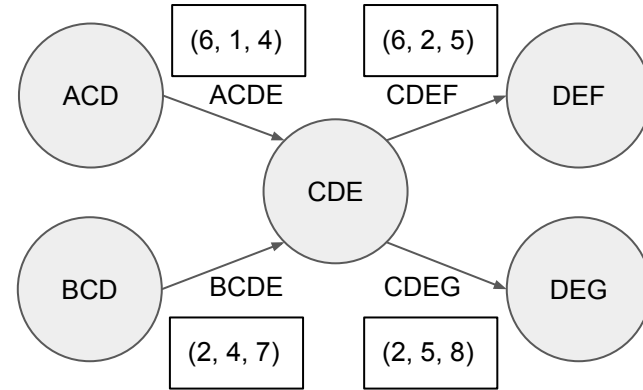


## 2c) Graph reductions: crosses

Crosses are nodes with indegree  $> 1$  and outdegree  $> 1$  (i.e., a node  $v$  with incoming edges  $[u_1, v]$ ,  $[u_2, v]$ , ... and outgoing edges  $[v, w_1]$ ,  $[v, w_2]$ , ...).

We again apply a heuristic approach to resolve this feature, which may be the result of sequencing errors or repetitive regions.

If there are pairs of edges which have continuing occurrences with each other, merge these pairs of edges.\*



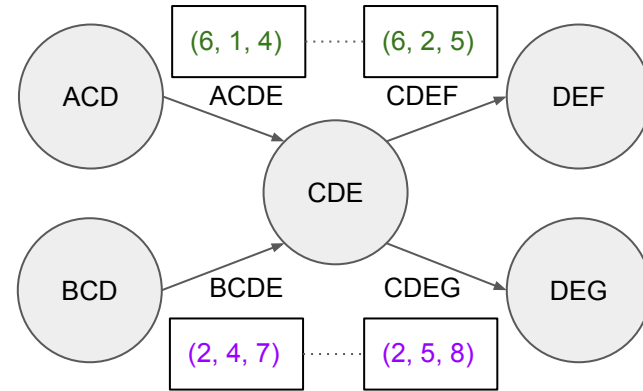
Idury and Waterman formalize this by applying the overlap test in both directions (i.e., merge  $[u_1, v]$  and  $[v, w_1]$  if  $[v, w_1]$  passes the overlap test for  $[u_1, v]$  and  $[u_1, v]$  passes the overlap test for  $[v, w_1]$ ).

## 2c) Graph reductions: crosses

Crosses are nodes with indegree  $> 1$  and outdegree  $> 1$  (i.e., a node  $v$  with incoming edges  $[u_1, v]$ ,  $[u_2, v]$ , ... and outgoing edges  $[v, w_1]$ ,  $[v, w_2]$ , ...).

We again apply a heuristic approach to resolve this feature, which may be the result of sequencing errors or repetitive regions.

If there are pairs of edges which have continuing occurrences with each other, merge these pairs of edges.\*



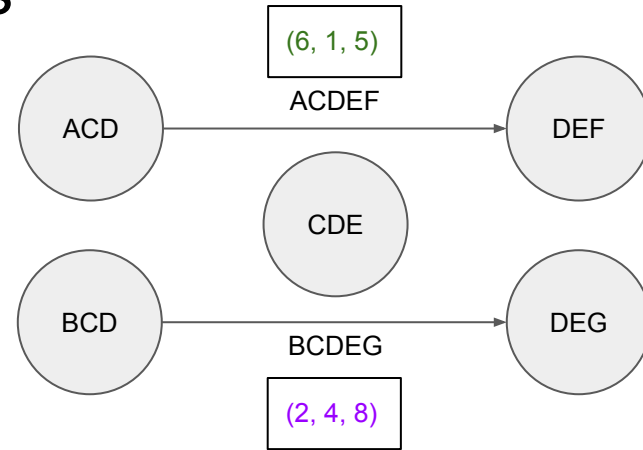
Idury and Waterman formalize this by applying the overlap test in both directions (i.e., merge  $[u_1, v]$  and  $[v, w_1]$  if  $[v, w_1]$  passes the overlap test for  $[u_1, v]$  and  $[u_1, v]$  passes the overlap test for  $[v, w_1]$ ).

## 2c) Graph reductions: crosses

Crosses are nodes with indegree  $> 1$  and outdegree  $> 1$  (i.e., a node  $v$  with incoming edges  $[u_1, v]$ ,  $[u_2, v]$ , ... and outgoing edges  $[v, w_1]$ ,  $[v, w_2]$ , ...).

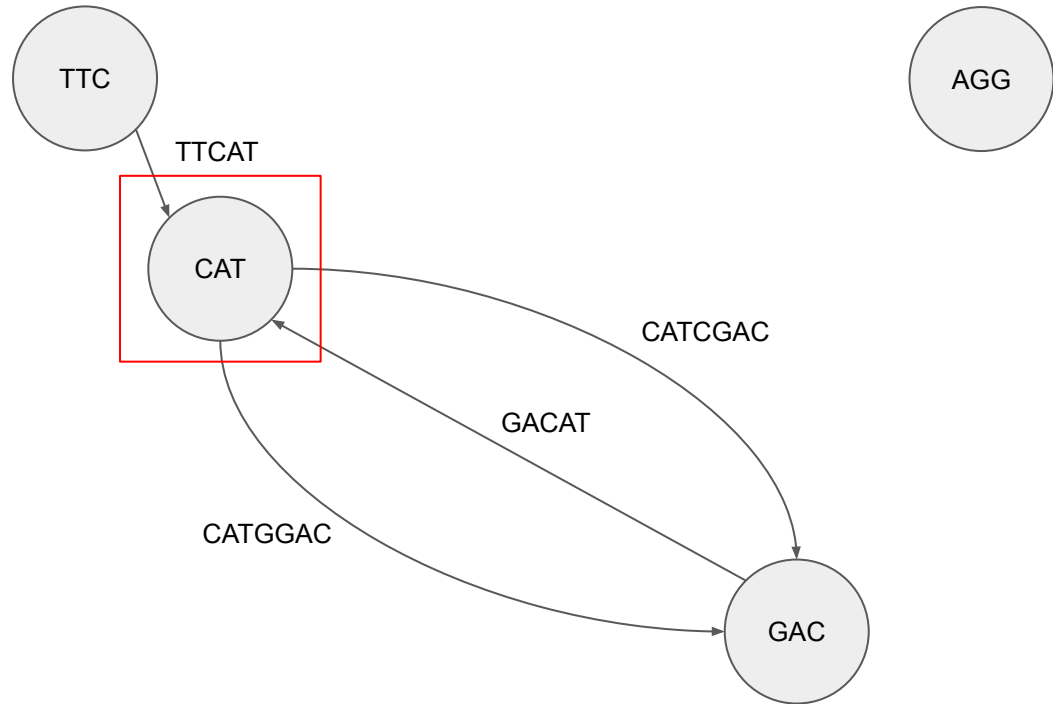
We again apply a heuristic approach to resolve this feature, which may be the result of sequencing errors or repetitive regions.

