II Fragment Assembly

di Matteo Sammartino, Michele Sottile e Daniele Vitale

- Introduzione al Problema
- Modelli Computazionali
- Algoritmi
- Soluzioni Reali

- Introduzione al Problema
  - Motivazioni Biologiche
  - Metodo Shotgun
  - Concetti Fondamentali
  - Complicazioni
- Modelli Computazionali
- Algoritmi
- Soluzioni Reali

Introduzione

- Problema biologico
  - Sequenziamento di un genoma
- Motivazione:
  - Sequenziamento e mappatura completa di geni e altre sequenze:
    - Sequenze trascritte e non
    - Terminatori
    - Sequenze regolatrici
    - Polimorfismi

Introduzione (2)

- Impossibile sequenziare direttamente porzioni di lunghezza maggiore di qualche centinaio di basi
- Soluzione:
  - Frammentare il DNA e sequenziare i frammenti
- Problema informatico:
  - Riassemblare i frammenti sfruttando le sovrapposizioni (*overlap*)

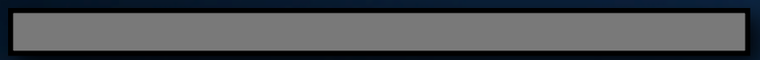
Metodo Shotgun

Metodo Shotgun

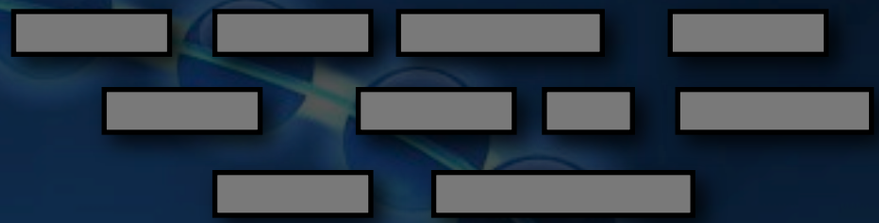
Estrazione DNA

Metodo Shotgun

Estrazione DNA

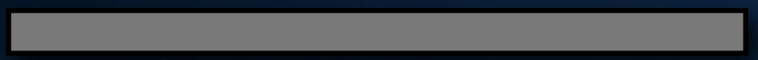


Frammentazione randomizzata

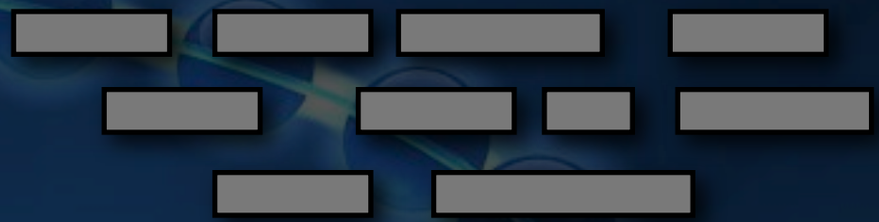


Metodo Shotgun

Estrazione DNA



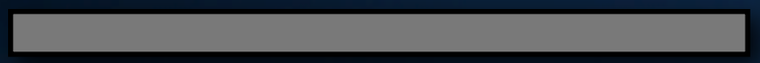
Frammentazione randomizzata



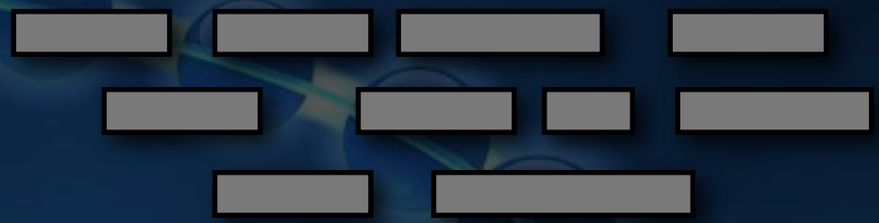
Inserzione in Vettori

Metodo Shotgun

Estrazione DNA

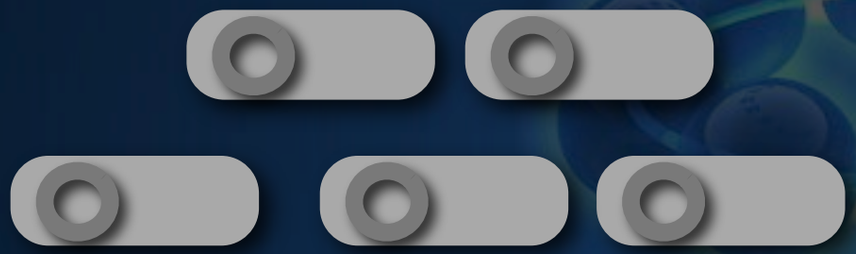


Frammentazione randomizzata



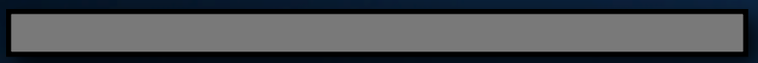
Inserzione in Vettori

Transinfezione di *E.coli*

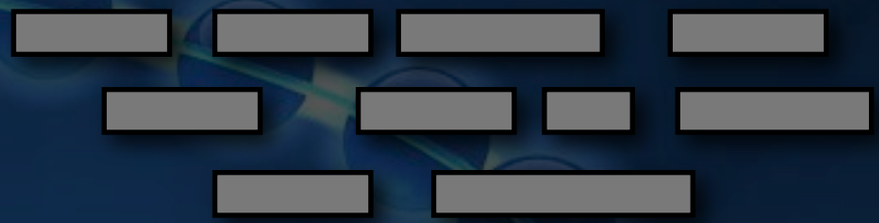


Metodo Shotgun

Estrazione DNA

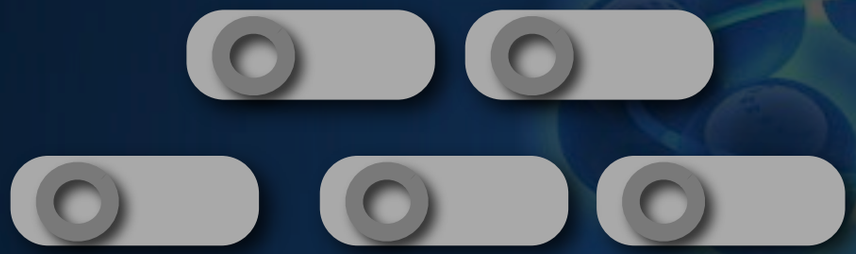


Frammentazione randomizzata



Inserzione in Vettori

Transinfezione di *E.coli*



Isolamento DNA Vettore e sequenziamento

Overlap

- Date due stringhe s_1 e s_2, un overlap è una stringa suffisso di s_1 simile ad un prefisso di s_2
- Consideriamo un overlap come il risultato di un allineamento semiglobale
- Riassemblaggio tramite overlap

Esatto	Approssimato
A C C G T - -	C A T A G T C - - -
- - C G T G C	- - T A - A C T A T

Caso ideale

- Frammenti privi di errore
- Overlap esatti

Complicazioni

- Errori
  - Mutazioni puntiformi
  - Chimere
  - Contaminazione da parte del DNA vettore
- Orientamento sconosciuto
- Regioni ripetute
- Mancanza di copertura

Mutazioni Puntiformi

Sostituzione

```
- - A C C G T - -
- - - - C G T G C
T T A C - - - - -
- T G C C G T - -
```

T T A C C G T G C

Inserzione

```
- - A C C - G T - -
- - - - C A G T G C
T T A C - - - - - -
- T A C C - G T - -
```

T T A C C - G T G C

Cancellazione

```
- - A C C G T - -
- - - - C G T G C
T T A C - - - - -
- T A C - G T - -
```

T T A C C G T G C

Consenso a maggioranza!

Chimere

- Due frammenti non contigui si compongono

Input:

ACCGT

CGTTGC

TTAC

TACCGT

TTATGC = TTA+TGC

```
- - A C C G T - -
- - - - C G T G C
T T A C - - - - -
- T A C C G T - -
---------------
T T A C C G T G C

T T A - - - T G C
```

Eliminazione in fase di preprocessing

Contaminazioni

- Frammenti contaminati da parti non provenienti dalla sequenza originaria
- Causa:
  - Errore durante la purificazione del frammento dal vettore DNA
- Soluzione:
  - Riconoscimento ed eliminazione in fase di preprocessing

Orientamento Sconosciuto

- Impossibile stabilire a quale dei due filamenti il frammento appartenga
- E' necessario tener conto del complementare di ogni frammento

Input:
CACGT
ACGT
ACTACG

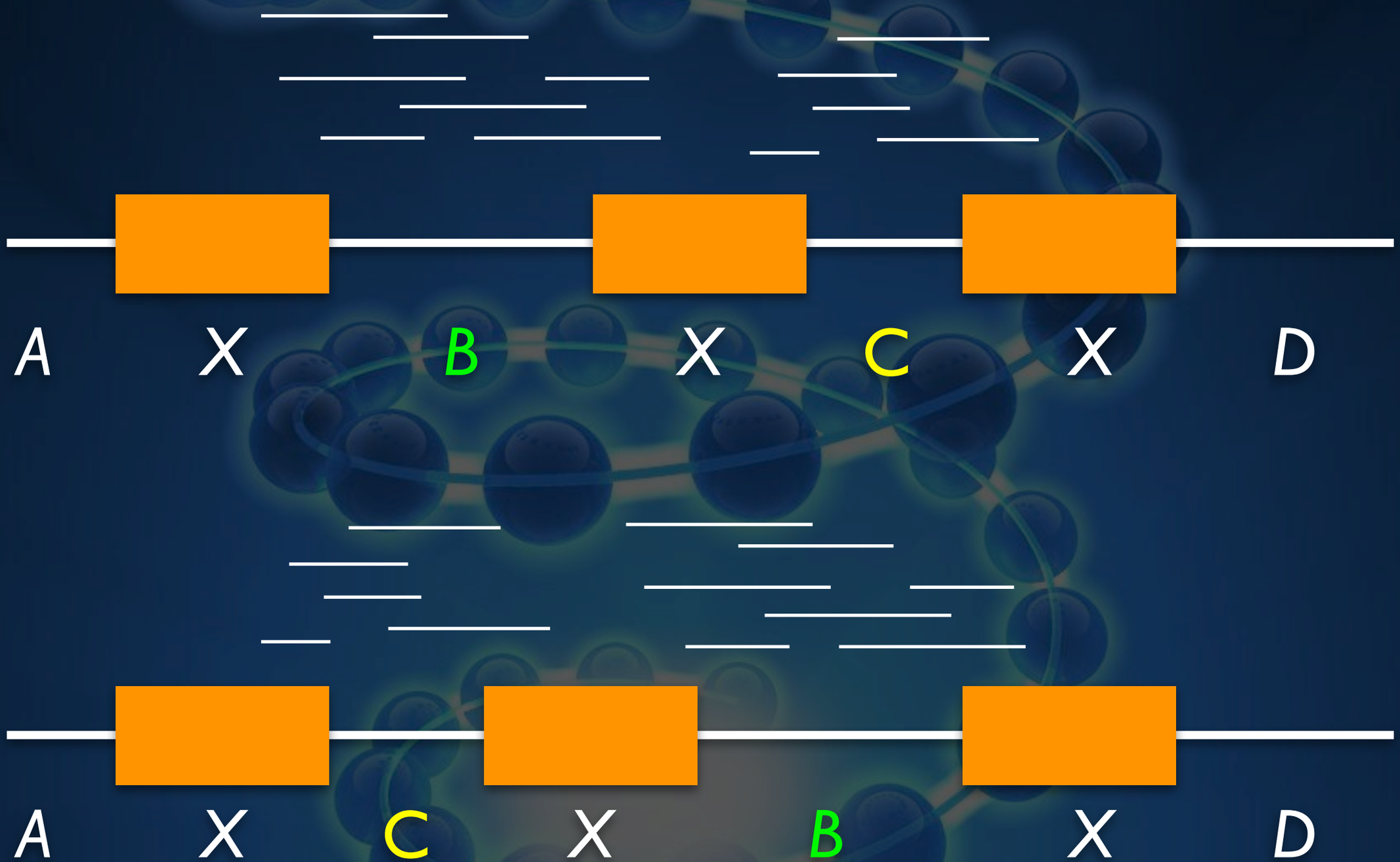
```
C A C G T - - -
- A C G T - - -
- - C G T A G T
---------------
C A C G T A G T
```

Soluzione forza bruta : 2^n confronti

Ripetizioni

- Ripetizione = regione che si ripete nella molecola target
- Ripetizioni brevi non comportano difficoltà
- Ripetizioni lunghe:
  - Due regioni simili considerate come la medesima
  - Frammento piccolo arbitrariamente posizionabile in ogni ripetizione
  - Regioni diverse affiancate da ripetizioni creano ambiguità

Esempio



Mancanza di Copertura

- Frammentazione random non garantisce copertura di ogni posizione:
  - Informazione può essere insufficiente per ricostruzione completa del target
    - Regioni disgiunte (contigs)
- Soluzione:
  - Ridondanza nella frammentazione
  - Directed Sequencing

Esempio

gap

- Introduzione al Problema
- Modelli Computazionali
- Algoritmi
- Soluzioni Reali

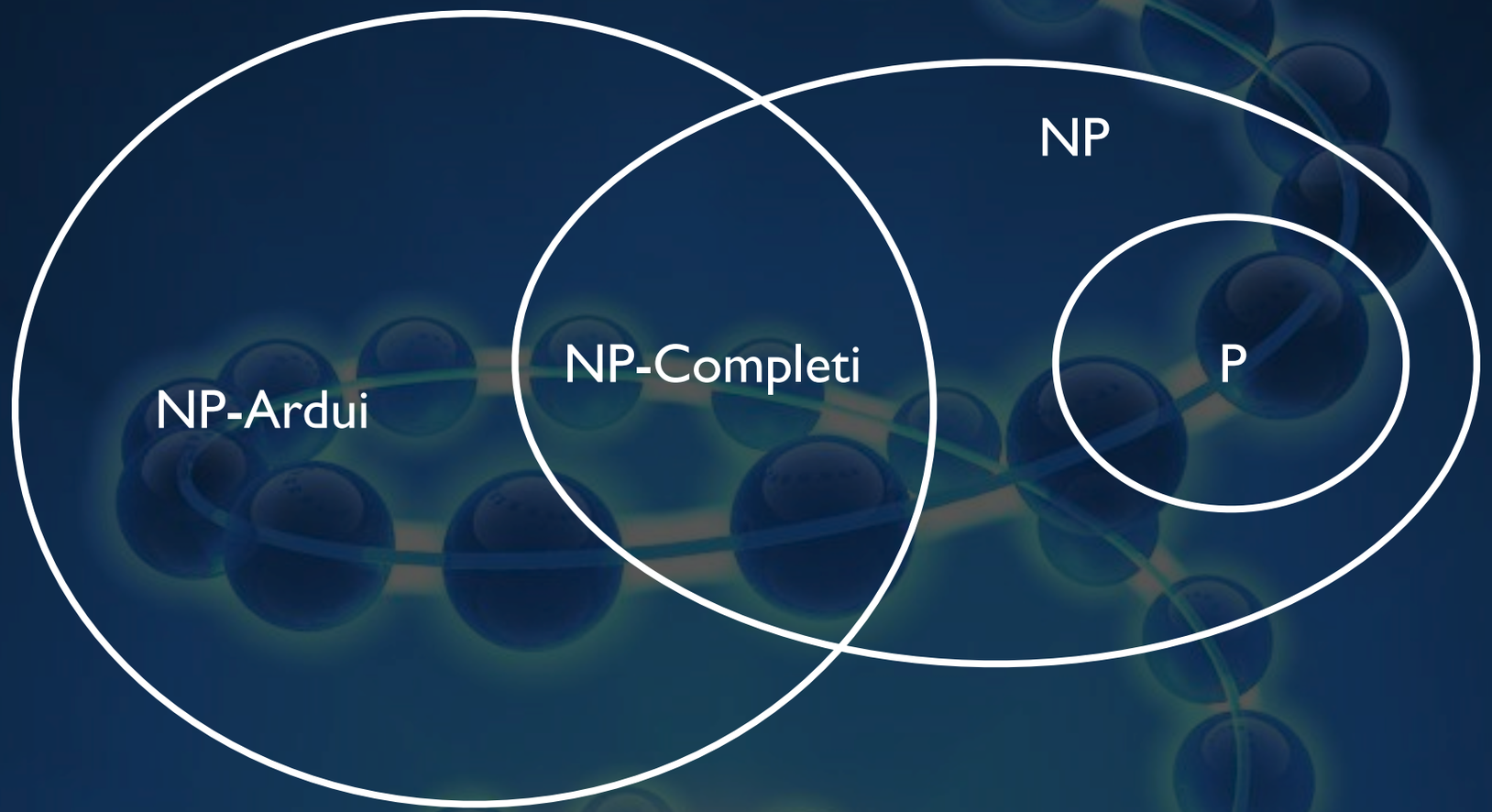
- Introduzione al Problema
- Modelli Computazionali
- Algoritmi
- Soluzioni Reali

  - SCS
  - Recostruction
  - Multicontig
  - FA vs AM

Modelli Computazionali

- Modello esatto:
  - SCS
- Modelli approssimati:
  - RECONSTRUCTION
  - MULTICONTIG
- Assunzione: no chimere e contaminazioni
- Ogni modello raffina il precedente

Classi di Complessità



Shortest Common Superstring

- Input: Collezione $\mathcal{F}$ di stringhe
- Output: La più corta stringa S tale che per ogni $f \in \mathcal{F}$, S è una superstringa di f