If there are pairs of edges which have continuing occurrences with each other, merge these pairs of edges.\*

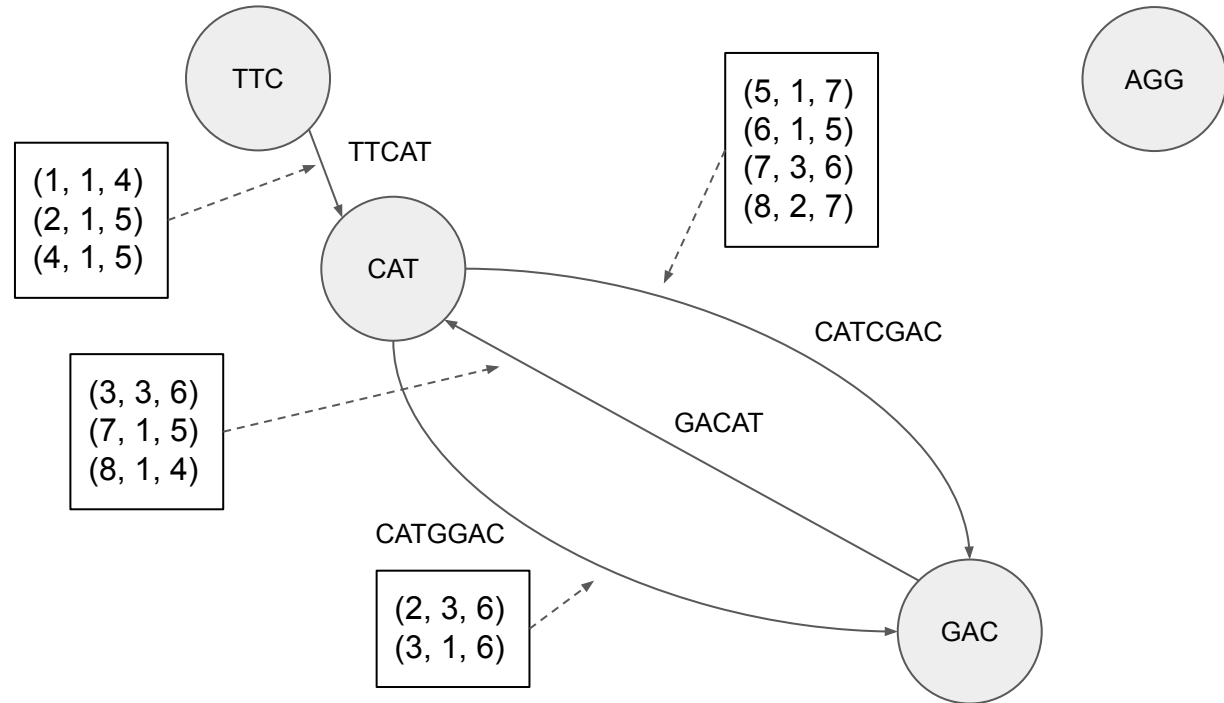


Idury and Waterman formalize this by applying the overlap test in both directions (i.e., merge  $[u_1, v]$  and  $[v, w_1]$  if  $[v, w_1]$  passes the overlap test for  $[u_1, v]$  and  $[u_1, v]$  passes the overlap test for  $[v, w_1]$ ).

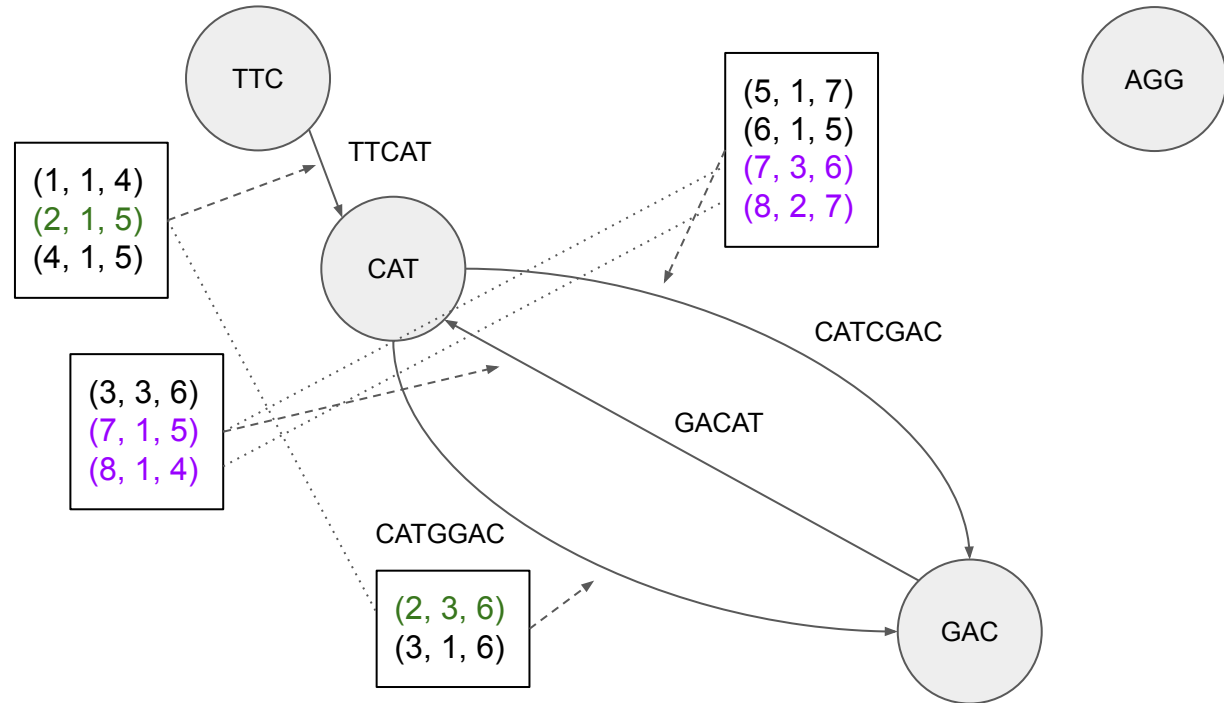
## 2c) Graph reductions: crosses



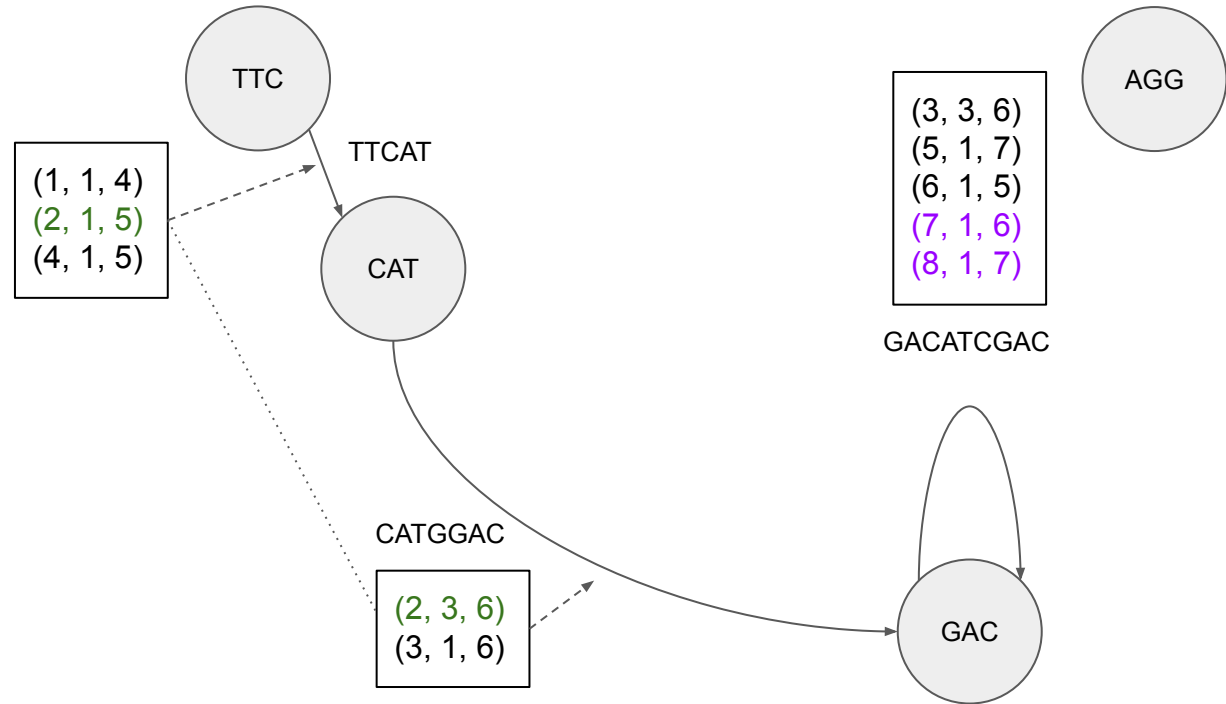
## 2c) Graph reductions: crosses



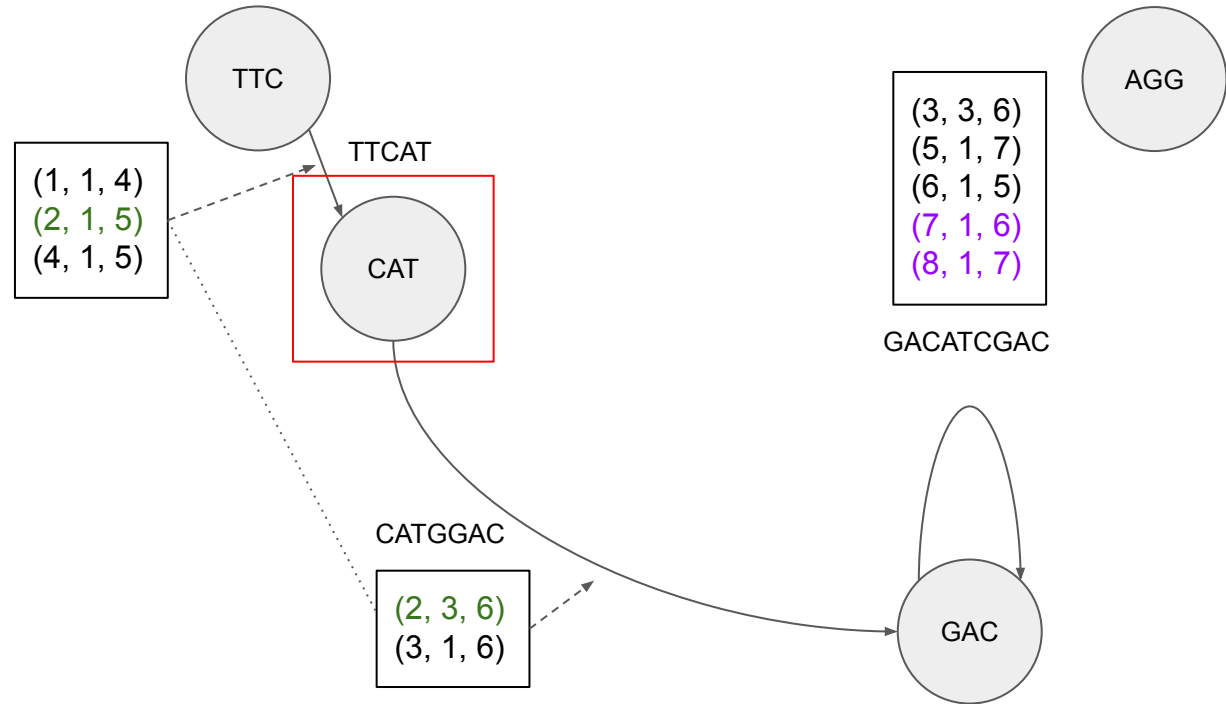
## 2c) Graph reductions: crosses



## 2c) Graph reductions: crosses

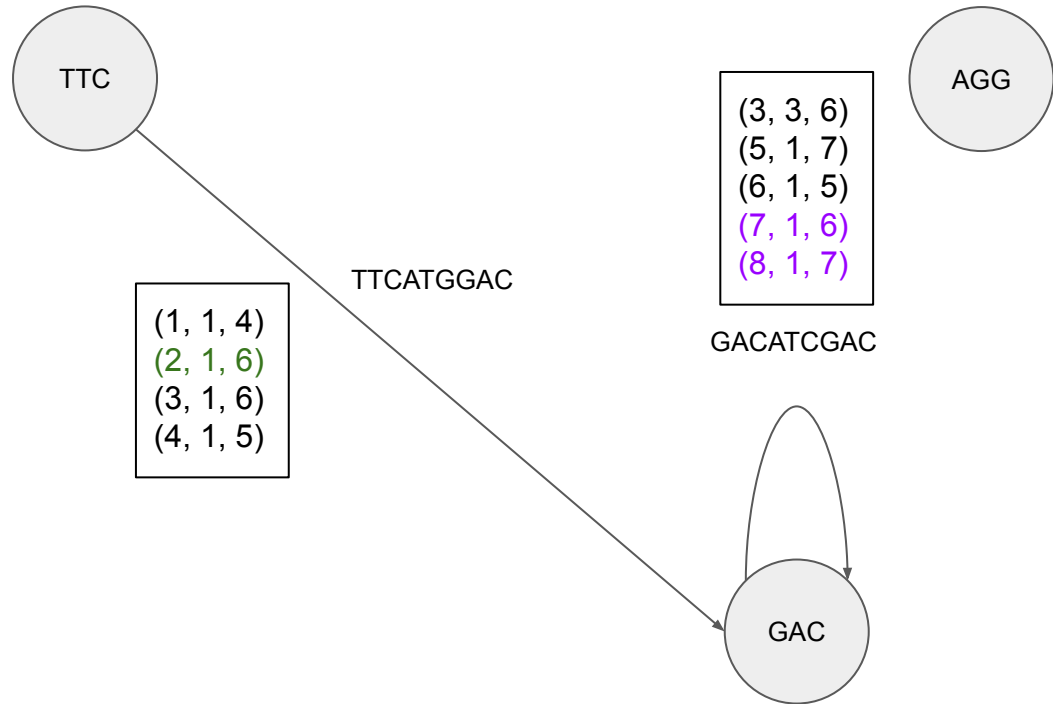


## 2a) Graph reductions: singletons





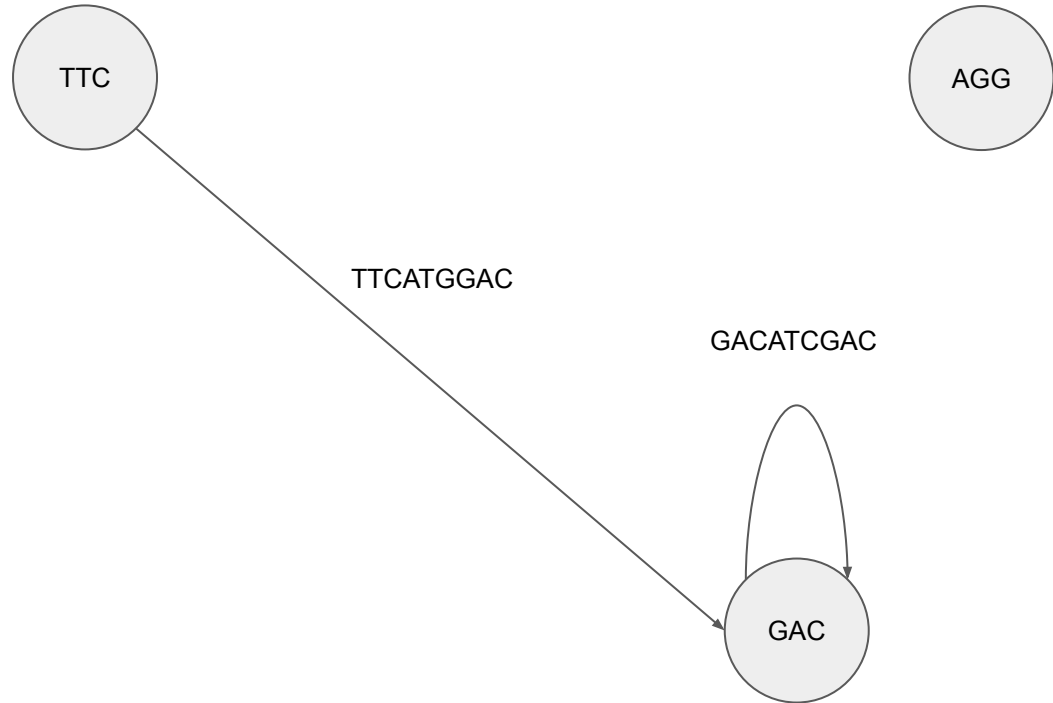
## 2a) Graph reductions: singletons



## 2) Irreducible sequence graph

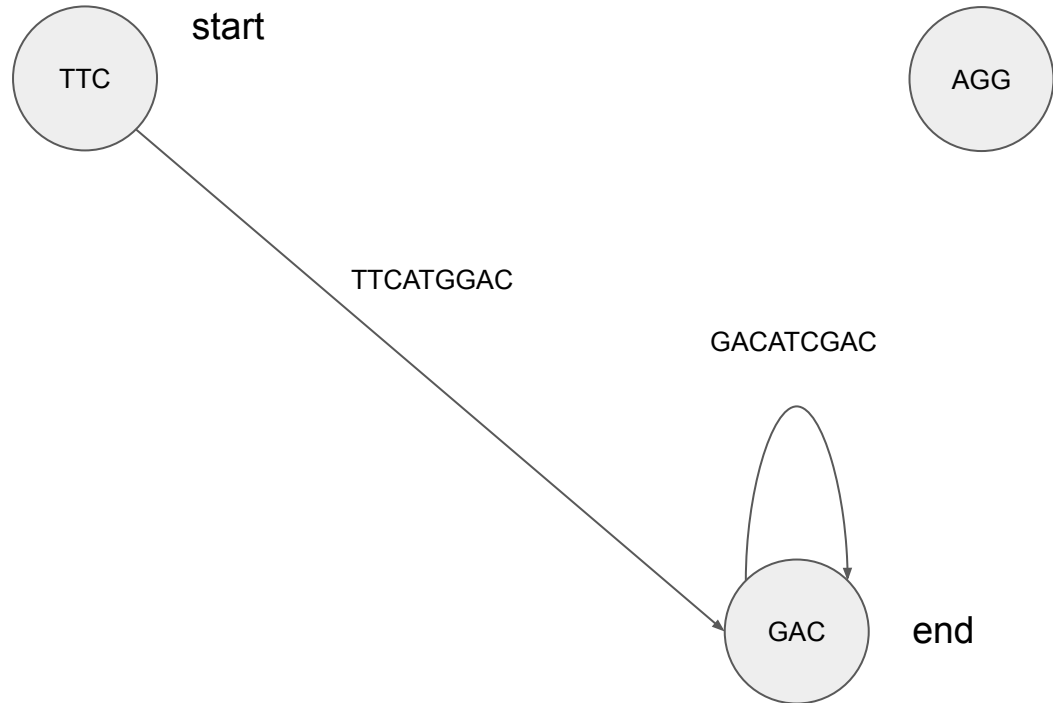
The graph has now been fully reduced, and we can search for an Eulerian path.