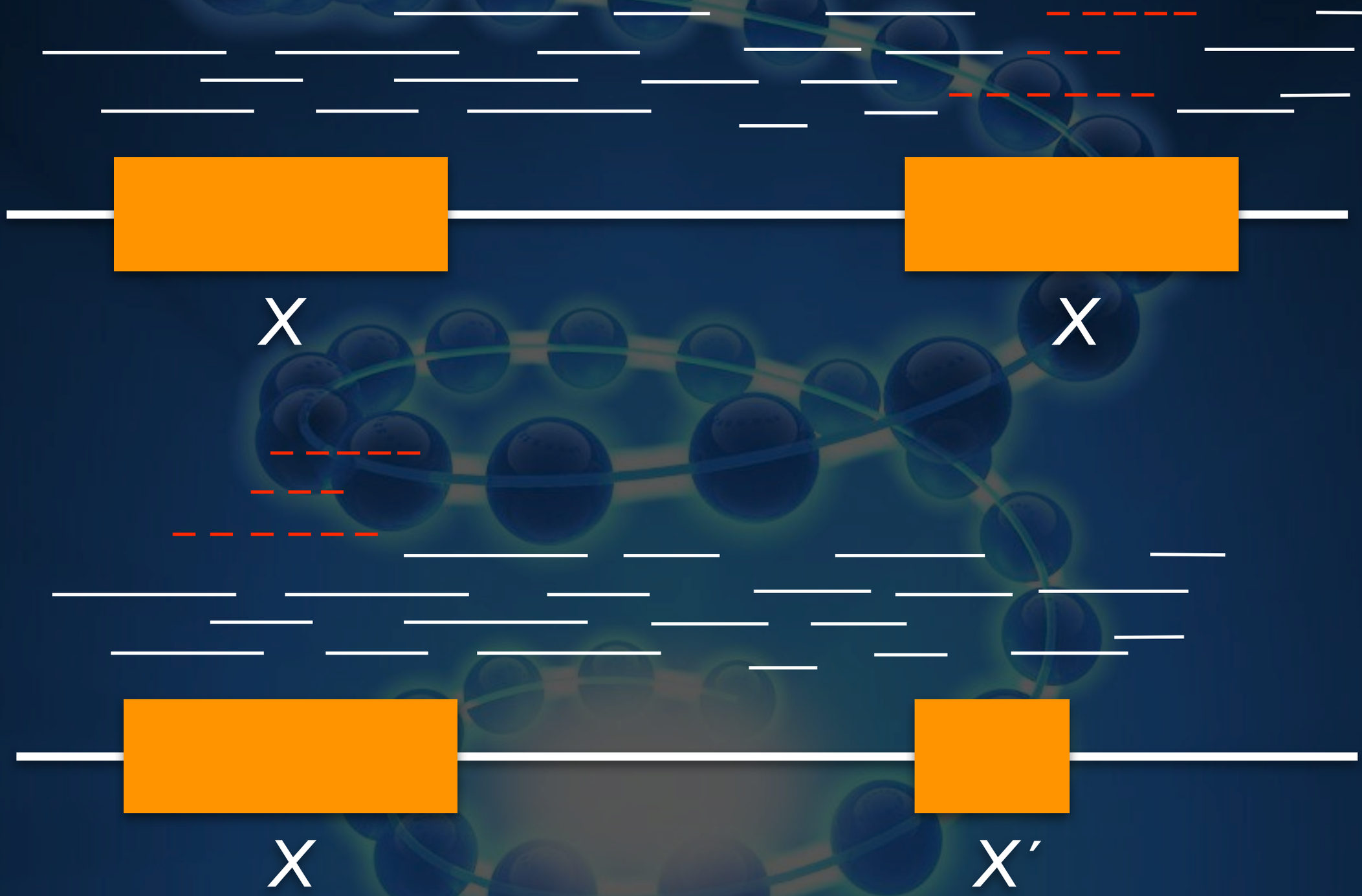
Es. $\mathcal{F} = \{ACT, CTA, AGT\}$

$S = ACTAGT$

- Non modella errori, orientamento, mancanza di copertura e ripetizioni
- NP-Arduo ma esistono algoritmi approssimati

Esempio

Esempio



Reconstruction

- Input: Una collezione di stringhe $\mathcal{F}$ ed una tolleranza agli errori $0 \leq \epsilon \leq 1$
- Output: La più corta stringa S tale che per ogni $f \in \mathcal{F}$ vale

$$\min(d_s(f, S), d_s(\overline{f}, S)) \leq \epsilon |f|$$

dove $\overline{f}$ è il complemento inverso di f e

$$d_s(a, b) = \min_{s \in S(b)} d(a, s)$$

con d = edit distance e $S(b)$ = insieme delle sottostringhe di b

Reconstruction (2)

Es.

- - - - - - GC - GATAG - - - -

CAGTCGCTGATCGTACG

$$d_s(a, b) = 2$$

- NP-Arduo (SCS caso particolare)
- Non modella ripetizioni e mancanza di copertura

Multicontig

- Arricchisce il precedente modello con la nozione di buon linkage
- t-contig = layout dove gli overlap hanno lunghezza almeno t
- S è un ϵ-consensus per un contig quando l'edit distance tra ogni frammento f del contig e la sottostringa corrispondente nel consensus S è al più $\epsilon|f|$

Multicontig (2)

- Input: Una collezione $\mathcal{F}$ di stringhe, un intero $t \geq 0$ e una tolleranza $0 \leq \epsilon \leq 1$
- Output: Sottocollezioni C_i in minimo numero, con $1 \leq i \leq k$, costituenti una partizione di $\mathcal{F}$, tali che ogni C_i ammetta un t-contig con un ϵ-consensus
- Modella errori, orientamento, mancanza di copertura e (parzialmente) le ripetizioni
- NP-Arduo

Esempio

$\mathcal{F} = \{\text{GTAC}, \text{TAATG}, \text{TGTAA}\}$

$t = 3$

```
- - T A A T G     G T A C
T G T A A - -
```

$t = 2$

```
T A A T G - - -   G T A C
- - - T G T T A
```

$t = 1$

```
T G T A A - - - - -
- - T A A T G - - -
- - - - - - G T A C
```

Fragment Assembly vs Allineamento Multiplo

- Fragment Assembly = Allineamento Multiplo +
  - Orientamento
  - Frammenti più corti dell'allineamento complessivo
  - Peso diverso ai gap interni ed esterni ai frammenti
  - Parametri di qualità

Parametri di Qualità

- Scoring:
  - Massimizzazione del *Sum of Pairs*
  - Minimizzazione dell'entropia per ogni colonna

$$E = -\sum_{c \in \{A,C,G,T,-\}} p_c \log p_c$$

dove p_c è la frequenza relativa con cui ogni carattere occorre

- Misure combinate

Parametri di Qualità (2)

- Coverage: massimizzazione del numero di frammenti che coprono ogni colonna
- Linkage: massimizzazione delle lunghezze degli overlap tra i frammenti

Es.

```
- - - - - - - A C T T T T - - - - - -
T C C G A G - - - - - - - A C G G A C
- - - - - - - A C T T T T - - - - - -
T C C G A G - - - - - - - A C G G A C
- - - - - - - A C T T T T - - - - - -
-------------------------------------
T C C G A G A C T T T T A C G G A C
```

Buon coverage ma cattivo linkage!

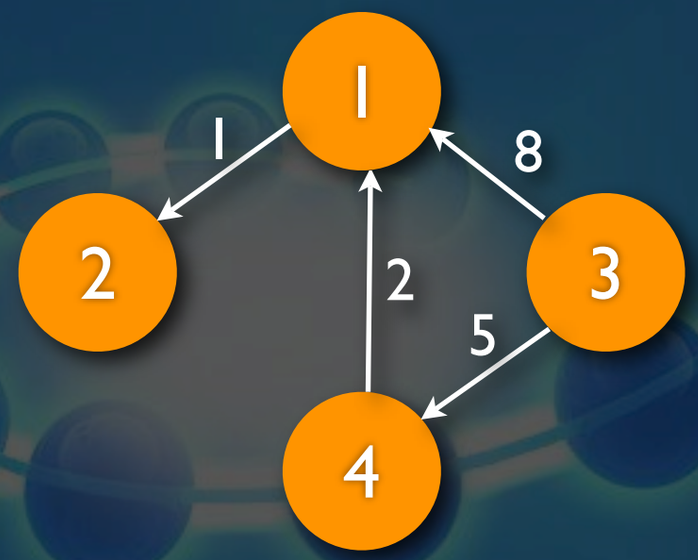
- Introduzione al Problema
- Modelli Computazionali
- Algoritmi
- Soluzioni Reali

- Introduzione al Problema
- Modelli Computazionali
- Algoritmi
- Soluzioni Reali

- Definizioni
- Cammino Hamiltoniano
- Algoritmo Greedy
- Problemi

Algoritmi

- I problemi presentati possono essere ricondotti a problemi su grafi pesati e orientati
- Un grafo G è una coppia $(\mathcal{N}, \mathcal{A})$ dove $\mathcal{N}$ è l'insieme dei nodi e $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ è l'insieme degli archi. Si ha una funzione $w : \mathcal{A} \to \mathbb{R}$ che associa un peso ad ogni arco.



Algoritmi (2)

- Un cammino P è una sequenza $n_1, n_2, \ldots, n_m$ di nodi tali che $(n_i, n_{i+1}) \in \mathcal{A}$
- Costo del cammino

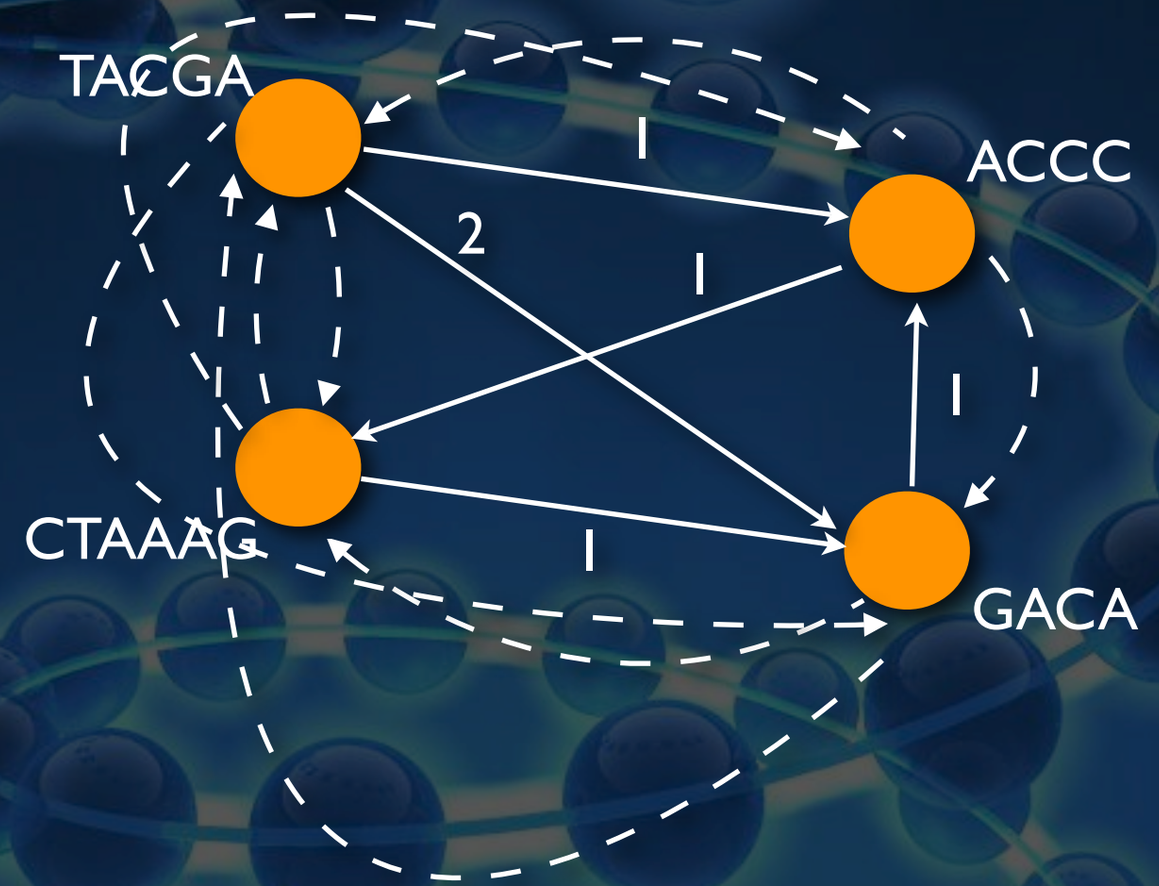
$$w(P) = \sum_{i=0}^{m-1} w(n_i, n_{i+1})$$

Overlap Multigraph

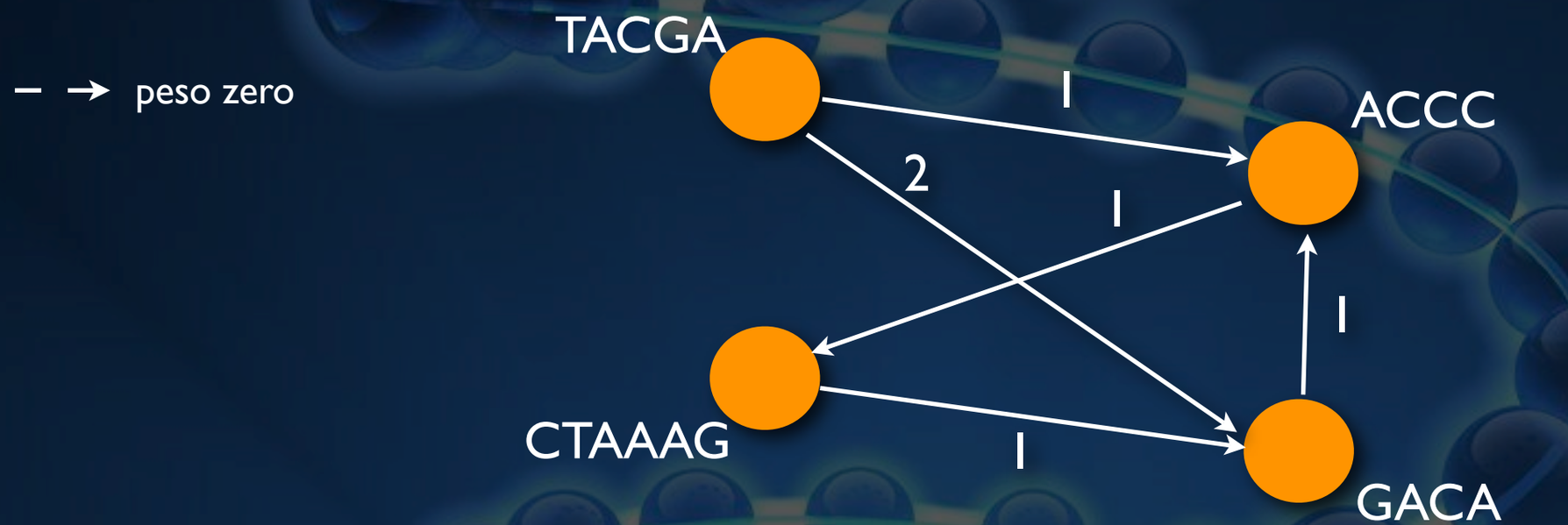
- Multigrafo:
  - Più archi per ogni coppia di nodi
- Un overlap multigraph per la collezione di frammenti $\mathcal{F}$, denotato con $\mathcal{OM}(\mathcal{F})$, è un multigrafo orientato e pesato tale che:
- $\mathcal{N} = \mathcal{F}$
- Ad ogni overlap tra u e v corrisponde un arco con peso pari alla lunghezza dell'overlap