You may notice that the reductions accumulate lots of occurrence tuples within the graph edges. These remaining edges are referred to as “super edges,” and in general, we assign more weight (confidence) to edge labels with more occurrences.



### 3) Perform an Eulerian tour

We identify the start node as having  $\text{outdegree} - \text{indegree} = 1$  and the terminal node as having  $\text{indegree} - \text{outdegree} = 1$ .

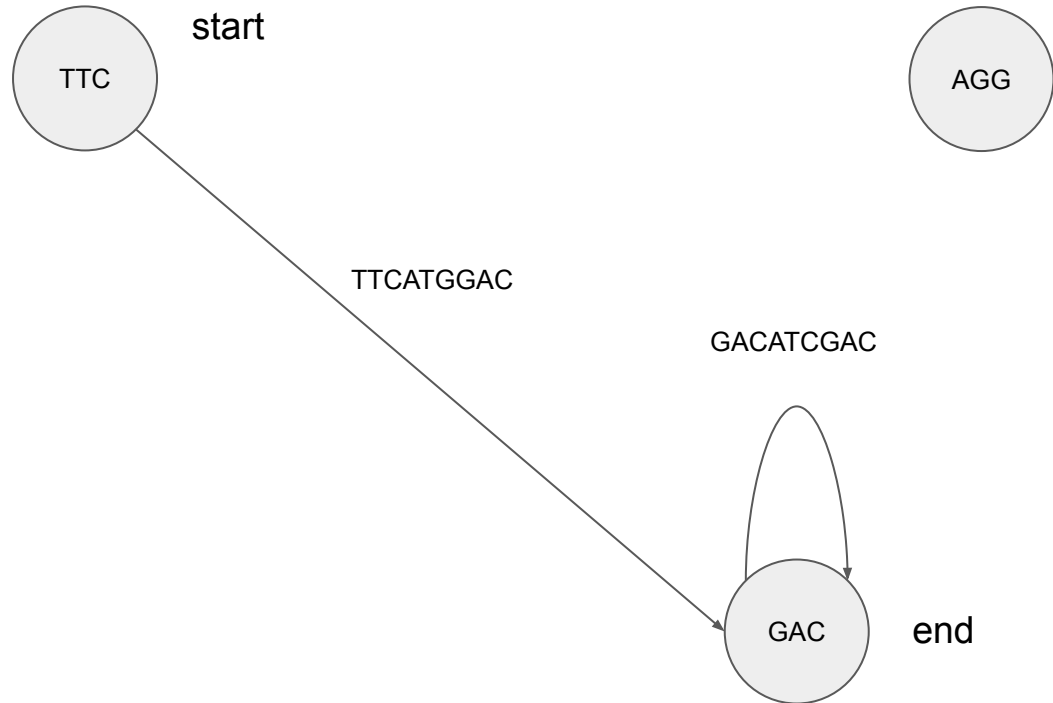


### 3) Perform an Eulerian tour

We identify the start node as having  $\text{outdegree} - \text{indegree} = 1$  and the terminal node as having  $\text{indegree} - \text{outdegree} = 1$ .