Esempio

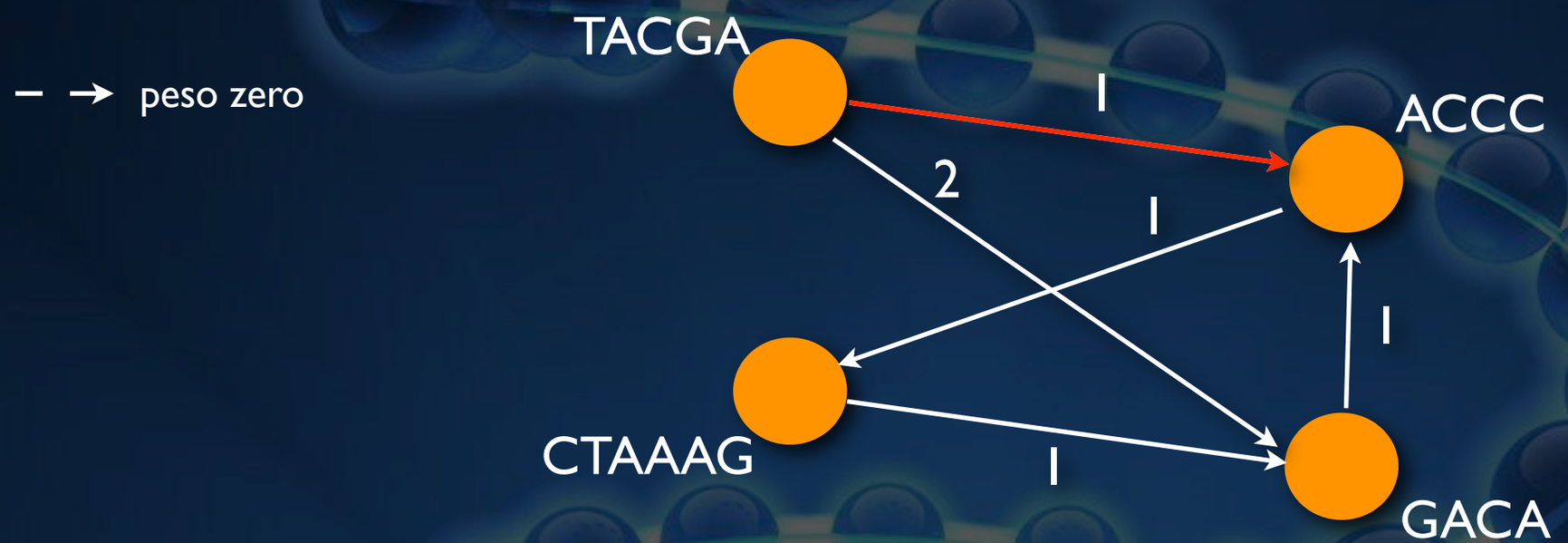
peso zero



Esempio



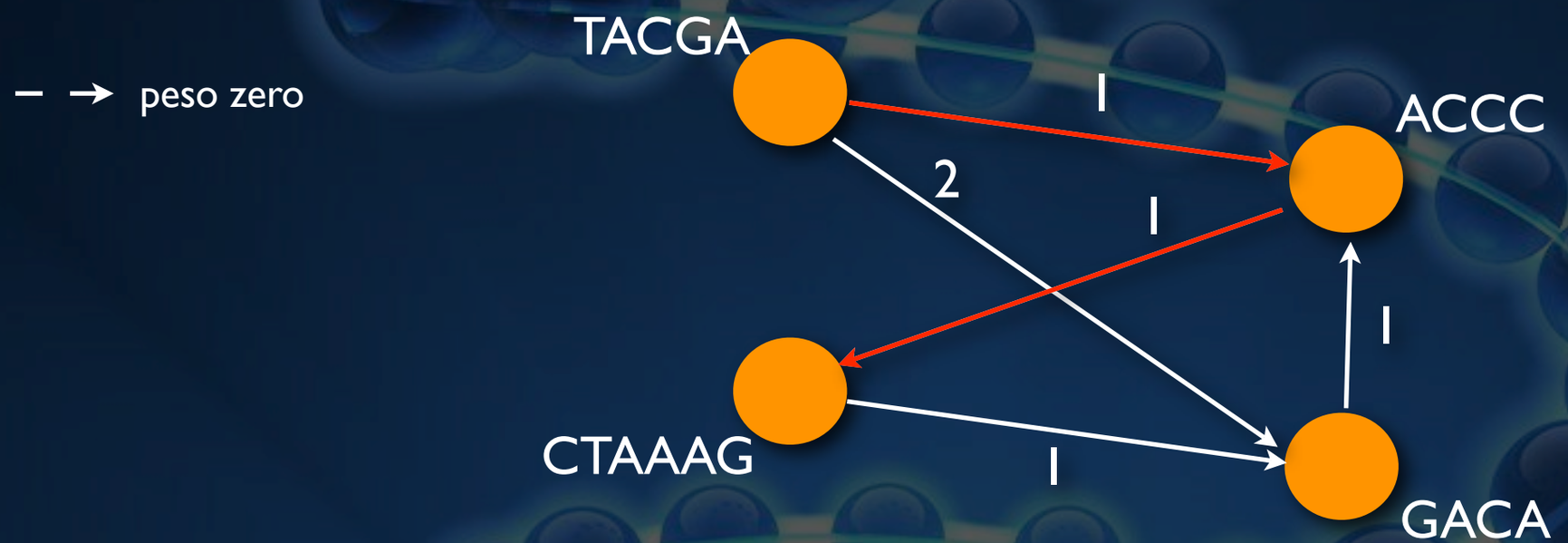
Esempio



TACGA- - - - - - - - - - -

- - - - ACCC- - - - - - - - -

Esempio

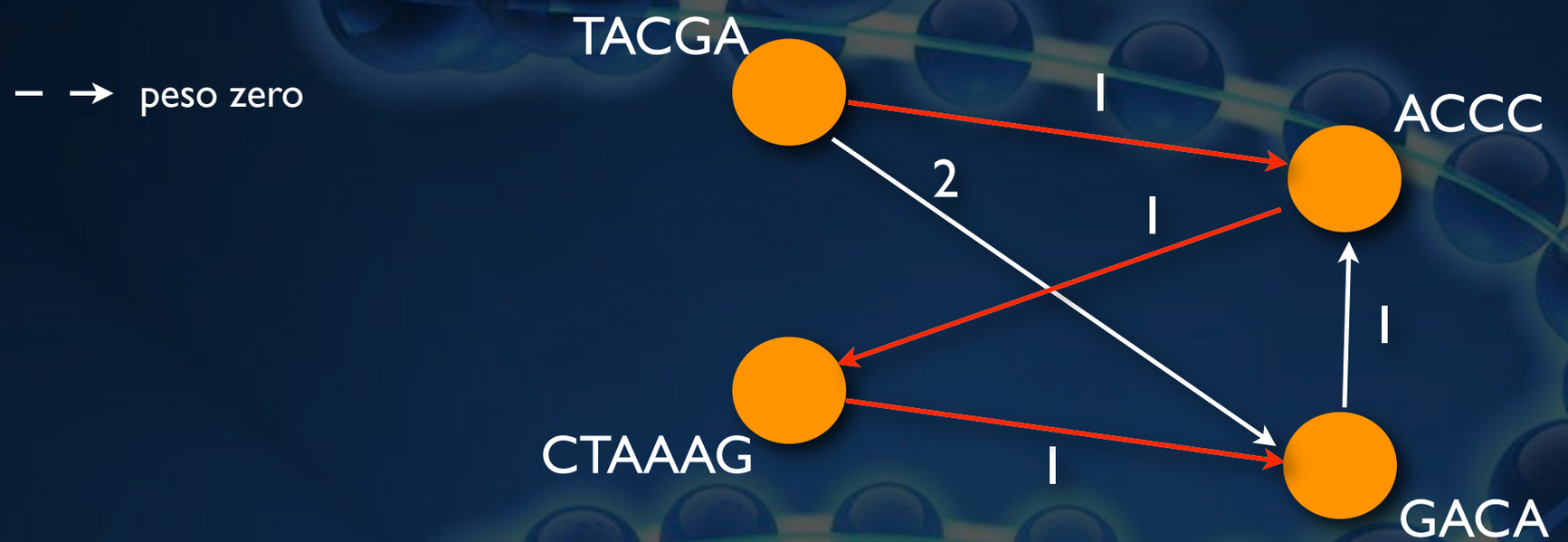


TACGA----------

----ACCC---------

-------CTAAAG---

Esempio



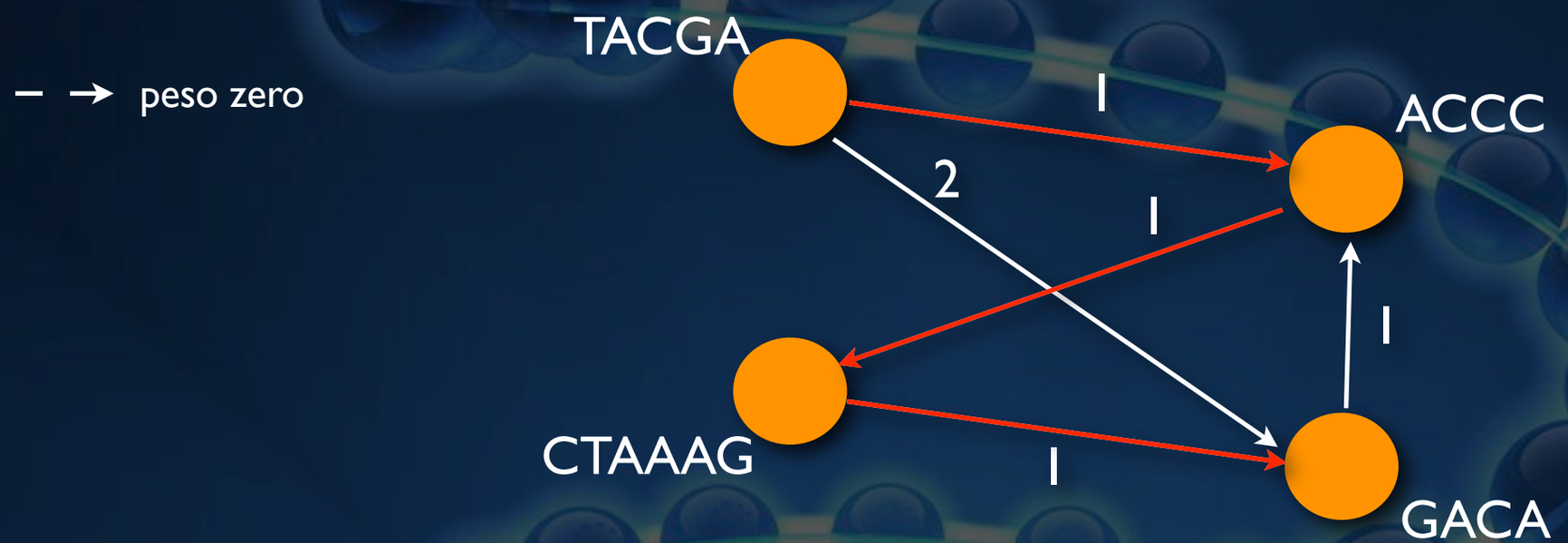
TACGA----------

----ACCC---------

-------CTAAAG---

----------GACA

Esempio




```
TACGA----------
----ACCC-------
-------CTAAAG--
----------GACA
==========
TACGACCCTAAAGACA
```

Cammino Hamiltoniano

- Cammino Hamiltoniano = cammino che attraversa tutti i nodi una sola volta
- In $\mathcal{OM}(\mathcal{F})$ rappresenta una Common Superstring
- Sia $S(\mathcal{H})$ la superstringa associata al cammino hamiltoniano $\mathcal{H}$, vale

$$|S(\mathcal{H})| = \|\mathcal{F}\| - w(\mathcal{H})$$

dove $\|\mathcal{F}\| = \sum_{f \in \mathcal{F}} |f|$

Cammino Hamiltoniano (2)

- $\mathcal{F}$ è substring-free se per ogni $a, b \in \mathcal{F}$, a non è sottostringa di b
- *Teorema:* Se $\mathcal{F}$ è substring-free ed S è la Shortest Common Superstring per $\mathcal{F}$, allora esiste un Cammino Hamiltoniano $\mathcal{H}$ tale che $S = S(\mathcal{H})$
- *Teorema:* Per ogni $\mathcal{F}$ esiste una collezione substring-free equivalente

Algoritmo Greedy

- Non prevede errori e non gestisce l'orientamento
- Obiettivo: Individuare il Cammino Hamiltoniano di costo massimo

$$|S(\mathcal{H})| = ||\mathcal{F}|| - w(\mathcal{H})$$

- Preprocessing:
  - Per ogni coppia di nodi si considera l'arco di peso massimo, ottenendo un semplice grafo (overlap graph $\mathcal{OG}(\mathcal{F})$)
  - Collezione *substring-free*

Algoritmo Greedy (2)

- Idea: Costruiamo un cammino aggiungendo progressivamente gli archi in ordine decrescente di peso
- Prima di aggiungere un arco (u, v) si testano le condizioni di validità del cammino:
  - u non ha già archi uscenti
  - v non ha già archi entranti
  - (u, v) non crea cicli

Algoritmo Greedy (3)

- Test dei cicli:
  - Teniamo traccia dei sottocammini progressivamente costruiti inserendo i nodi in insiemi disgiunti
  - Se gli estremi dell'arco in esame appartengono allo stesso insieme, aggiungendo l'arco al cammino si creerebbe un ciclo

Pseudocodice

for $i \leftarrow 1$ **to** n **do**
    $in[i] \leftarrow 0$
    $out[i] \leftarrow 0$
    MakeSet(i)

Ordina gli archi per peso crescente

foreach arco (f,g) nell'ordine **do**

    if $in[g]=0$ **and** $out[f]=0$ **and** *FindSet*(f) $\neq$ *FindSet*(g)
        select(f,g)
        $in[g] \leftarrow 1$
        $out[f] \leftarrow 1$
        Union(*FindSet*(f),*FindSet*(g))
    if è rimasta una sola componente
        break

return archi selezionati

Pseudocodice

for $i \leftarrow 1$ **to** n **do**
    $in[i] \leftarrow 0$
    $out[i] \leftarrow 0$
    MakeSet(i)

Inizializzazione: nessun nodo ha ancora archi entranti o uscenti, ogni sottocammino contiene un solo nodo

Ordina gli archi per peso crescente
foreach arco (f,g) nell'ordine **do**

    if $in[g]=0$ **and** $out[f]=0$ **and** *FindSet*(f) $\neq$ *FindSet*(g)
        select(f,g)
        $in[g] \leftarrow 1$
        $out[f] \leftarrow 1$
        Union(*FindSet*(f),*FindSet*(g))
    if è rimasta una sola componente
        break
return archi selezionati

Pseudocodice

for $i \leftarrow 1$ **to** n **do**
    $in[i] \leftarrow 0$
    $out[i] \leftarrow 0$
    MakeSet(i)

Inizializzazione: nessun nodo ha ancora archi entranti o uscenti, ogni sottocammino contiene un solo nodo

Ordina gli archi per peso crescente

foreach arco (f,g) nell'ordine **do**

    if $in[g]=0$ **and** $out[f]=0$ **and** *FindSet*(f) $\neq$ *FindSet*(g)
        select(f,g)
        $in[g] \leftarrow 1$
        $out[f] \leftarrow 1$
        Union(*FindSet*(f),*FindSet*(g))
    if è rimasta una sola componente
        break

return archi selezionati

Test sugli archi entranti ed uscenti

Pseudocodice

for $i \leftarrow 1$ **to** n **do**
    $in[i] \leftarrow 0$
    $out[i] \leftarrow 0$
    MakeSet(i)

Inizializzazione: nessun nodo ha ancora archi entranti o uscenti, ogni sottocammino contiene un solo nodo

Ordina gli archi per peso crescente
foreach arco (f,g) nell'ordine **do**

Test di ciclo

if $in[g]=0$ **and** $out[f]=0$ **and** *FindSet*(f) $\neq$ *FindSet*(g)

Test sugli archi entranti ed uscenti

        select(f,g)
        $in[g] \leftarrow 1$
        $out[f] \leftarrow 1$
        Union(*FindSet*(f),*FindSet*(g))
    if è rimasta una sola componente
        break
return archi selezionati

Pseudocodice

for $i \leftarrow 1$ **to** n **do**
    $in[i] \leftarrow 0$
    $out[i] \leftarrow 0$
    MakeSet(i)

Inizializzazione: nessun nodo ha ancora archi entranti o uscenti, ogni sottocammino contiene un solo nodo

Ordina gli archi per peso crescente

foreach arco (f,g) nell'ordine **do**

    if $in[g]=0$ **and** $out[f]=0$ **and** *FindSet*(f) $\neq$ *FindSet*(g)

Test di ciclo

Test sugli archi entranti ed uscenti

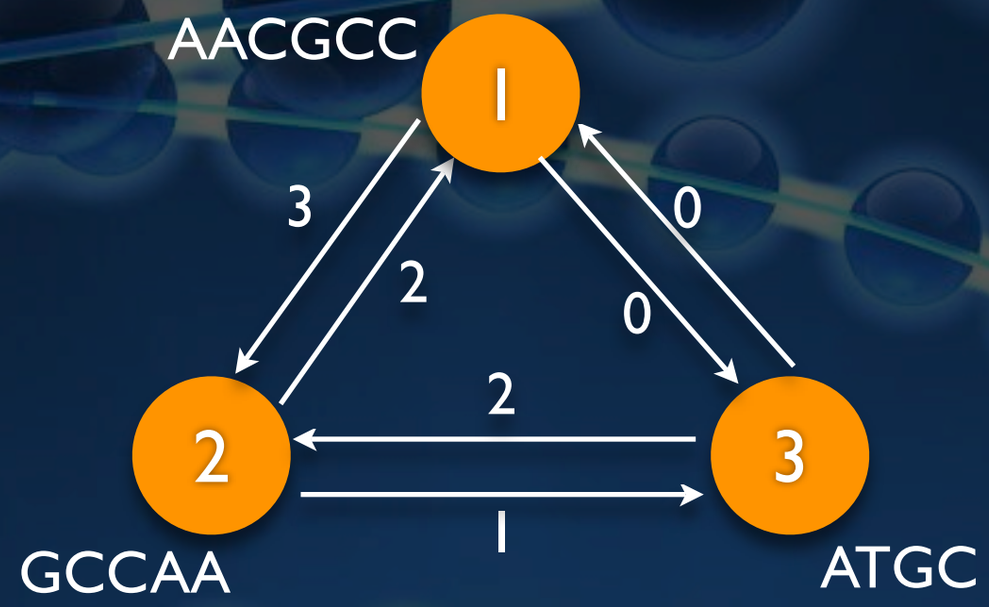
        select(f,g)
        $in[g] \leftarrow 1$
        $out[f] \leftarrow 1$
        Union(*FindSet*(f),*FindSet*(g))

Aggiunta dell'arco al cammino e fusione dei sottocammini parziali

    if è rimasta una sola componente
        break

return archi selezionati

Esempio



Inizializzazione

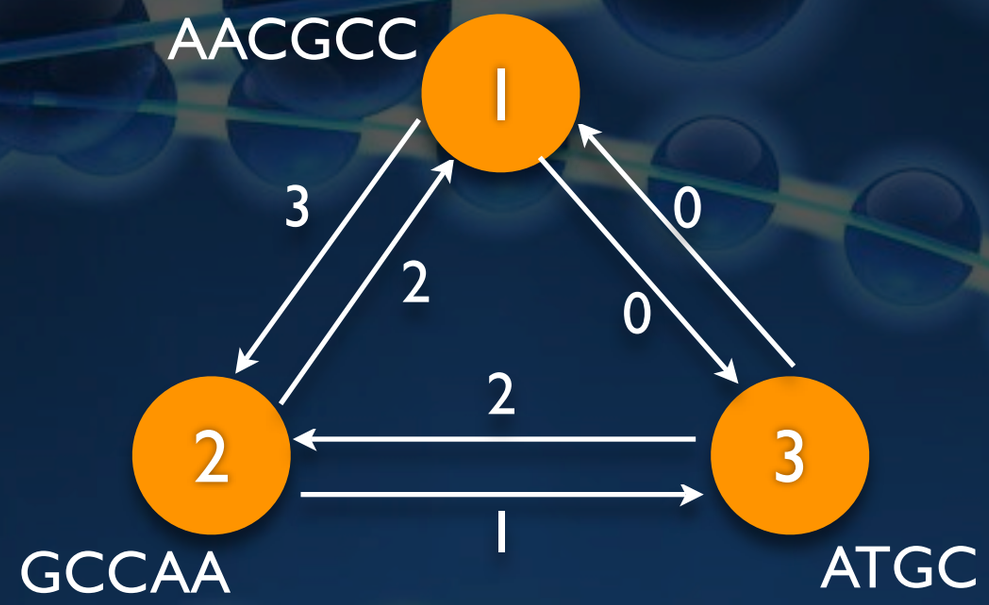
in[0,0,0] out[0,0,0]

Risultato MakeSet()
Set_1={ 1 }, Set_2={ 2 }, Set_3={ 3 }

Ordinamento Archi
(1,2), (2,1), (3,2), (2,3), (1,3), (3,1)

Esempio

Iterazione I



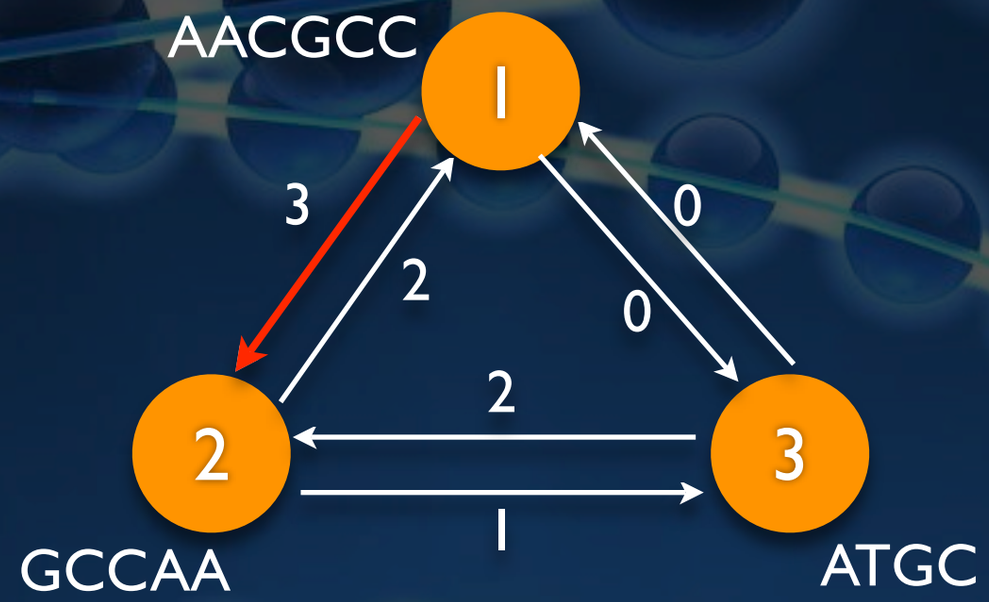
Stato

in[0,0,0] out[0,0,0]

Set_1={ 1 }, Set_2={ 2 }, Set_3={ 3 }

Esempio

Iterazione I



Arco Selezionato (1,2)

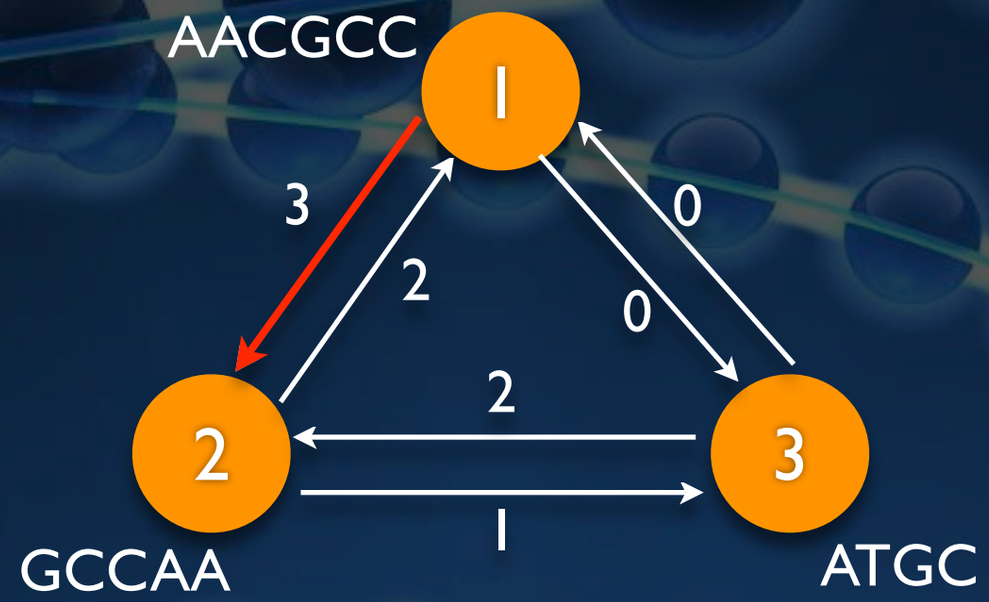
Stato

in[0,0,0] out[0,0,0]

Set_1={ 1 }, Set_2={ 2 }, Set_3={ 3 }

Esempio

Iterazione I



Arco Selezionato (1,2)

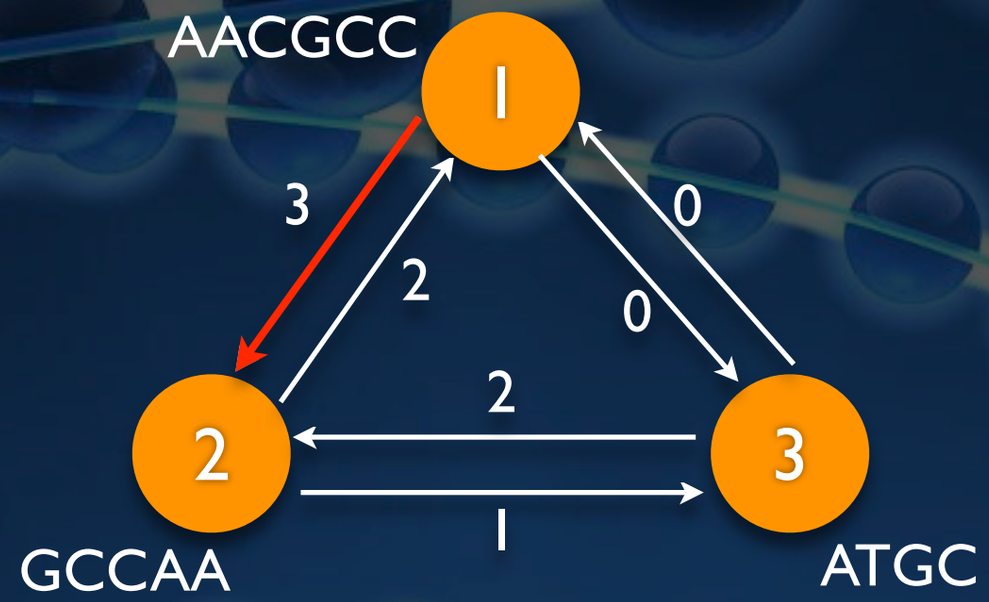
Stato

in[0, 1,0] out[1,0,0]

Set_1={ 1 }, Set_2={ 2 }, Set_3={ 3 }

Esempio

Iterazione I



Arco Selezionato (1,2)

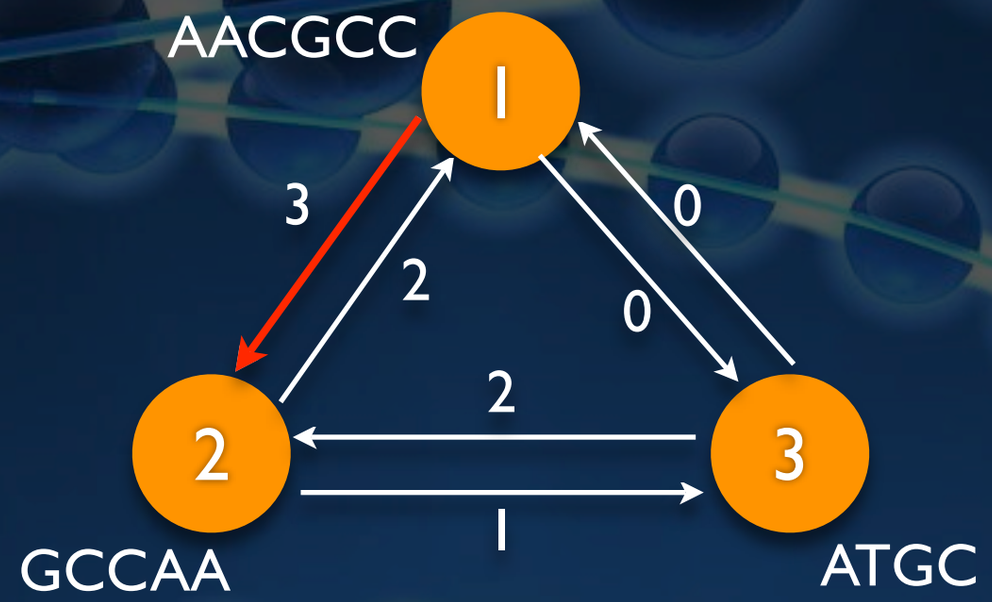
Stato

in[0, 1,0] out[1,0,0]

$Set_{1,2}$={ 1,2 }, Set_3={ 3 }

Esempio

Iterazione I



Arco Selezionato (1,2)

Stato

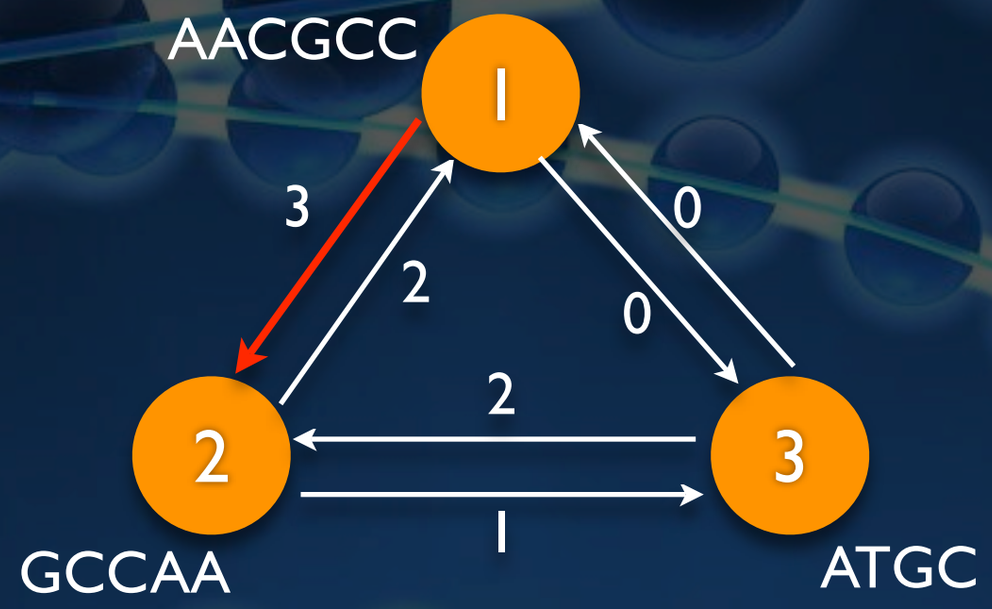
in[0, 1,0] out[1,0,0]

$Set_{1,2}$={ 1,2 }, Set_3={ 3 }

Archi rimanenti
(2,1), (3,2),
(2,3), (1,3), (3,1)

Esempio

Iterazione 2



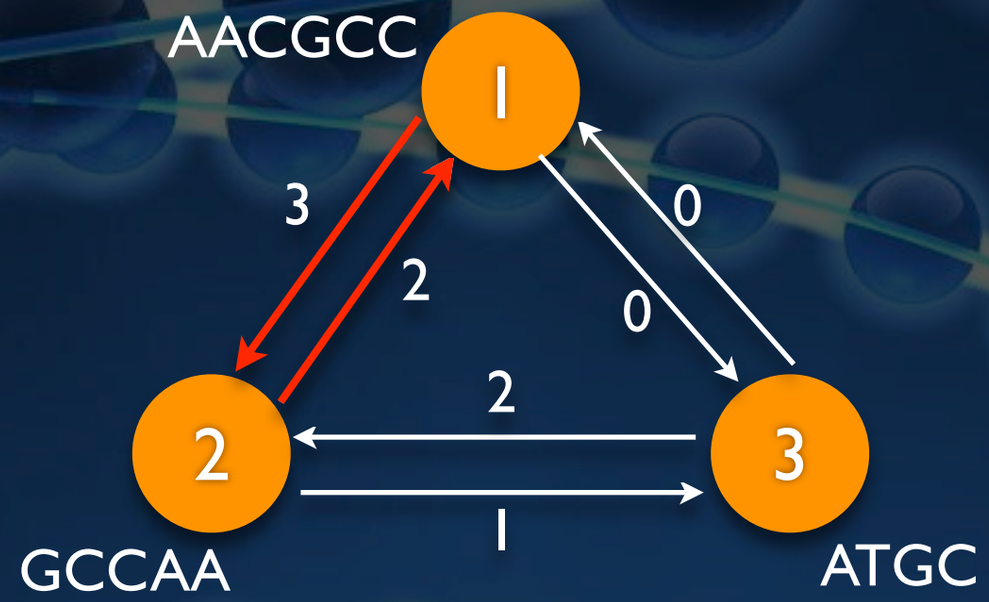
Stato

in[0,1,0] out[1,0,0]

$Set_{1,2}$={ 1,2 }, Set_3={ 3 }

Esempio

Iterazione 2



Arco Selezionato (2,1)

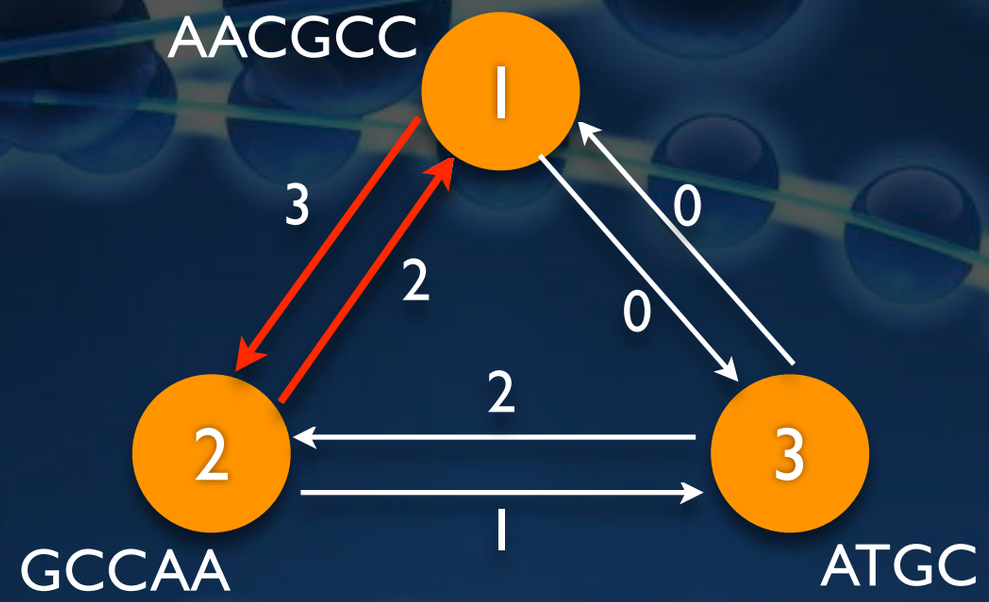
Stato

in[0, 1,0] out[1,0,0]

$Set_{1,2}$={ 1,2 }, Set_3={ 3 }

Esempio

Iterazione 2



ciclo

Arco Selezionato (2,1)

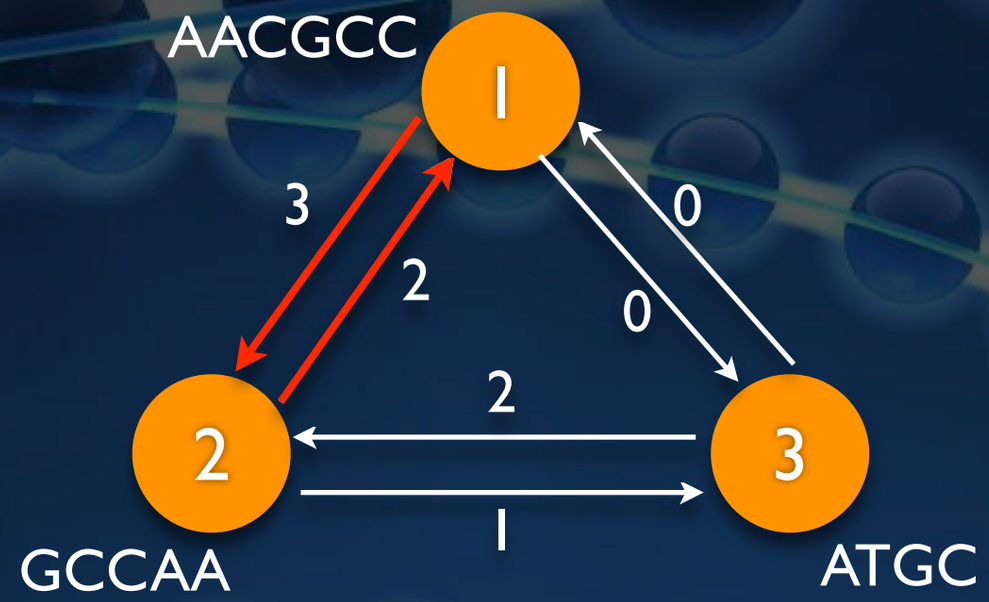
Stato

in[0, 1,0] out[1,0,0]

$Set_{1,2}$={ 1,2 }, Set_3={ 3 }

Esempio

Iterazione 2



ciclo

Arco Selezionato (2,1)

Stato

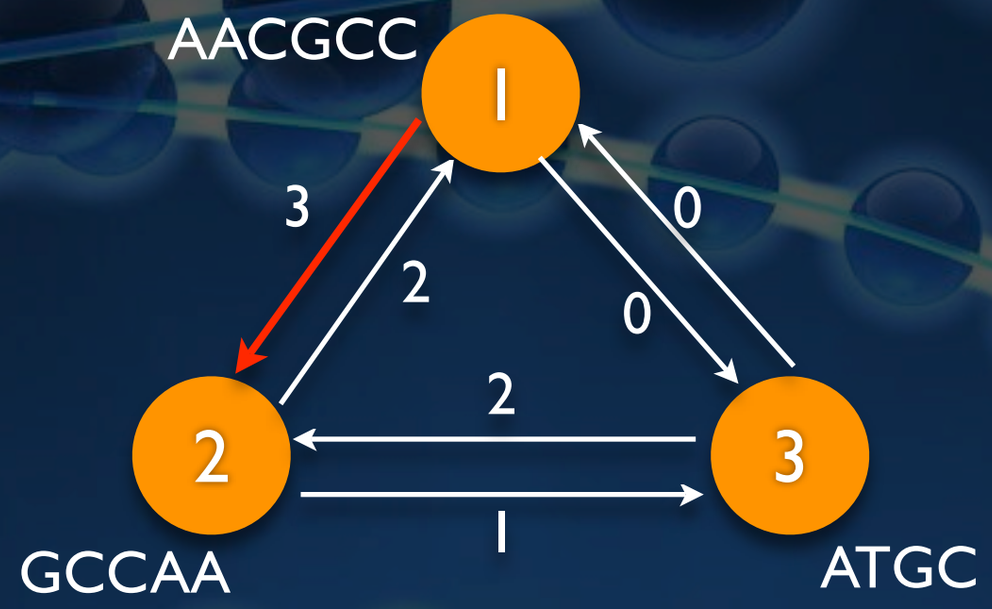
in[0,1,0] out[1,0,0]

$Set_{1,2}$={ 1,2 }, Set_3={ 3 }

Archi rimanenti
(3,2), (2,3),
(1,3), (3,1)

Esempio

Iterazione 3



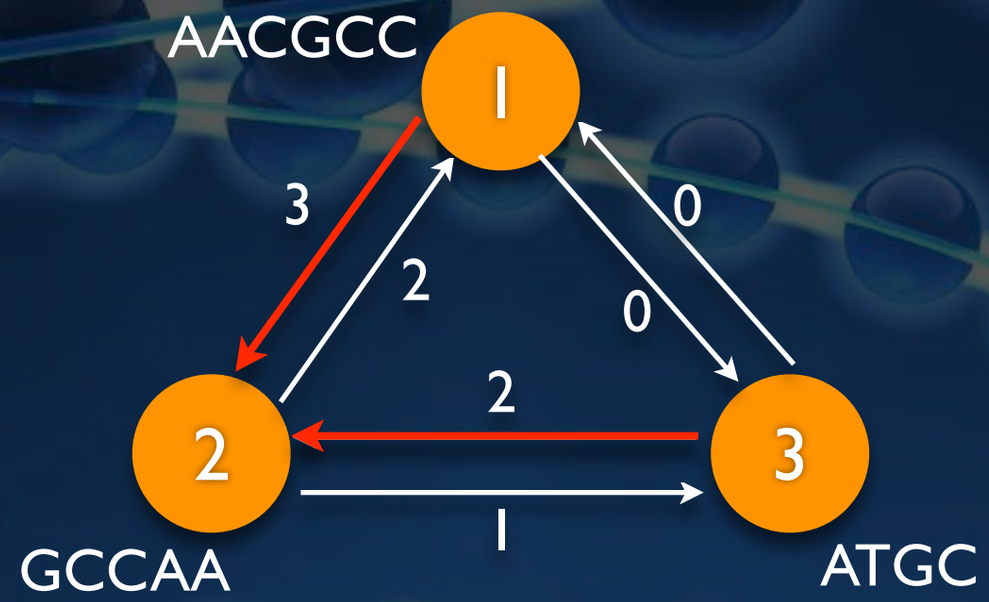
Stato

in[0,1,0] out[1,0,0]