Beginning with the start node, follow outgoing edges until all edges are used once and the terminal node is reached. Infer the sequence by merging edge labels along the path taken.

If there are multiple outgoing edges to follow, choose edges based on the continuity of occurrences and edge weights.

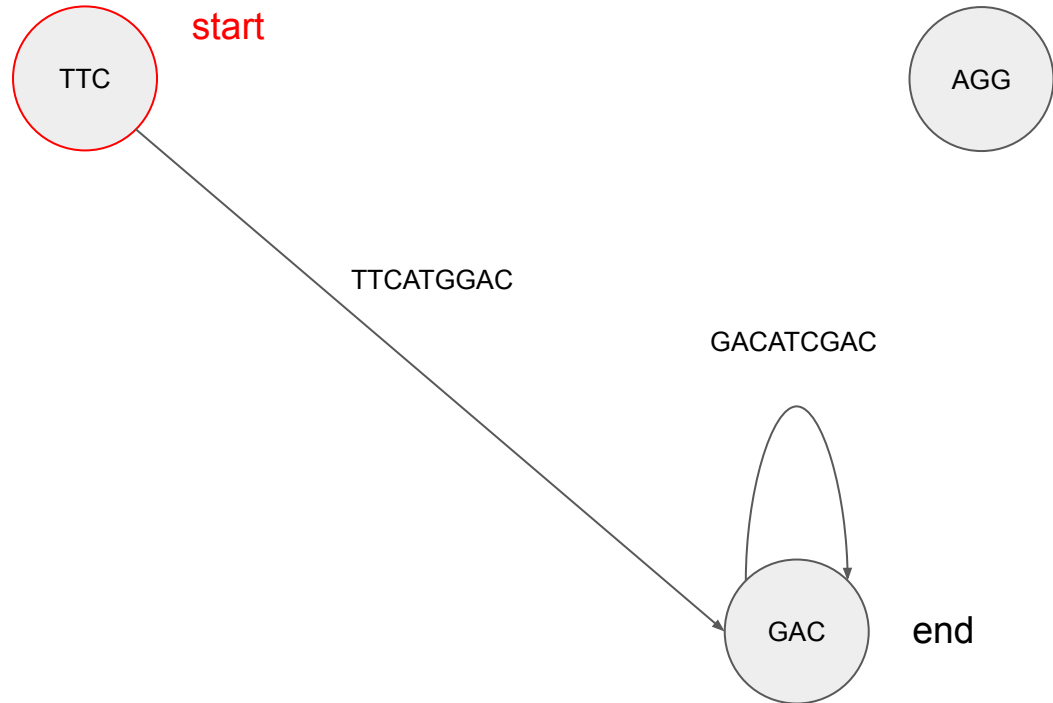


### 3) Perform an Eulerian tour

We identify the start node as having  $\text{outdegree} - \text{indegree} = 1$  and the terminal node as having  $\text{indegree} - \text{outdegree} = 1$ .

Beginning with the start node, follow outgoing edges until all edges are used once and the terminal node is reached. Infer the sequence by merging edge labels along the path taken.

If there are multiple outgoing edges to follow, choose edges based on the continuity of occurrences and edge weights.

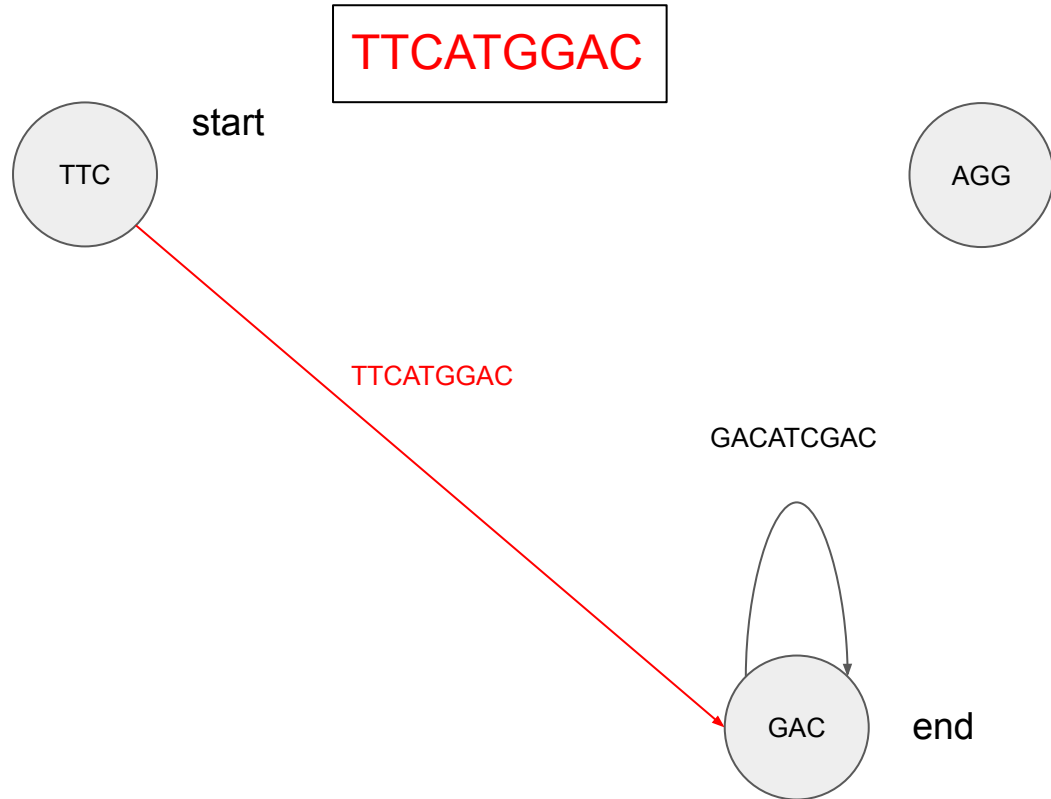


### 3) Perform an Eulerian tour

We identify the start node as having  $\text{outdegree} - \text{indegree} = 1$  and the terminal node as having  $\text{indegree} - \text{outdegree} = 1$ .

Beginning with the start node, follow outgoing edges until all edges are used once and the terminal node is reached. Infer the sequence by merging edge labels along the path taken.

If there are multiple outgoing edges to follow, choose edges based on the continuity of occurrences and edge weights.

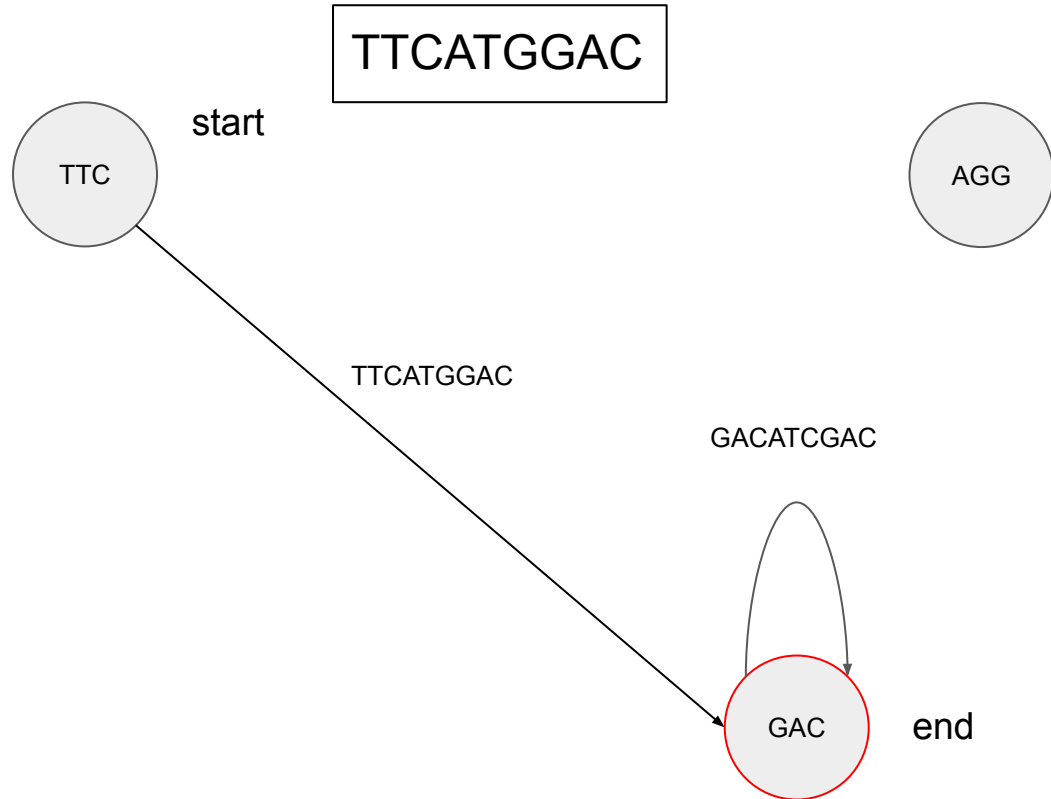


### 3) Perform an Eulerian tour

We identify the start node as having  $\text{outdegree} - \text{indegree} = 1$  and the terminal node as having  $\text{indegree} - \text{outdegree} = 1$ .

Beginning with the start node, follow outgoing edges until all edges are used once and the terminal node is reached. Infer the sequence by merging edge labels along the path taken.

If there are multiple outgoing edges to follow, choose edges based on the continuity of occurrences and edge weights.

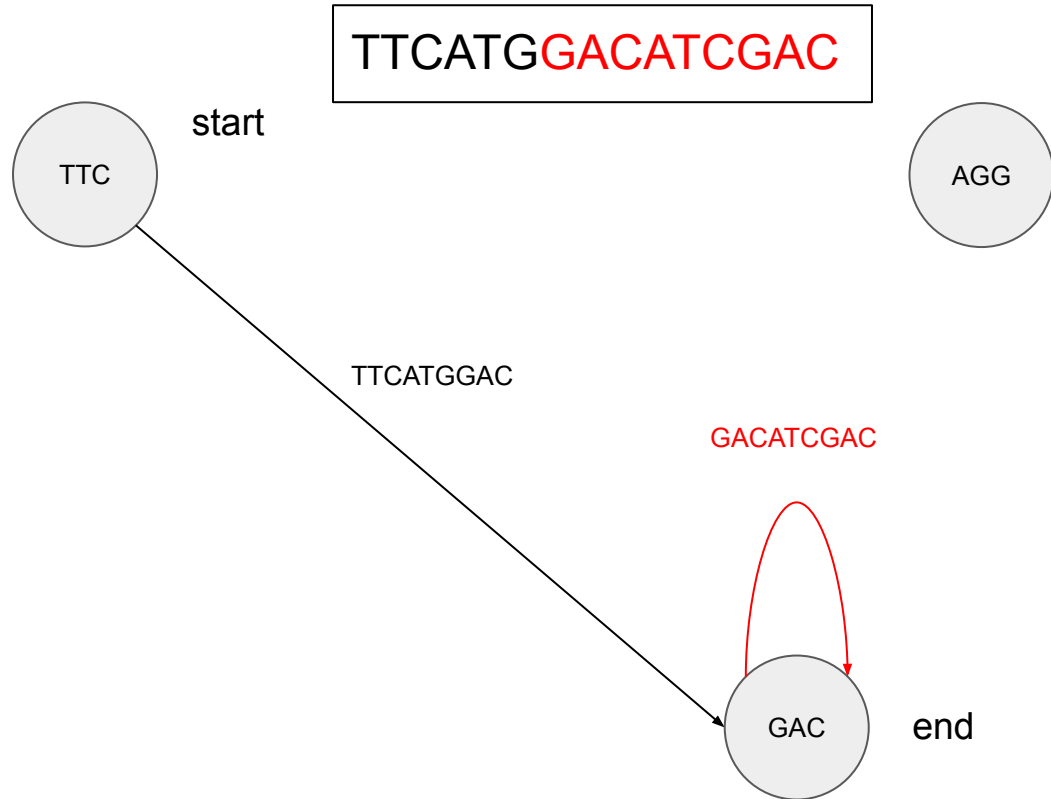


### 3) Perform an Eulerian tour

We identify the start node as having  $\text{outdegree} - \text{indegree} = 1$  and the terminal node as having  $\text{indegree} - \text{outdegree} = 1$ .

Beginning with the start node, follow outgoing edges until all edges are used once and the terminal node is reached. Infer the sequence by merging edge labels along the path taken.

If there are multiple outgoing edges to follow, choose edges based on the continuity of occurrences and edge weights.



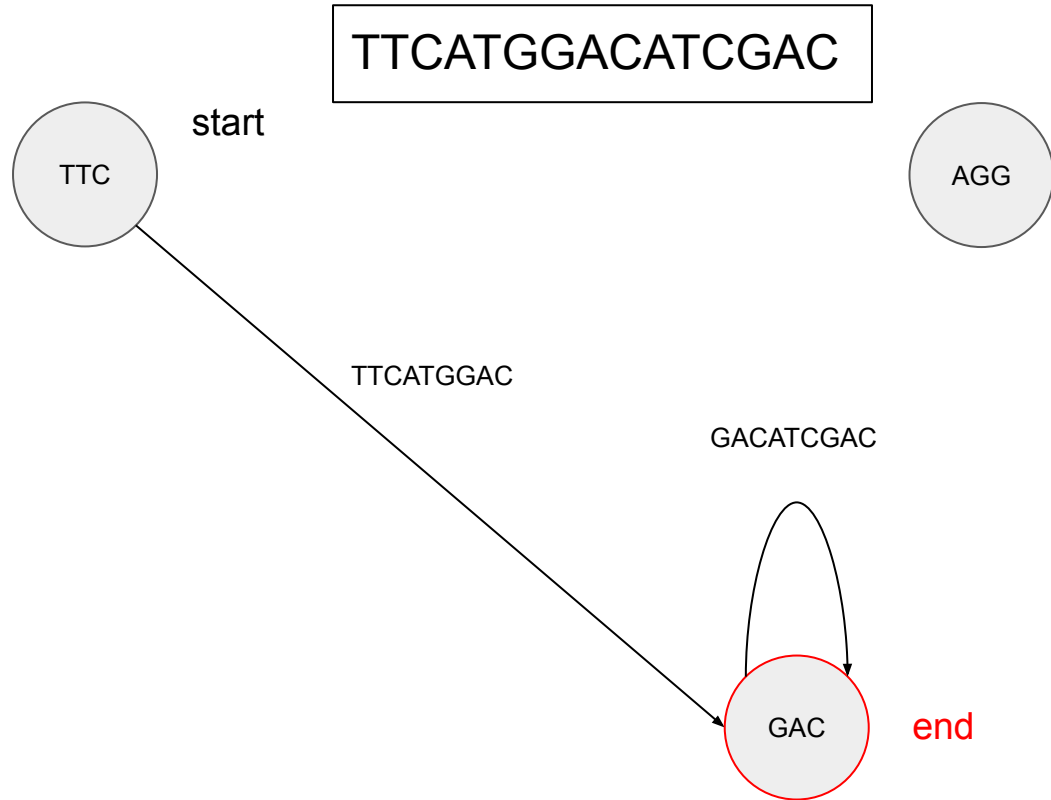


### 3) Perform an Eulerian tour

We identify the start node as having  $\text{outdegree} - \text{indegree} = 1$  and the terminal node as having  $\text{indegree} - \text{outdegree} = 1$ .