$Set_{1,2}$={ 1,2 }, Set_3={ 3 }

Esempio

Iterazione 3



Arco Selezionato (3,2)

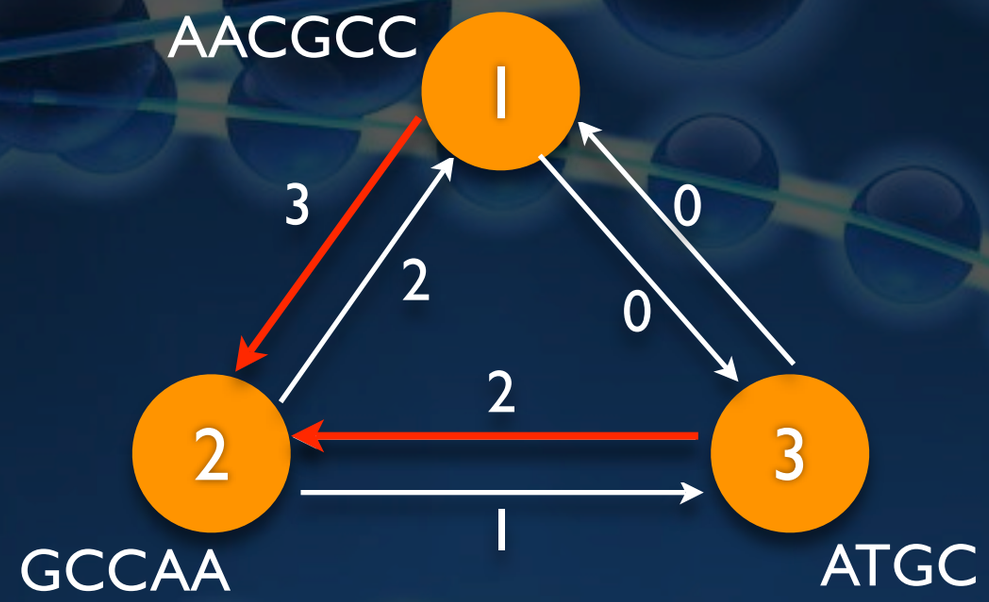
Stato

in[0, 1,0] out[1,0,0]

$Set_{1,2}$={ 1,2 }, Set_3={ 3 }

Esempio

Iterazione 3



Arco Selezionato (3,2)

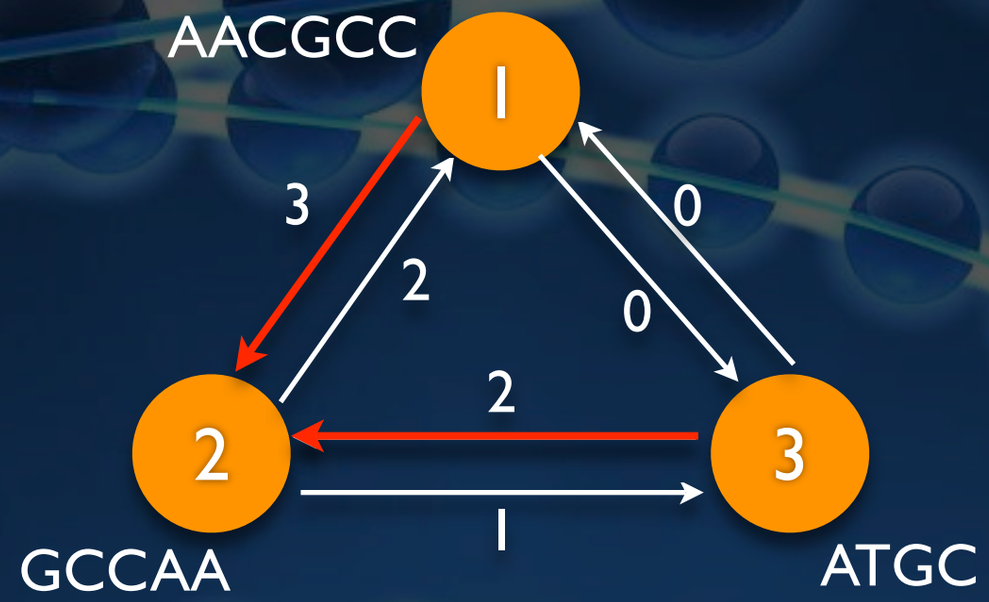
Stato

in[0,1,0] out[1,0,0]

$Set_{1,2}$={ 1,2 }, Set_3={ 3 }

Esempio

Iterazione 3



Arco Selezionato (3,2)

Stato

in[0,1,0] out[1,0,0]

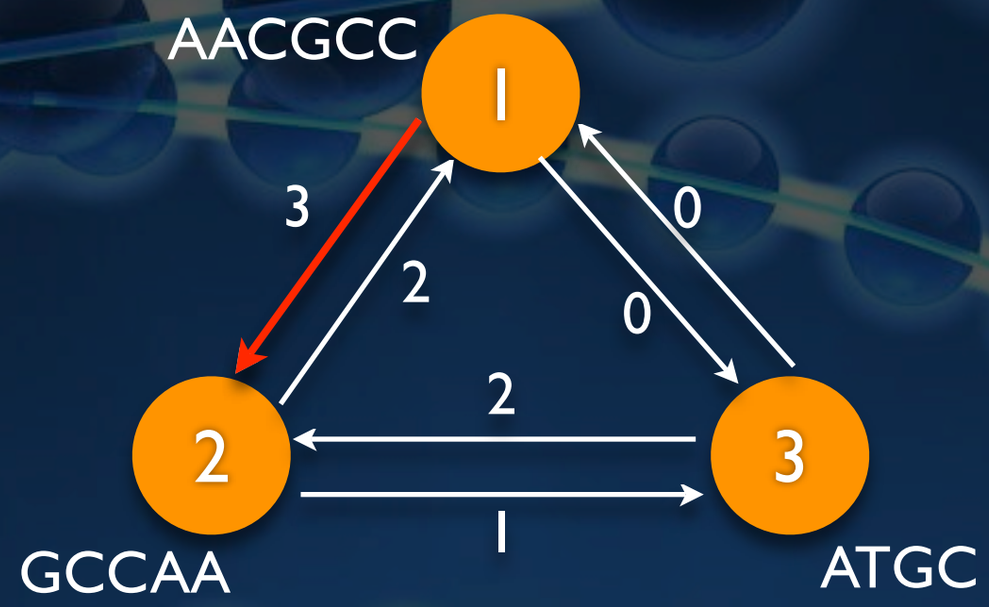
$Set_{1,2}$={ 1,2 }, Set_3={ 3 }

Archi rimanenti

(2,3), (1,3), (3,1)

Esempio

Iterazione 4



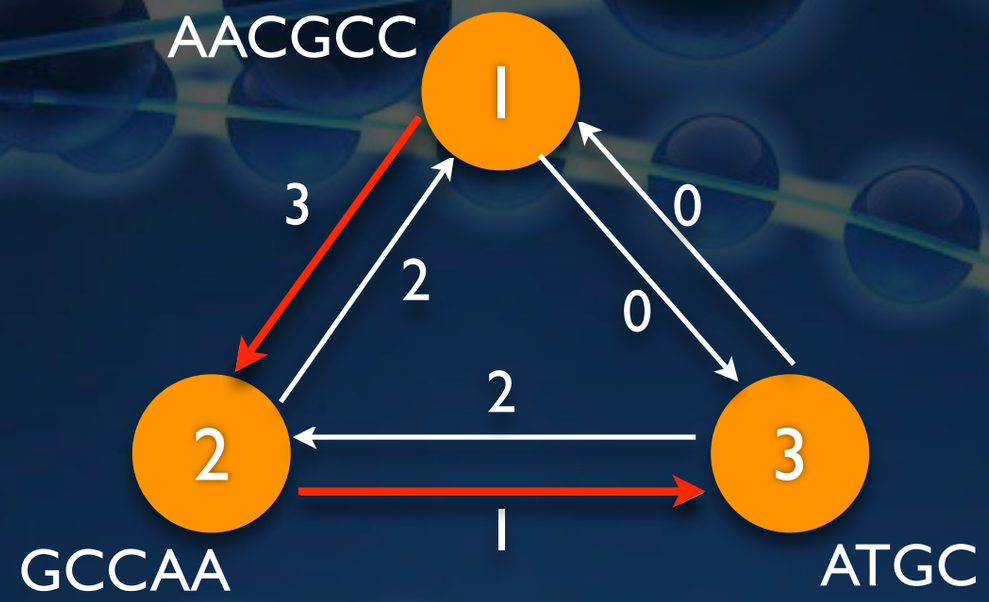
Stato

in[0,1,0] out[1,0,0]

$Set_{1,2}$={ 1,2 }, Set_3={ 3 }

Esempio

Iterazione 4



Arco Selezionato (2,3)

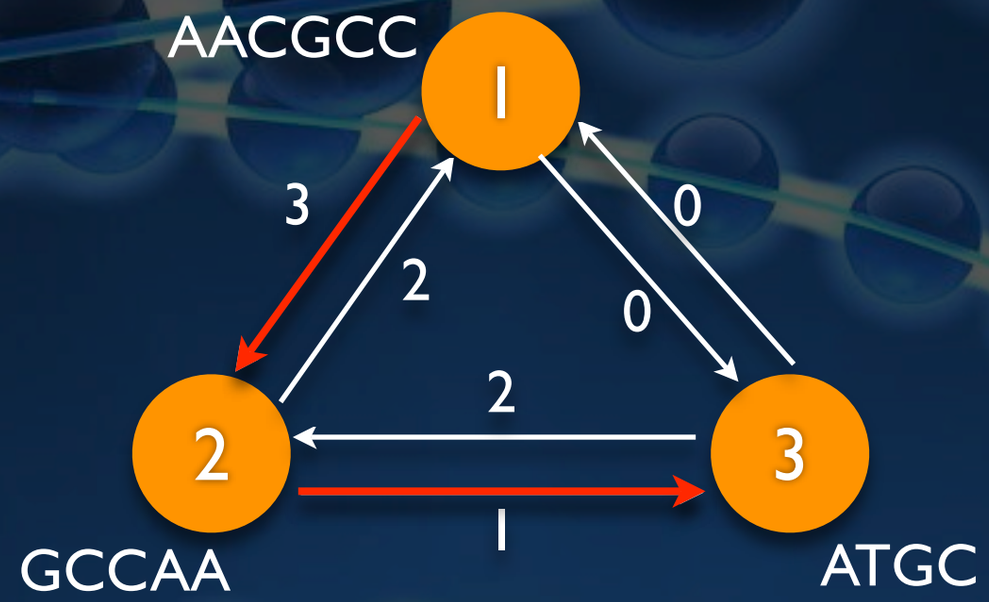
Stato

in[0, 1,0] out[1,0,0]

$Set_{1,2}$={ 1,2 }, Set_3={ 3 }

Esempio

Iterazione 4



Arco Selezionato (2,3)

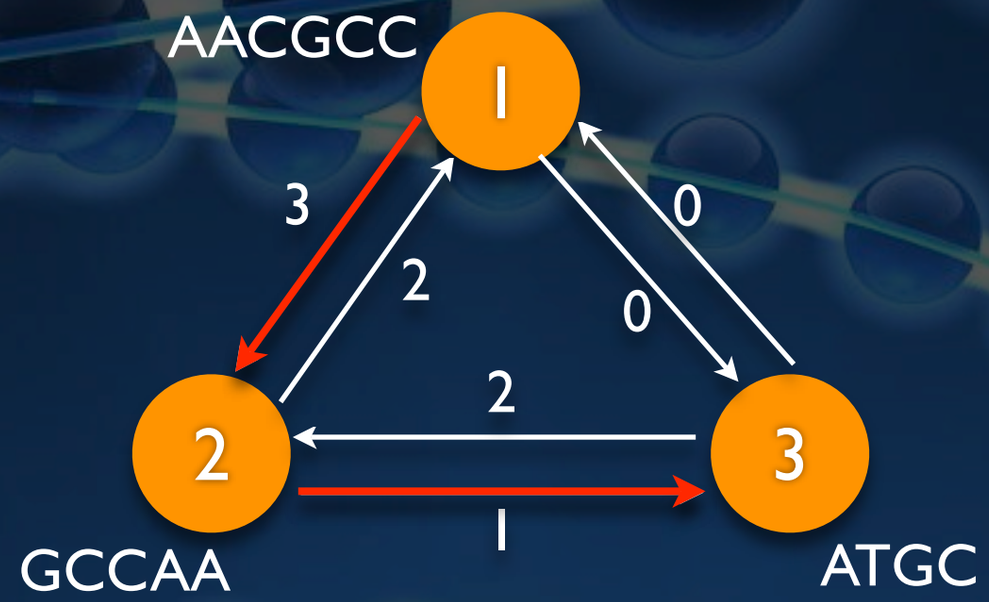
Stato

in[0, 1, 1] out[1, 1,0]

$Set_{1,2}$={ 1,2 }, Set_3={ 3 }

Esempio

Iterazione 4



Arco Selezionato (2,3)

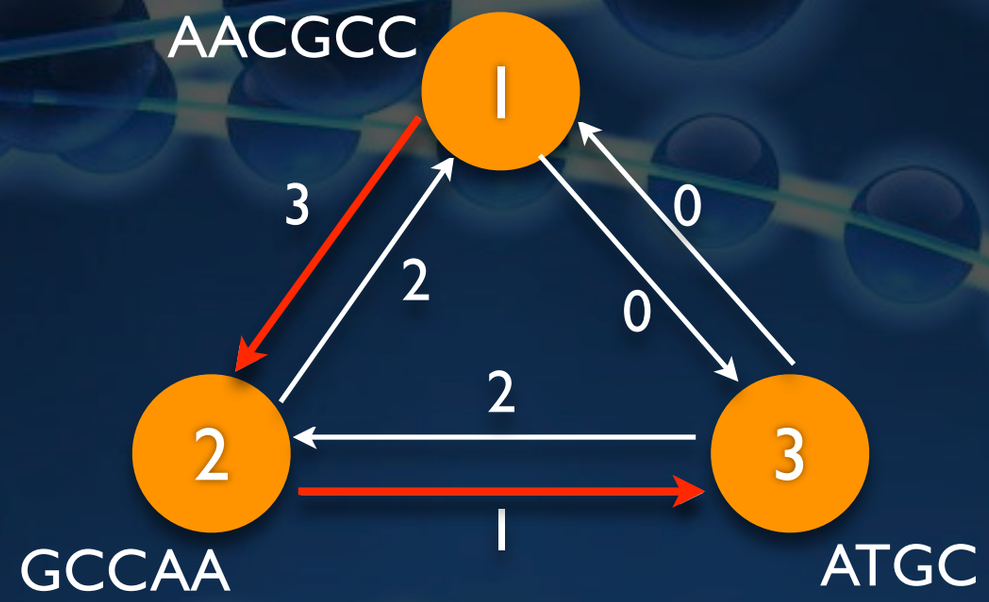
Stato

in[0, 1, 1] out[1, 1, 0]

$Set_{1,2,3}$={ 1,2,3 }

Esempio

Iterazione 4



Arco Selezionato (2,3)

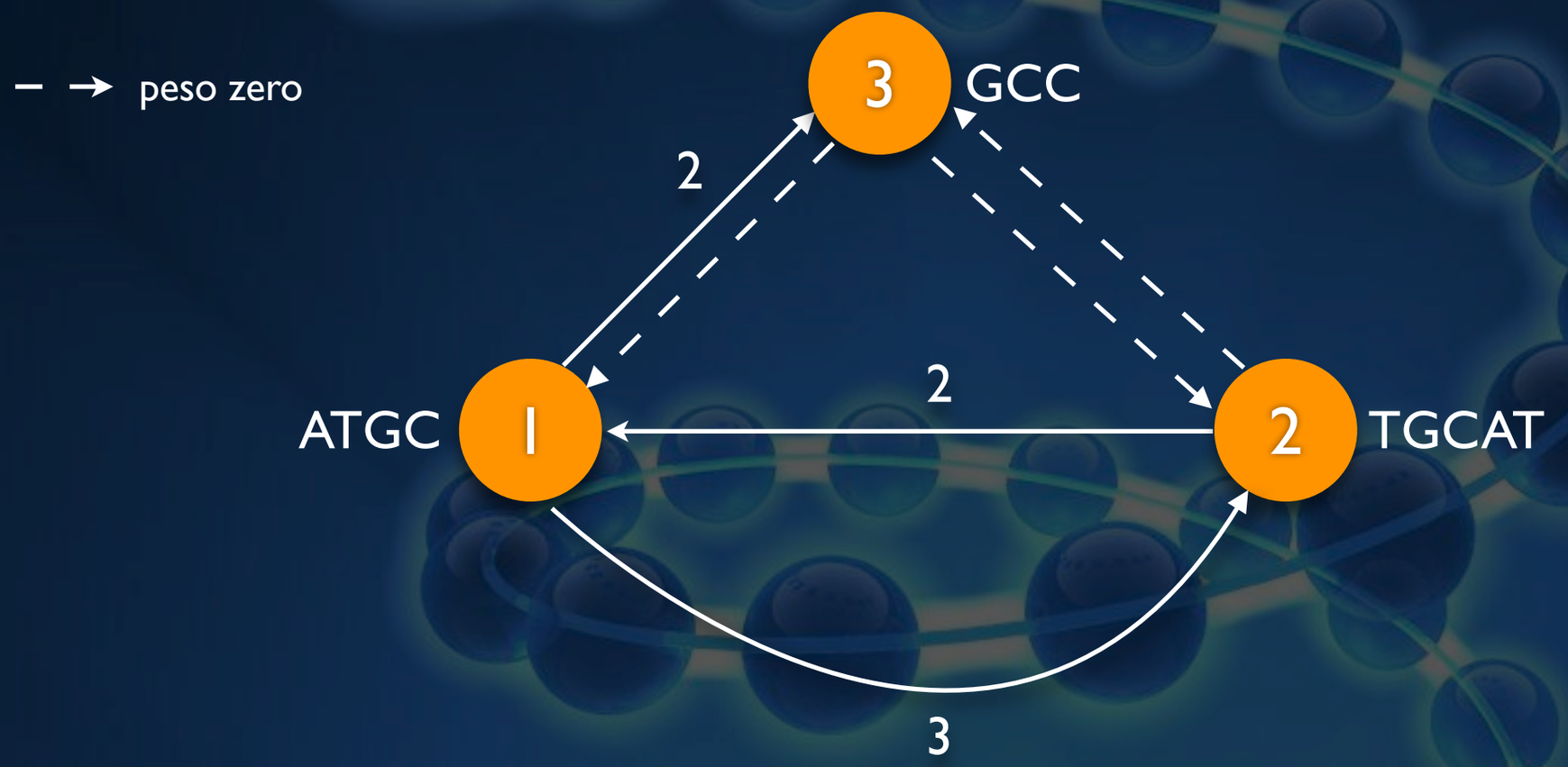
Stato

in[0,1,1] out[1,1,0]