Beginning with the start node, follow outgoing edges until all edges are used once and the terminal node is reached. Infer the sequence by merging edge labels along the path taken.

If there are multiple outgoing edges to follow, choose edges based on the continuity of occurrences and edge weights.



## 4) Align the reads to the inferred sequence

$f_1$	TTCAGG
$f_2$	TTCATGG
$f_3$	ATGGACA
$f_4$	TTCAT
$f_5$	CATCGAC
$f_6$	TCGAC
$f_7$	GACATC
$f_8$	ACATCGA

TTCATGGACATCGAC

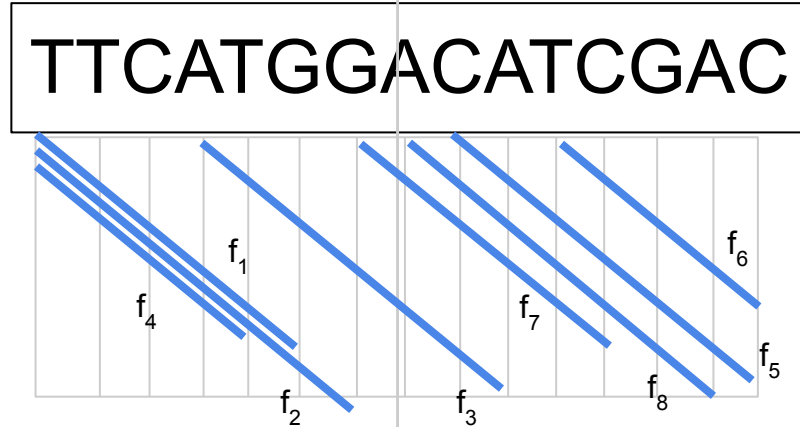
First, we apply hashing methods to identify where each fragment might align well to the sequence.

This will produce “candidate diagonals.”

We can then perform alignment along those diagonals, which is more efficient than using the entire edit graph.

#### 4) Align the reads to the inferred sequence

$f_1$  TTCAGG  
 $f_2$  TTCATGG  
 $f_3$  ATGGACA  
 $f_4$  TTCAT  
 $f_5$  CATCGAC  
 $f_6$  TCGAC  
 $f_7$  GACATC  
 $f_8$  ACATCGA

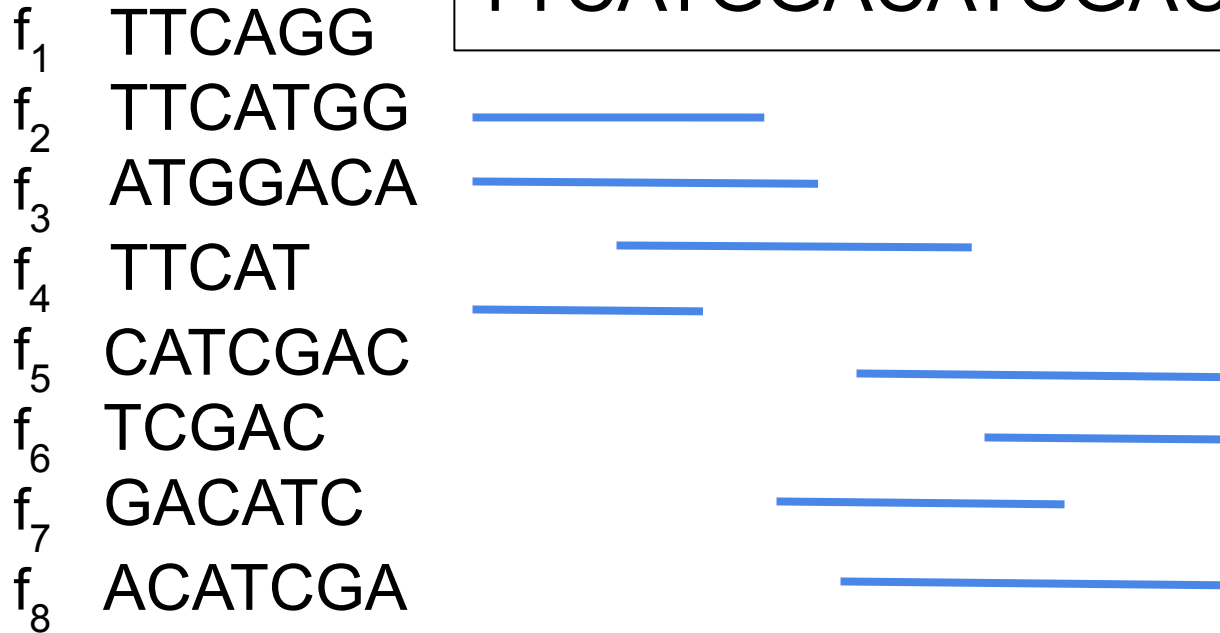


First, we apply hashing methods to identify where each fragment might align well to the sequence.

This will produce “candidate diagonals.”

We can then perform alignment along those diagonals, which is more efficient than using the entire edit graph.

## 4) Align the reads to the inferred sequence



First, we apply hashing methods to identify where each fragment might align well to the sequence.

This will produce “candidate diagonals.”

We can then perform alignment along those diagonals, which is more efficient than using the entire edit graph.

4) Align the reads to the inferred sequence

TTCATGGACATCGAC

TTCAGG

TTCATGG

ATGGACA

TTCAT

CATCGAC

TCGAC

GACATC

ACATCGA

# Statistics of Sequence Graphs

We have seen that graph reductions greatly help to limit the size of the sequence graph during assembly. But how many reductions can we expect to make? What do we expect the graph structure to look like?

Define the following:


- $k$  = tuple size, edge label length
- $L$  = length of original sequence
- $N$  = number of fragments (reads)
- $\ell$  = average fragment length
- $c$  = mean depth of coverage
- $T = N(\ell - k + 2) =$  number of  $(k-1)$  regions
- $r$  = error rate

Assume:

1. Errors are uniformly distributed over fragments and the length of each fragment
2. Error rate  $r$  is small
3. For any position  $i$  of the original sequence, the number of fragments covering  $i \dots (i+k-2)$  is a Poisson random variable
4. There are no repeats of length at least  $k$
5. The only sequencing errors are substitutions

# Statistics of Sequence Graphs

$$\text{Let } L' = L - k + 2 \text{ and } R = 1 - (1 - r)^{(k-1)}$$



This is the number of  $(k-1)$ -mers  
in the original sequence.

This is the  $(k-1)$ -mer error rate.