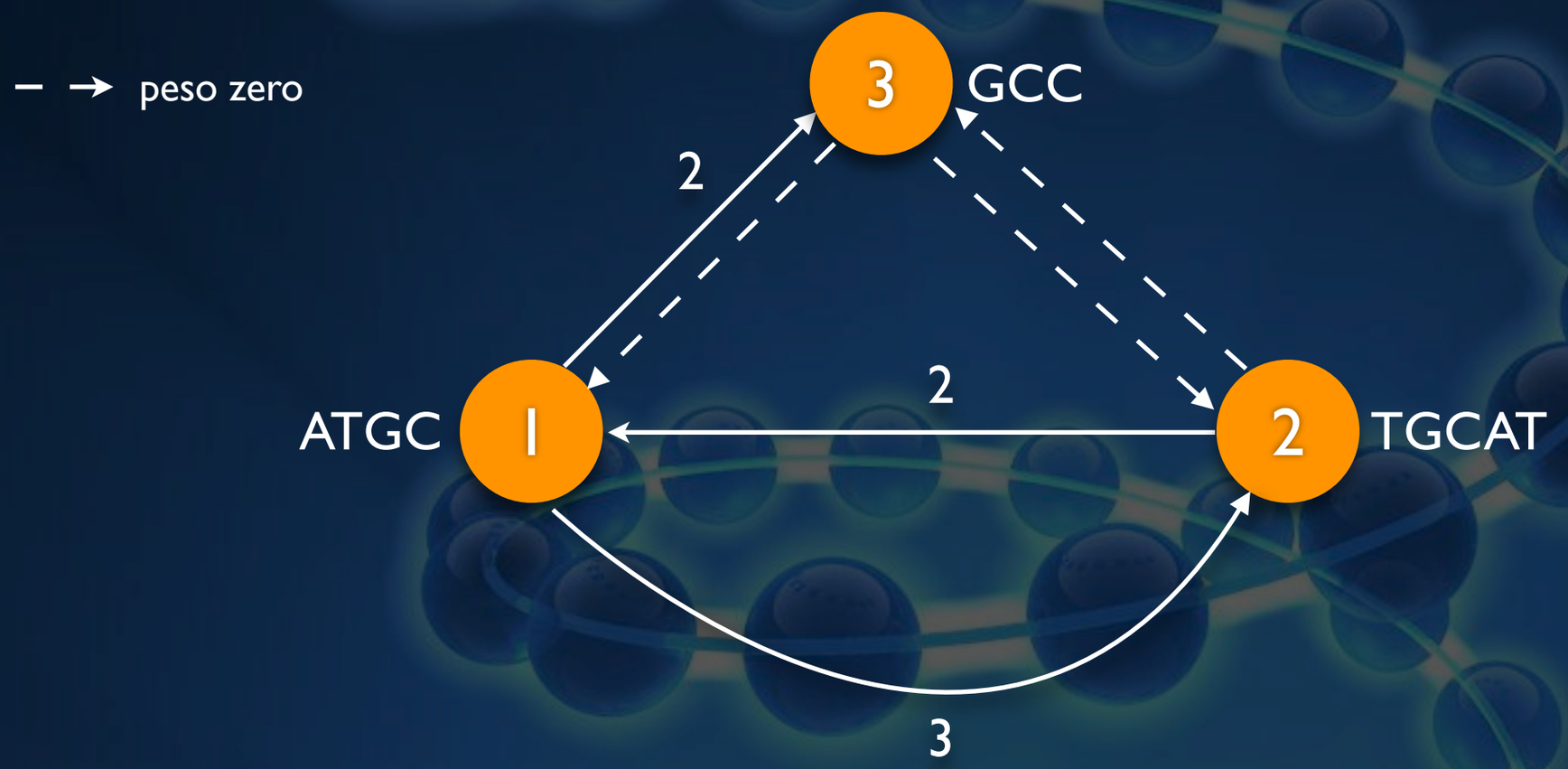
$Set_{1,2,3}$={ 1,2,3 }

Fine Algoritmo

Caso Fallimentare

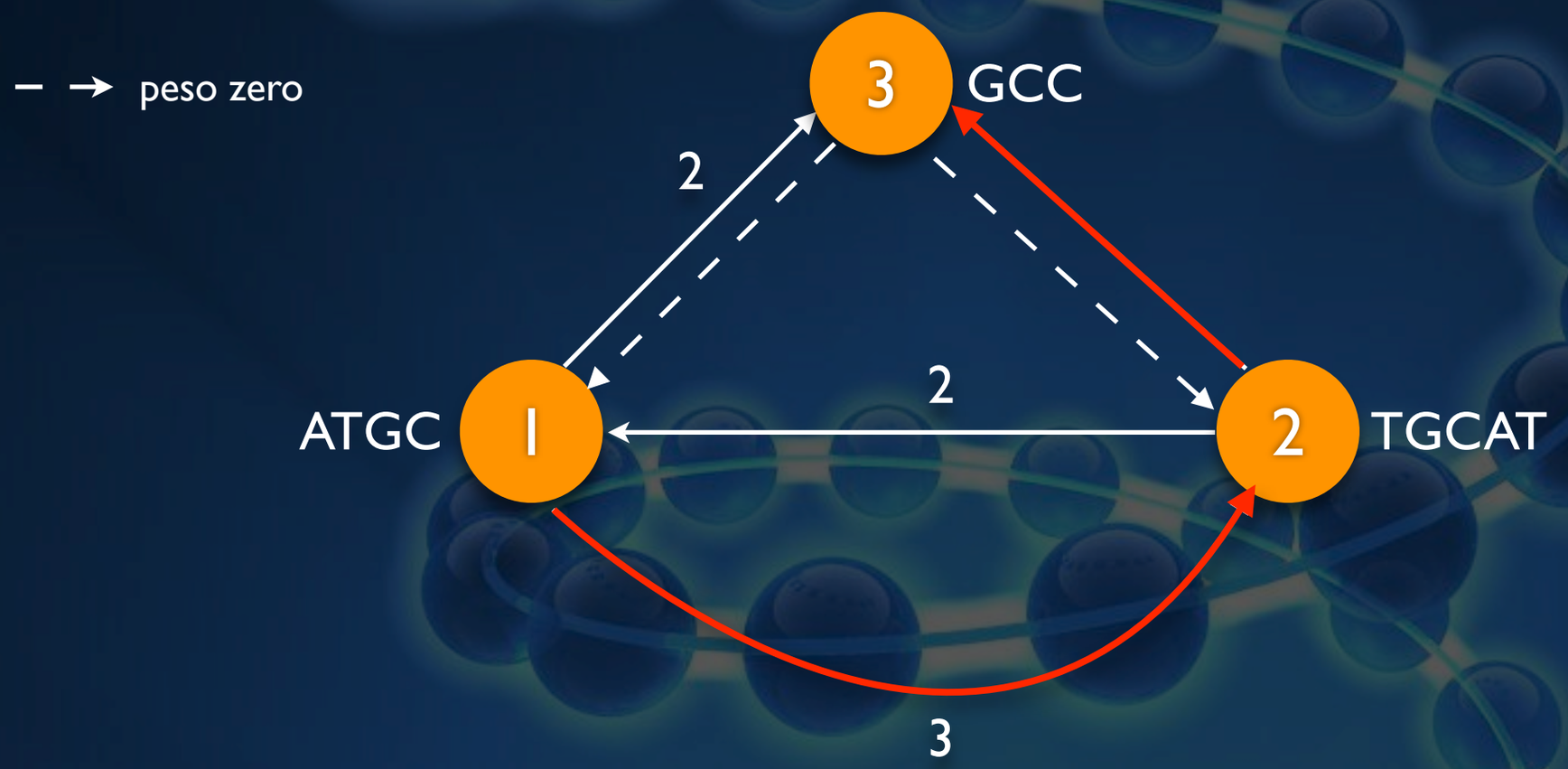


Caso Fallimentare



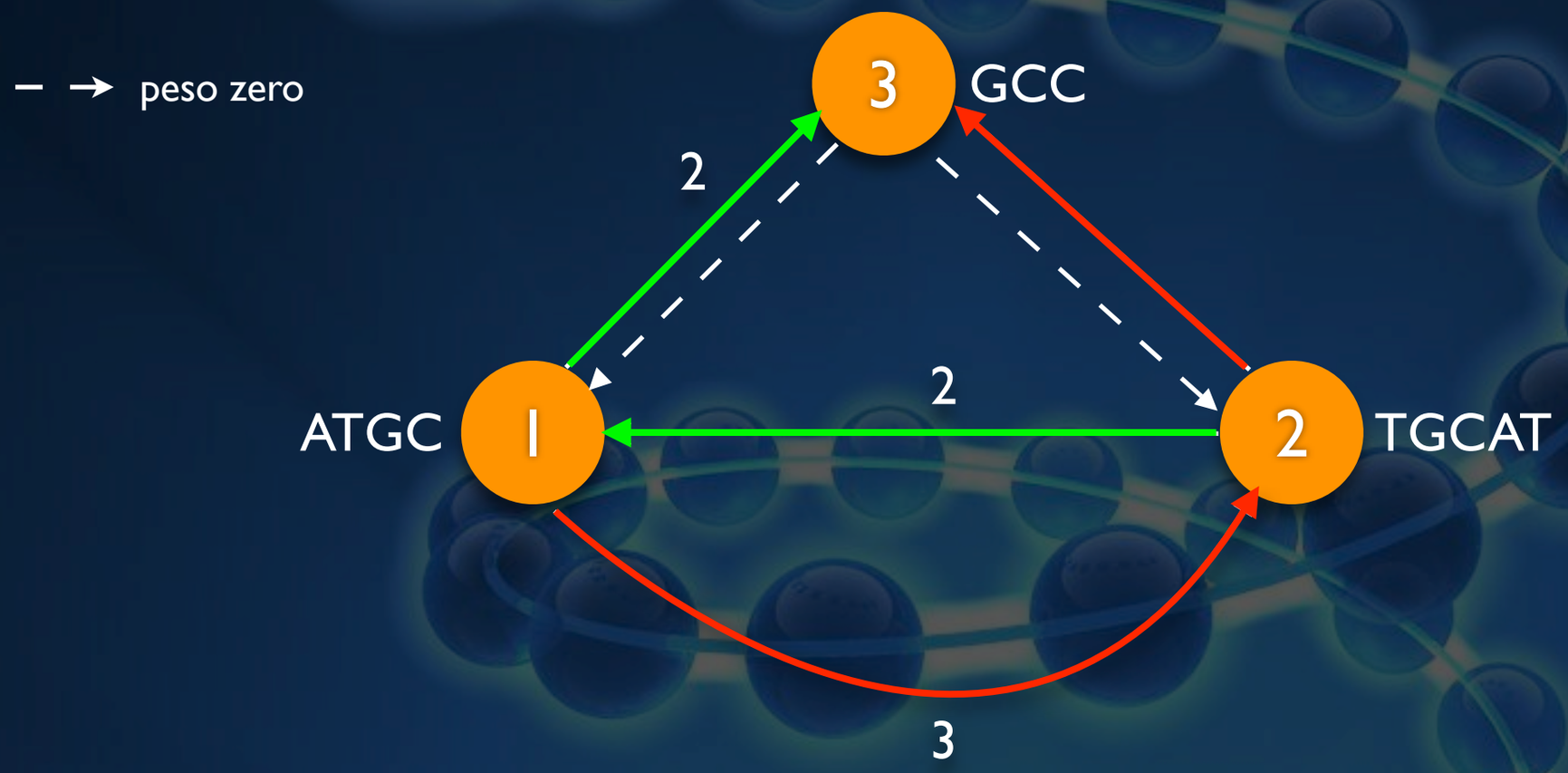
- Algoritmo greedy fallisce:

Caso Fallimentare



- Algoritmo greedy fallisce:
  - Costo cammino individuato = 3

Caso Fallimentare



- Algoritmo greedy fallisce:
  - Costo cammino individuato = 3
  - Costo cammino corretto = 4

Problema dell'Orientamento

- Possibile soluzione:
  - Si estende la collezione con i frammenti complementari inversi

$$\mathcal{F}' = \mathcal{F} \cup \{\overline{f} | f \in \mathcal{F}\}$$

- Si cerca un cammino nel grafo $\mathcal{OG}(\mathcal{F}')$, trovando idealmente un cammino complementare

$$f_1 \rightarrow f_2 \rightarrow \cdots \rightarrow f_k$$
$$\overline{f_k} \rightarrow \overline{f_{k-1}} \rightarrow \cdots \rightarrow \overline{f_1}$$

Problema degli errori

- Overlap approssimati
- Collezione substring-free in senso approssimato
- Possibilità di layout multipli per un medesimo cammino
- Soluzione: opportune strutture dati

Esempio

$$f = \mathrm{CATAGTC}, g = \mathrm{TAACTAT}, h = \mathrm{AGACTATCC}$$

Cammino: $f \to g \to h$

Esempio

$$f = \mathrm{CATAGTC}, g = \mathrm{TAACTAT}, h = \mathrm{AGACTATCC}$$

Cammino: $f \to g \to h$

Allineamenti semiglobali ottimali tra f e g

```
C A T A G T C - - -        C A T A G T C - - -
- - T A A - C T A T        - - T A - A C T A T
```

Esempio

$$f = \mathrm{CATAGTC}, g = \mathrm{TAACTAT}, h = \mathrm{AGACTATCC}$$

Cammino: $f \to g \to h$

Allineamenti semiglobali ottimali tra f e g

```
C A T A G T C - - -        C A T A G T C - - -
- - T A A - C T A T        - - T A - A C T A T
```

Caso 1.1

```
C A T A G T C - - - - -    d_s(f, S) = 1
- - T A A - C T A T - -    d_s(g, S) = 2
- - - A G A C T A T C C    d_s(h, S) = 0
-------------------------
C A T A G A C T A T C C
```

Esempio

$$f = \mathrm{CATAGTC}, g = \mathrm{TAACTAT}, h = \mathrm{AGACTATCC}$$

Cammino: $f \to g \to h$

Allineamenti semiglobali ottimali tra f e g

```
C A T A G T C - - -        C A T A G T C - - -
- - T A A - C T A T        - - T A - A C T A T
```

Caso 1.2

```
C A T A G T C - - - - -      d_s(f, S) = 0
- - T A A - C T A T - -      d_s(g, S) = 2
- - - A G A C T A T C C      d_s(h, S) = 1
-----------------------
C A T A G T C T A T C C
```

Esempio

$$f = \mathrm{CATAGTC}, g = \mathrm{TAACTAT}, h = \mathrm{AGACTATCC}$$

Cammino: $f \to g \to h$

Allineamenti semiglobali ottimali tra f e g

```
C A T A G T C - - -        C A T A G T C - - -
- - T A A - C T A T        - - T A - A C T A T
```

Caso 2

```
C A T A G T C - - - - -    d_s(f, S) = 1
- - T A - A C T A T - -    d_s(g, S) = 1
- - - A G A C T A T C C    d_s(h, S) = 0
-------------------------
C A T A G A C T A T C C
```

Problema della mancanza di copertura e ripetizioni

- Un cammino in $\mathcal{OG}(\mathcal{F}')$ con archi di costo 0 indica una possibile mancanza di copertura
- Indicatori di ripetizioni:
  - Cicli in $\mathcal{OG}(\mathcal{F}')$
  - Copertura particolarmente elevata

- Introduzione al Problema
- Modelli Computazionali
- Algoritmi
- Soluzioni Reali

- Introduzione al Problema
- Modelli Computazionali
- Algoritmi
- Soluzioni Reali
  - Grafo di de Brujin
  - String Graph
  - Confronto

Soluzioni Reali

- Modelli più complessi trattano esplicitamente il problema dell'orientamento e delle ripetizioni
- Strutture:
  - Grafo di De Bruijn
  - String Graph (Meyers)

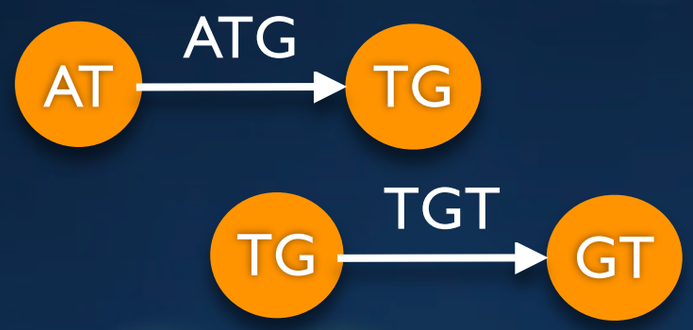
Grafo di De Bruijn

- Preprocessing:
  - Dai frammenti si generano tutte le possibili sottostringhe lunghe k (k-*mers*)
  - Motivo: le istanze di una ripetizione collassano in un singolo insieme di vertici
- Il grafo di De Bruijn è un grafo diretto con:
  - $\mathcal{N}$ = k-mers (senza ripetizioni)
  - $\mathcal{A}$ = $\{(k_1, k_2) | k_1[2..k] = k_2[1..k-1]\}$, ossia rappresentano $k+1$-mers

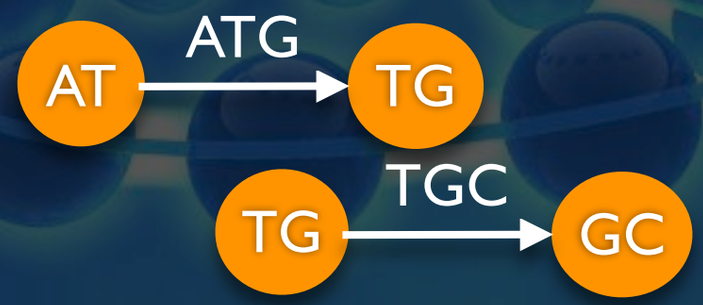
Esempio

$k = 2$

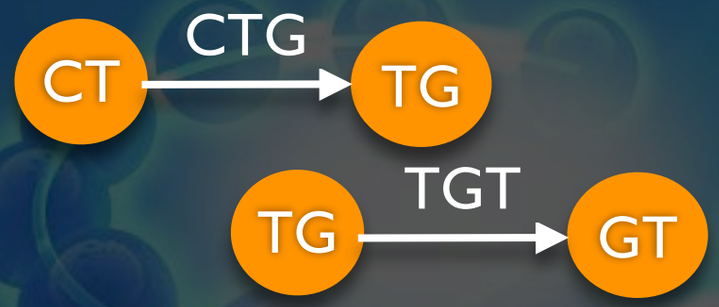
ATGT
AT
TG
GT

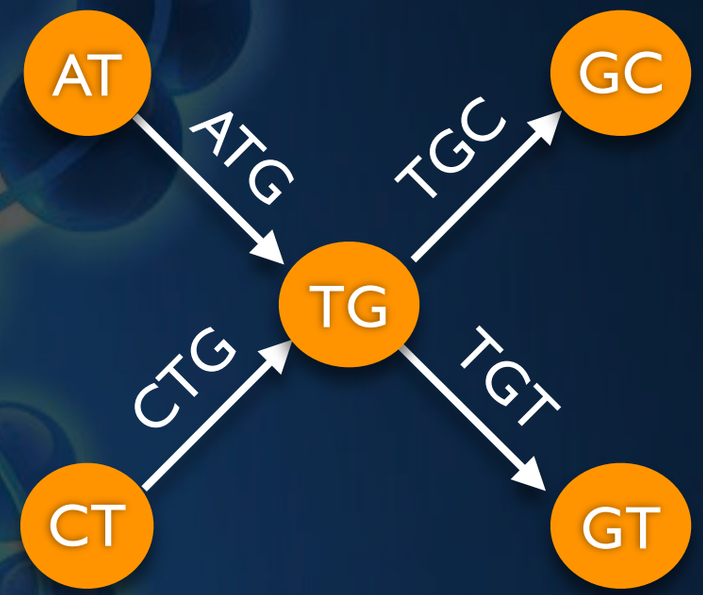


ATGC
AT
TG
GC



CTGT
CT
TG
GT





De Bruijn Superwalk Problem

- Ogni frammento f corrisponde ad un cammino nel grafo di De Bruijn

  $$w(f) = f[1..k] \rightarrow \cdots \rightarrow f[|f| - k + 1..|f|]$$

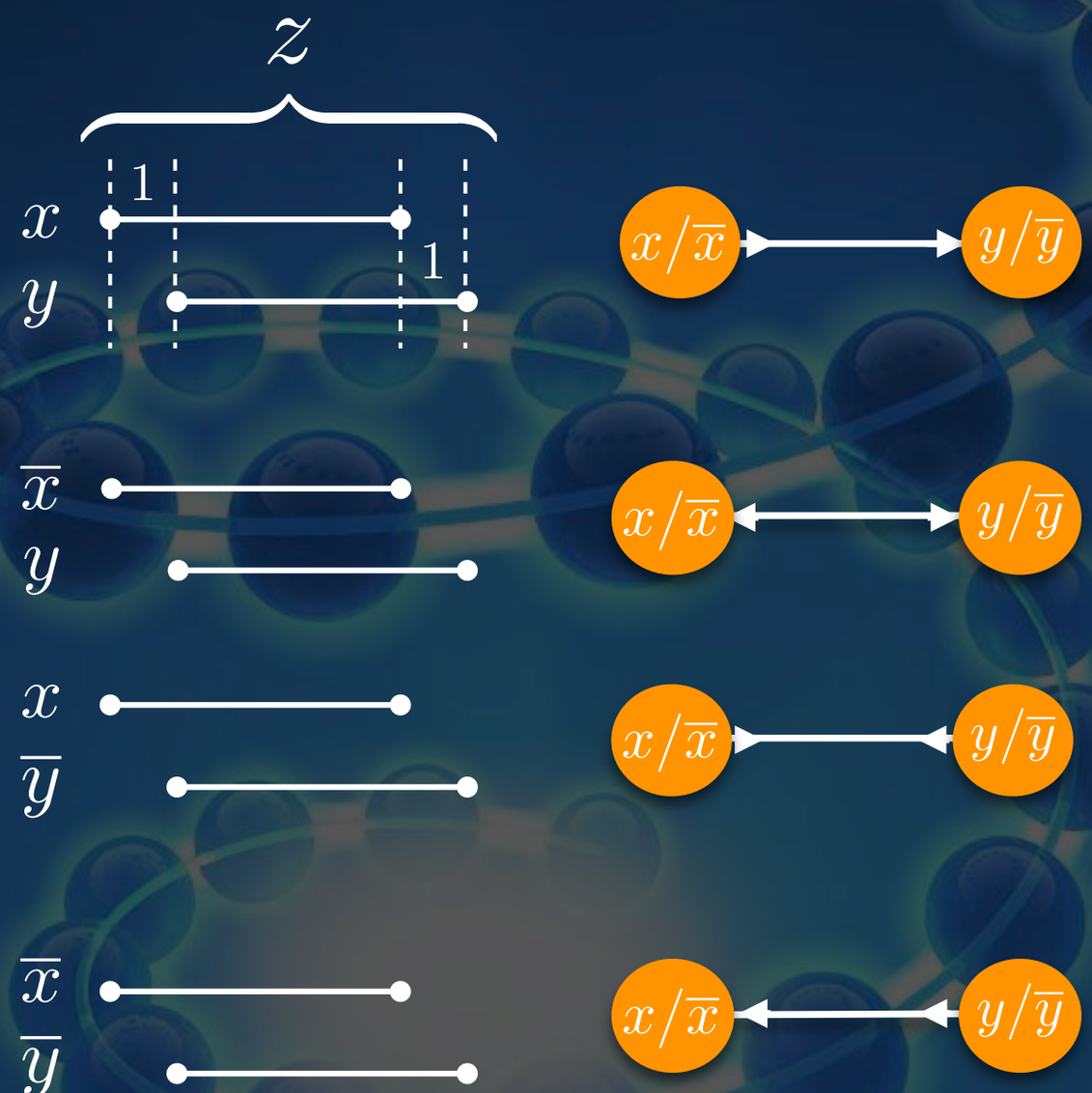
- Un superwalk è un cammino che ha come sottocammini $w(f)$ per ogni $f \in \mathcal{F}$
- Problema BSP:
  - Trovare un superwalk di lunghezza minima
- *Teorema:* Il problema BSP è NP-Arduo per ogni k e per $|\Sigma| \geq 3$

Grafo De Bruijn Bidiretto

- Soluzione al problema dell'orientamento:
  - Si considerano anche i k-mers complementari
  - Si costruisce un grafo bidiretto:
    - $\mathcal{N} = \{k_i / \overline{k_i} | k_i \text{ è un } (k\text{-}1)\text{-mer}\}$
    - Archi hanno orientamento indipendente ad ogni estremità e sono percorribili in entrambi i versi

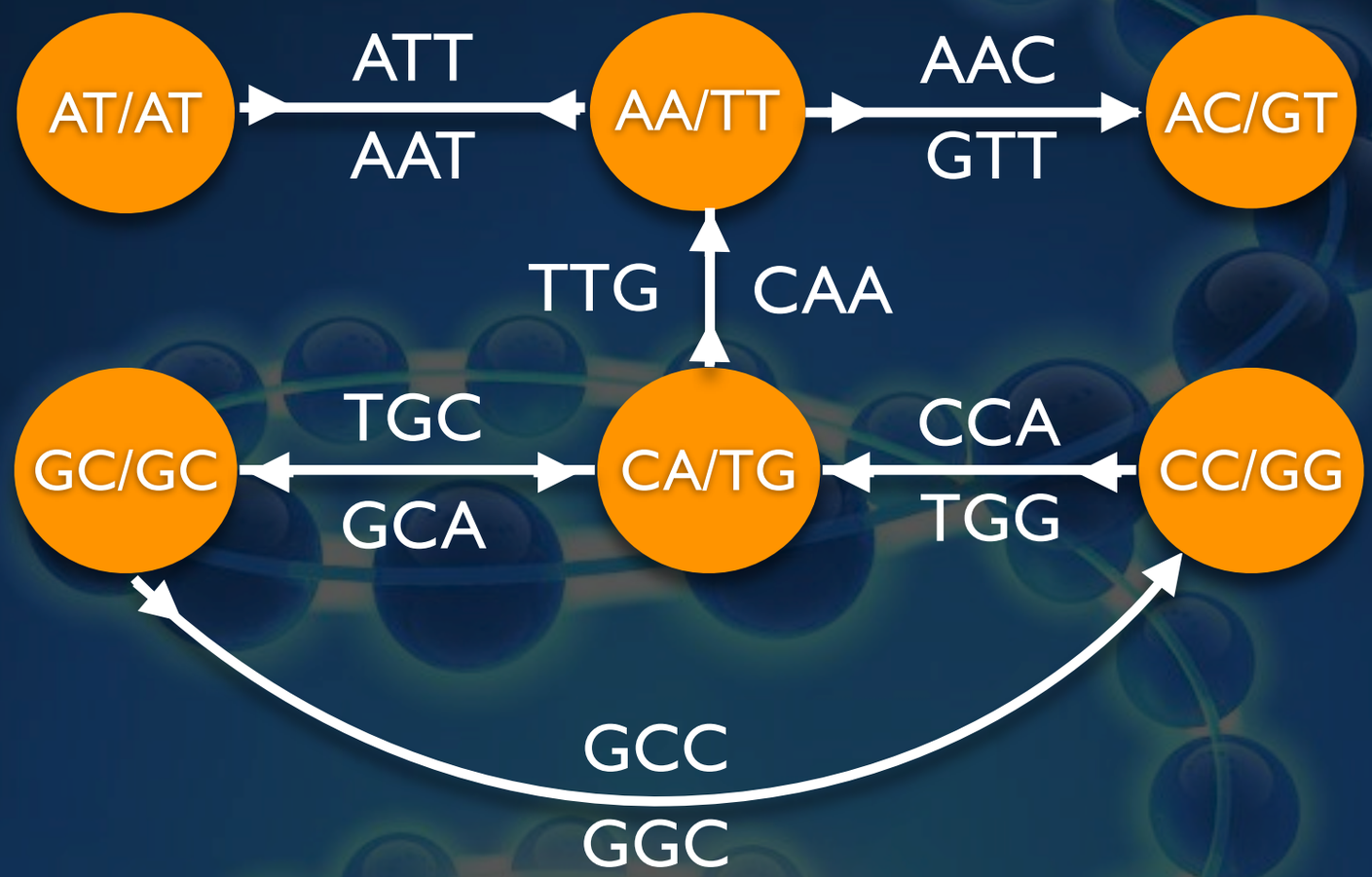
Costruzione degli archi

- Siano $x/\overline{x}, y/\overline{y} \in \mathcal{N}$, costruiamo l'arco per un k-mer z