Then we have the following results:

1. The expected number of vertices  $\mathbb{E}(|V|) = RT + [1 - e^{-c(1-R)}]L'$ .
2. The expected number of singletons  $\mathbb{E}(|S|) = RT + e^{-c(1-R)}[e^{c(1-R)(1-r)^2} + c(1-R)r(2-r) - 1]L'$ .
3. The expected number of forks  $\mathbb{E}(|F|) = 2e^{-c(1-R)}[e^{c(1-R)(1-r)} - e^{c(1-R)(1-r)^2} - c(1-R)r(1-r)]L'$ .

# Statistics of Sequence Graphs: vertices

Here we will briefly outline the proof of (1):

$$\textit{The expected number of vertices } \mathbb{E}(|V|) = RT + [1 - e^{-c(1-R)}]L'.$$

Vertices in the sequence graph can be classified as true (if they are found in the original sequence) or false (if they are generated from a sequencing error).

We can compute the number of expected vertices as the sum of the expected number of true vertices and the expected number of false vertices.

If  $T$  is the number of  $(k-1)$ -mers in our read set, and  $R$  is the false  $(k-1)$ -mer rate, then the expected number of false vertices is  $RT$ .



# Statistics of Sequence Graphs: vertices

To find the expected number of true vertices, we consider that the number of true vertices is equivalent to the number of positions such that at least one fragment has no sequencing errors.

Recall that the depth at a particular position (number of reads covering that position) is a Poisson variable with mean  $c \Leftrightarrow X \sim \text{Poisson}(c)$ . So we have:

$$\mathbb{E}(\text{True}) = L' \sum_{i=1}^{\infty} (1 - R^i) \mathbb{P}(X = i)$$

Number of (k-1)-mers in the original sequence

Expectation over all possible values for  $X \sim \text{Poisson}$

Probability that at least 1 out of  $i$  reads is correctly sequenced

Probability that  $i$  reads cover the (k-1)-length region

## Statistics of Sequence Graphs: vertices

$$\begin{aligned}\mathbb{E}(True) &= L' \sum_{i=1}^{\infty} (1 - R^i) \mathbb{P}(X = i) \\ &= L' \sum_{i=1}^{\infty} \left( \frac{e^{-c} c^i}{i!} - \frac{e^{-c} (cR)^i}{i!} \right) \\ &= L' (1 - e^{-c(1-R)}).\end{aligned}$$

pmf of Poisson:  $\Pr(X=k) = \frac{\lambda^k e^{-\lambda}}{k!}$

using Taylor expansion for e:  $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$

Summing the number of false vertices and true vertices produces:

*The expected number of vertices  $\mathbb{E}(|V|) = RT + [1 - e^{-c(1-R)}]L'$ .*

What did we expect from our example sequence graph?

- $k = 4$
- $L = 15$
- $N = 8$
- $\ell = 50/8 = 6.25$
- $c = 50/15 = 3.33$
- $T = 32$
- $r = 1/50 = 0.02$

TTCATGGACATCGAC

TTCAGG  
TTCATGG  
ATGGACA  
TTCAT  
CATCGAC  
TCGAC  
GACATC  
ACATCGA

# What did we expect from our example sequence graph?

1. The expected number of vertices  $\mathbb{E}(|V|) = RT + [1 - e^{-c(1-R)}]L'$ .
2. The expected number of singletons  $\mathbb{E}(|S|) = RT + e^{-c(1-R)}[e^{c(1-R)(1-r)^2} + c(1-R)r(2-r) - 1]L'$ .
3. The expected number of forks  $\mathbb{E}(|F|) = 2e^{-c(1-R)}[e^{c(1-R)(1-r)} - e^{c(1-R)(1-r)^2} - c(1-R)r(1-r)]L'$ .

- $k = 4$
- $L = 15$
- $N = 8$
- $\ell = 50/8 = 6.25$
- $c = 50/15 = 3.33$
- $T = 32$
- $r = 1/50 = 0.02$

$$\mathbb{E}(|V|) \approx 14.318$$

$$\mathbb{E}(|S|) \approx 12.869$$

$$\mathbb{E}(|F|) \approx 1.387$$

What did we expect from our example sequence graph?

$$E(|V|) \approx 14.318$$

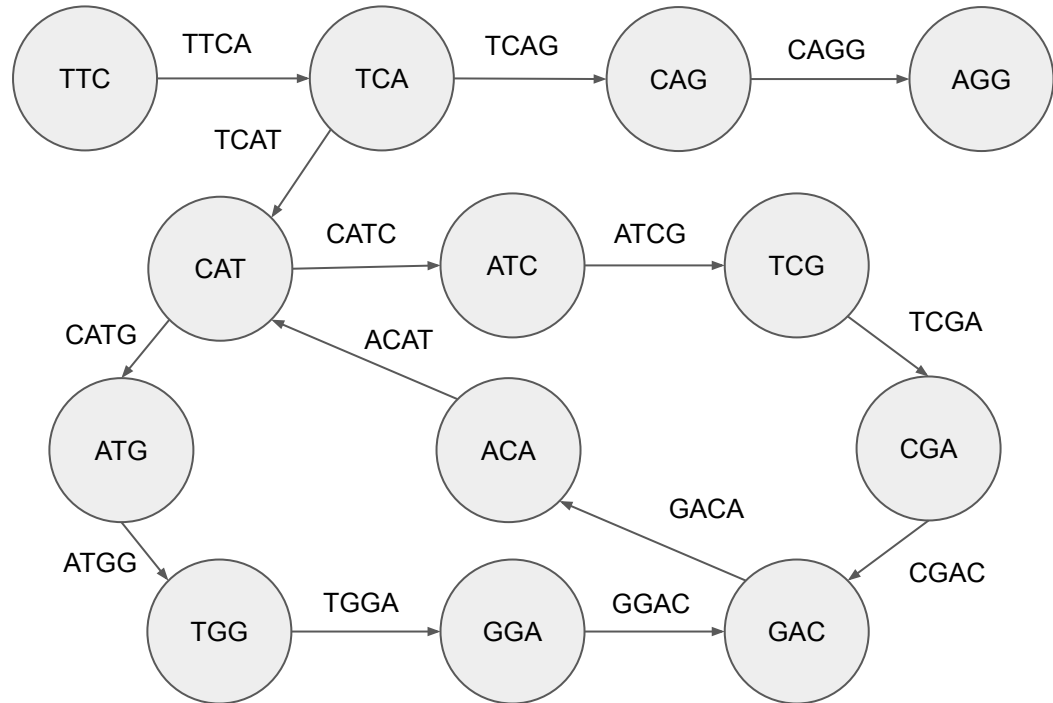
$$E(|S|) \approx 12.869$$

$$E(|F|) \approx 1.387$$

$$|V| = 13$$

$$|S| = 8$$

$$|F| = 1$$



# What did we expect from our example sequence graph?

$$E(|V|) \approx 14.318$$

$$E(|S|) \approx 12.869$$

$$E(|F|) \approx 1.387$$



This also predicts that we could simplify the graph to 1-3 nodes

$$|V| = 13$$

$$|S| = 8$$

$$|F| = 1$$



# Practical considerations

- The Idury-Waterman algorithm pioneered the use of De Bruijn graphs in assembly, particularly with the advent of shotgun sequencing over sequencing by hybridization
- The technical challenges associated with repeat regions, cost limitations on sequencing depth, and sequencing errors require heuristics to resolve
  - Idury and Waterman propose utilizing positional information, incorporating both **multiplicity** of k-mers and fragment **continuity** as deciding factors
- The randomness of sequencing errors and sampling reads prevents concrete upper bounds on efficiency but allow for probabilistic estimates of computational performance