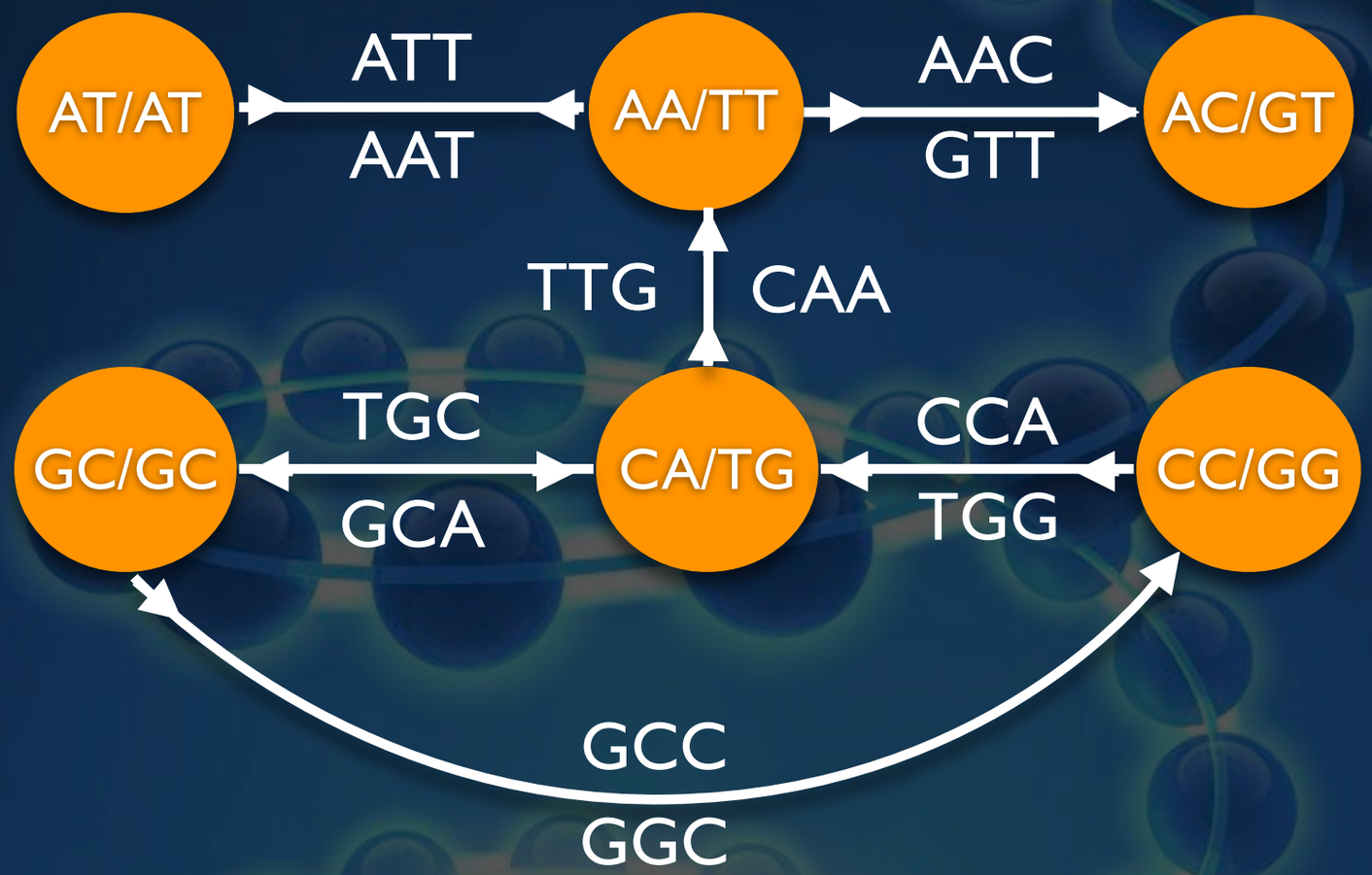
Esempio

ATT/AAT, TTG/CAA, TGC/GCA, GCC/GGC, CCA/TGG, CAA/TTG, AAC/GTT



Esempio

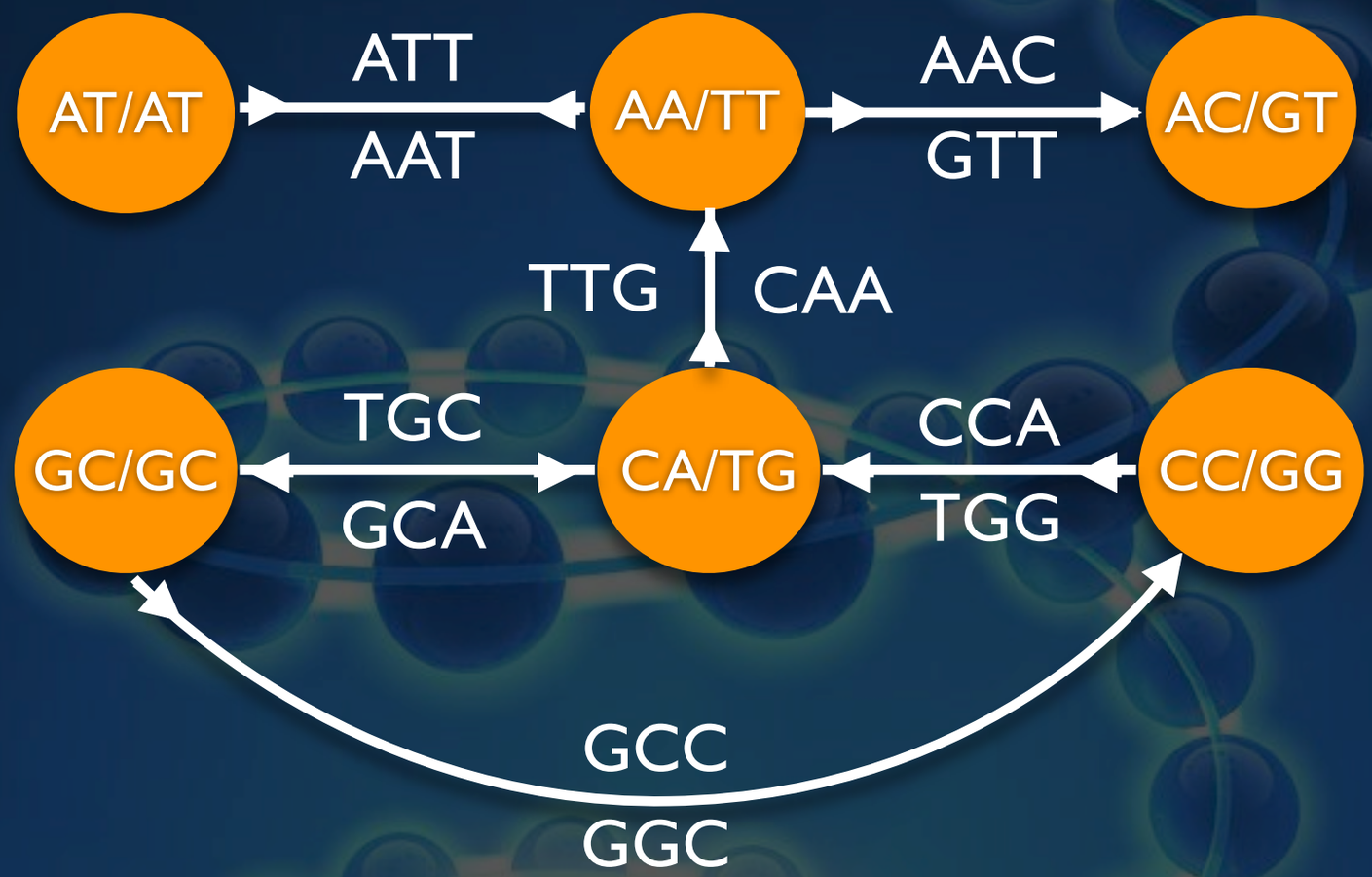
ATT/AAT, TTG/CAA, TGC/GCA, GCC/GGC, CCA/TGG, CAA/TTG, AAC/GTT



Esempio

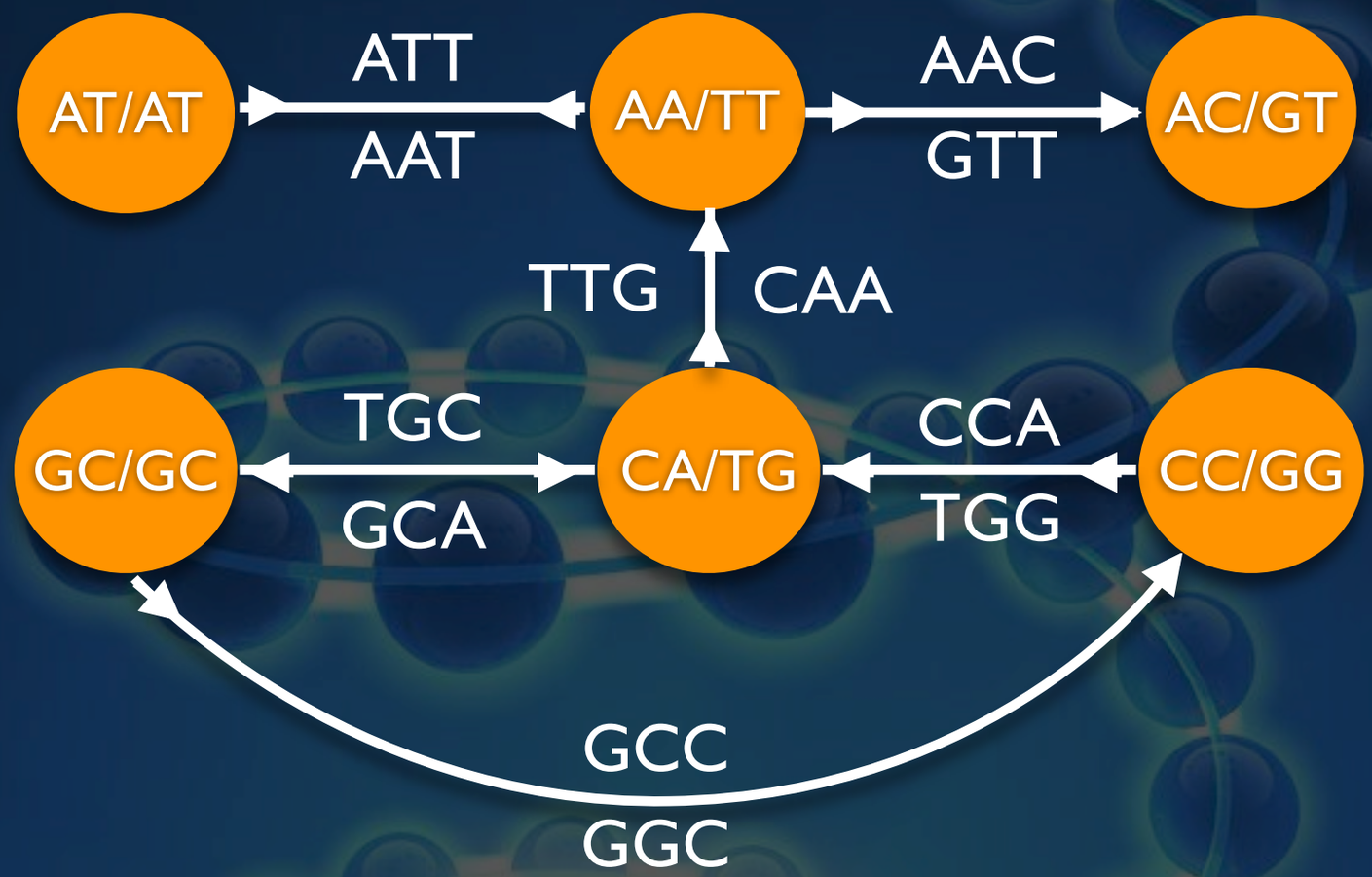
ATT/AAT, TTG/CAA, TGC/GCA, GCC/GGC, CCA/TGG, CAA/TTG, AAC/GTT

\+ / -



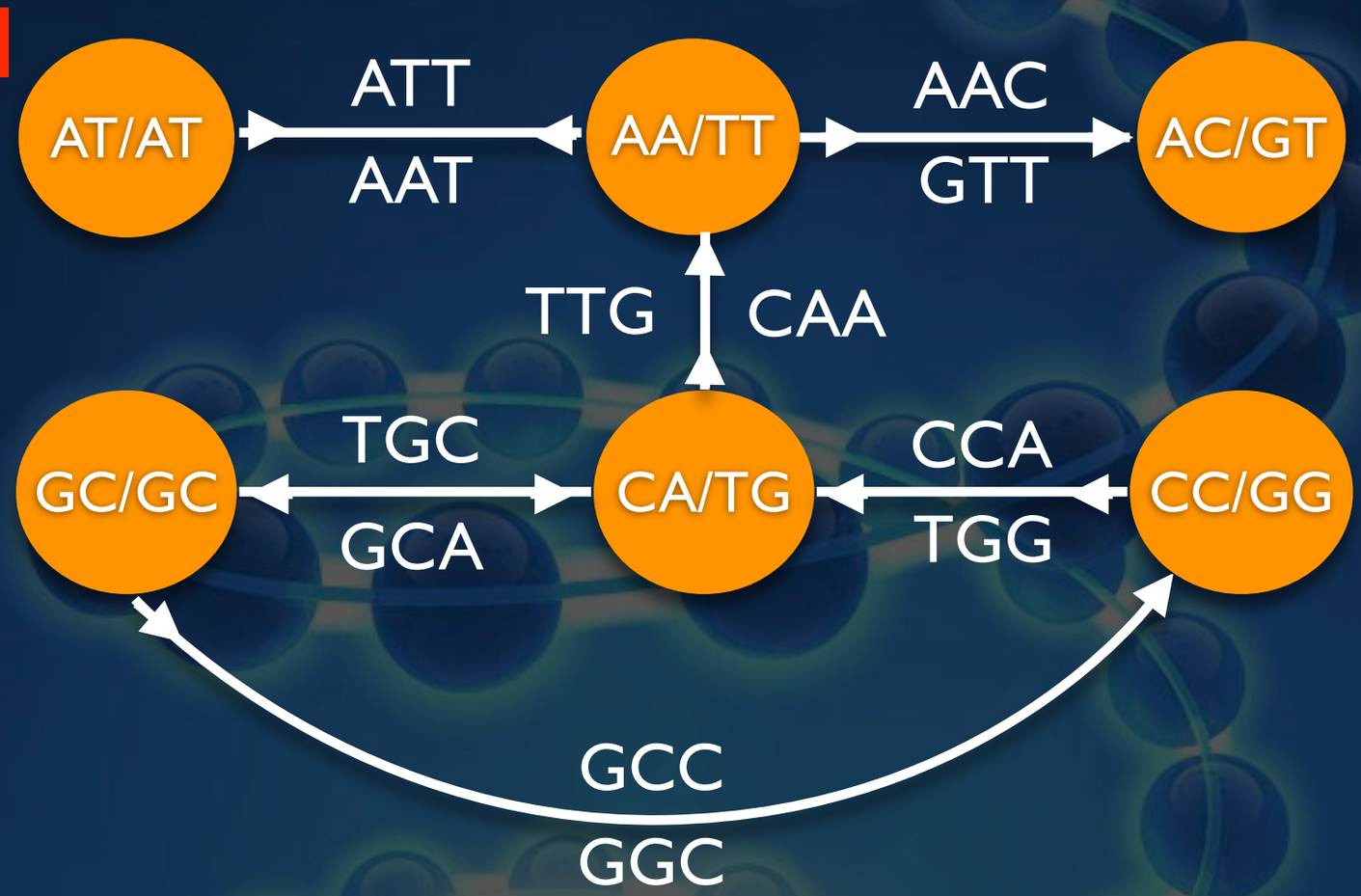
Esempio

ATT/AAT, TTG/CAA, TGC/GCA, GCC/GGC, CCA/TGG, CAA/TTG, AAC/GTT



Esempio

ATT/AAT, TTG/CAA, TGC/GCA, GCC/GGC, CCA/TGG, CAA/TTG, AAC/GTT

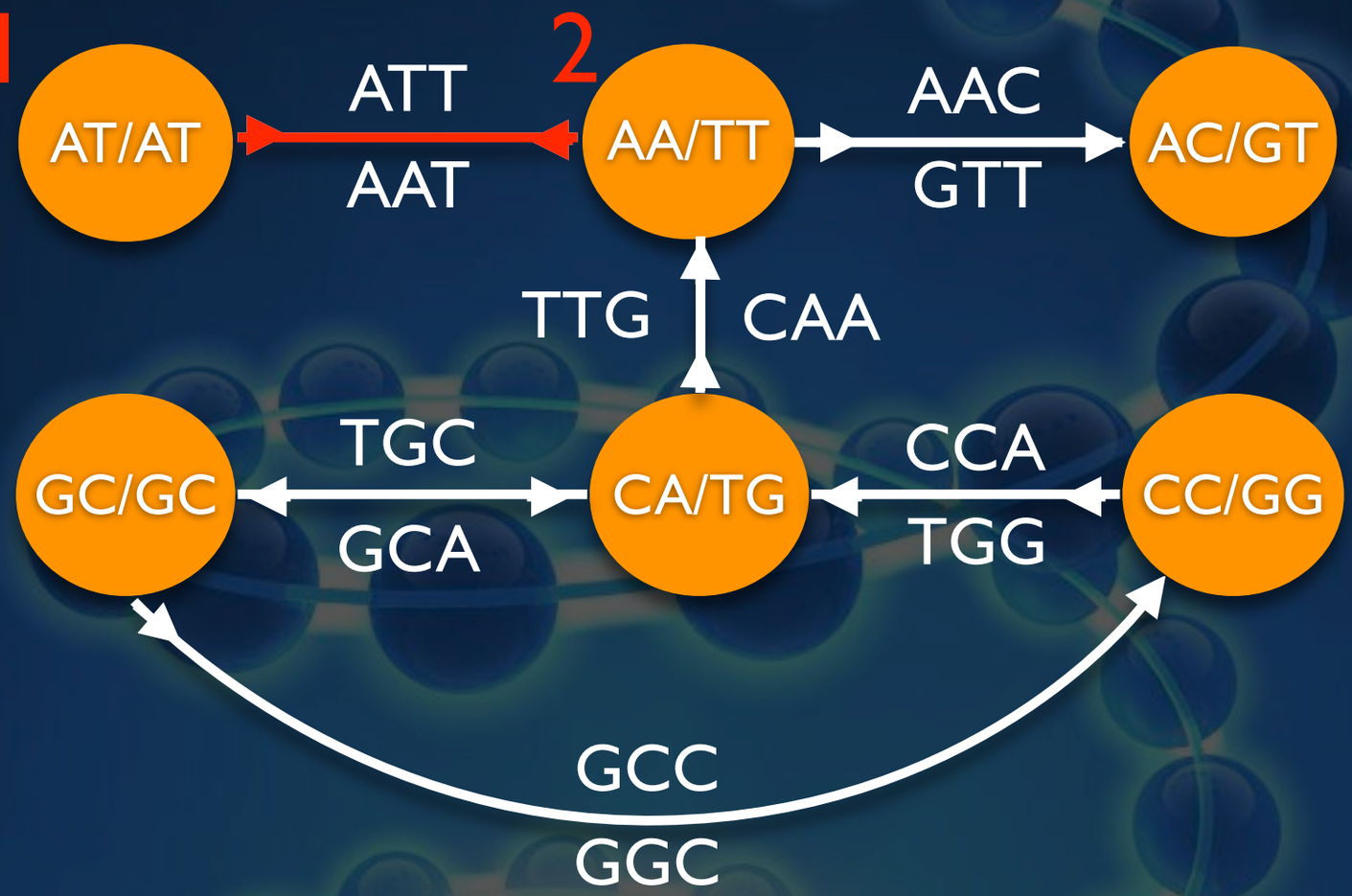


A T

A T

Esempio

ATT/AAT, TTG/CAA, TGC/GCA, GCC/GGC, CCA/TGG, CAA/TTG, AAC/GTT

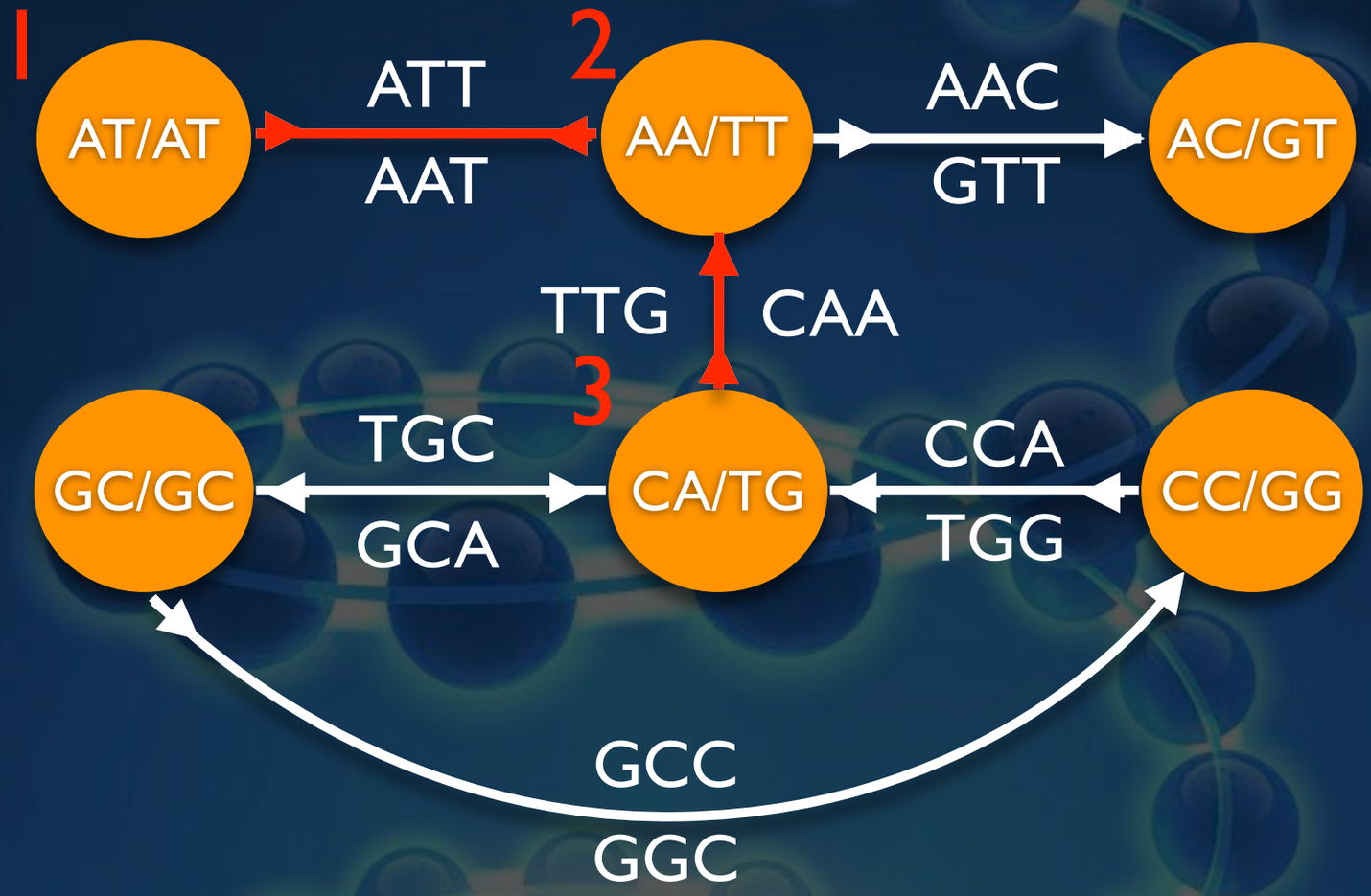


A T T

A A T

Esempio

ATT/AAT, TTG/CAA, TGC/GCA, GCC/GGC, CCA/TGG, CAA/TTG, AAC/GTT

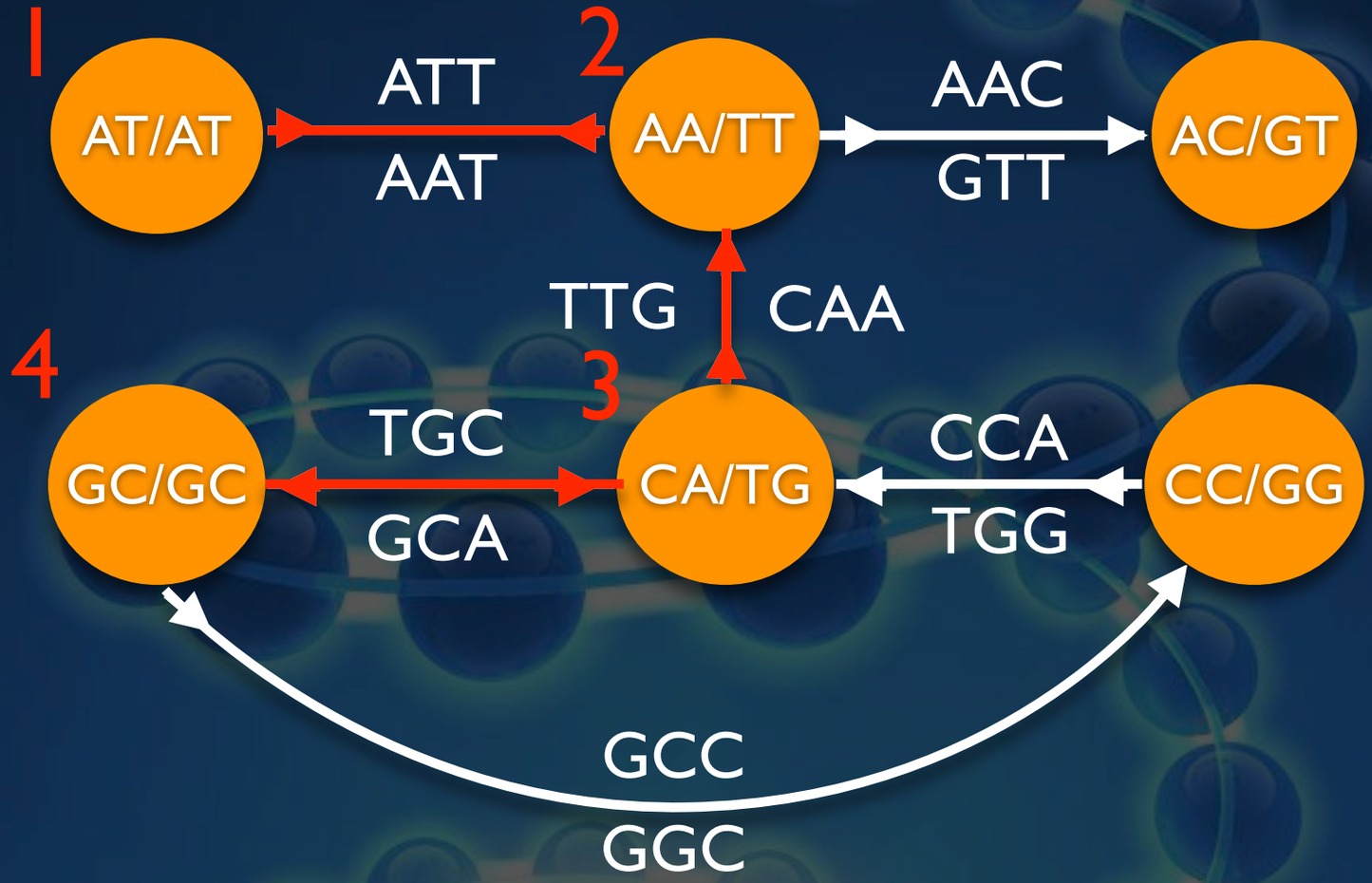


A T T G

C A A T

Esempio

ATT/AAT, TTG/CAA, TGC/GCA, GCC/GGC, CCA/TGG, CAA/TTG, AAC/GTT

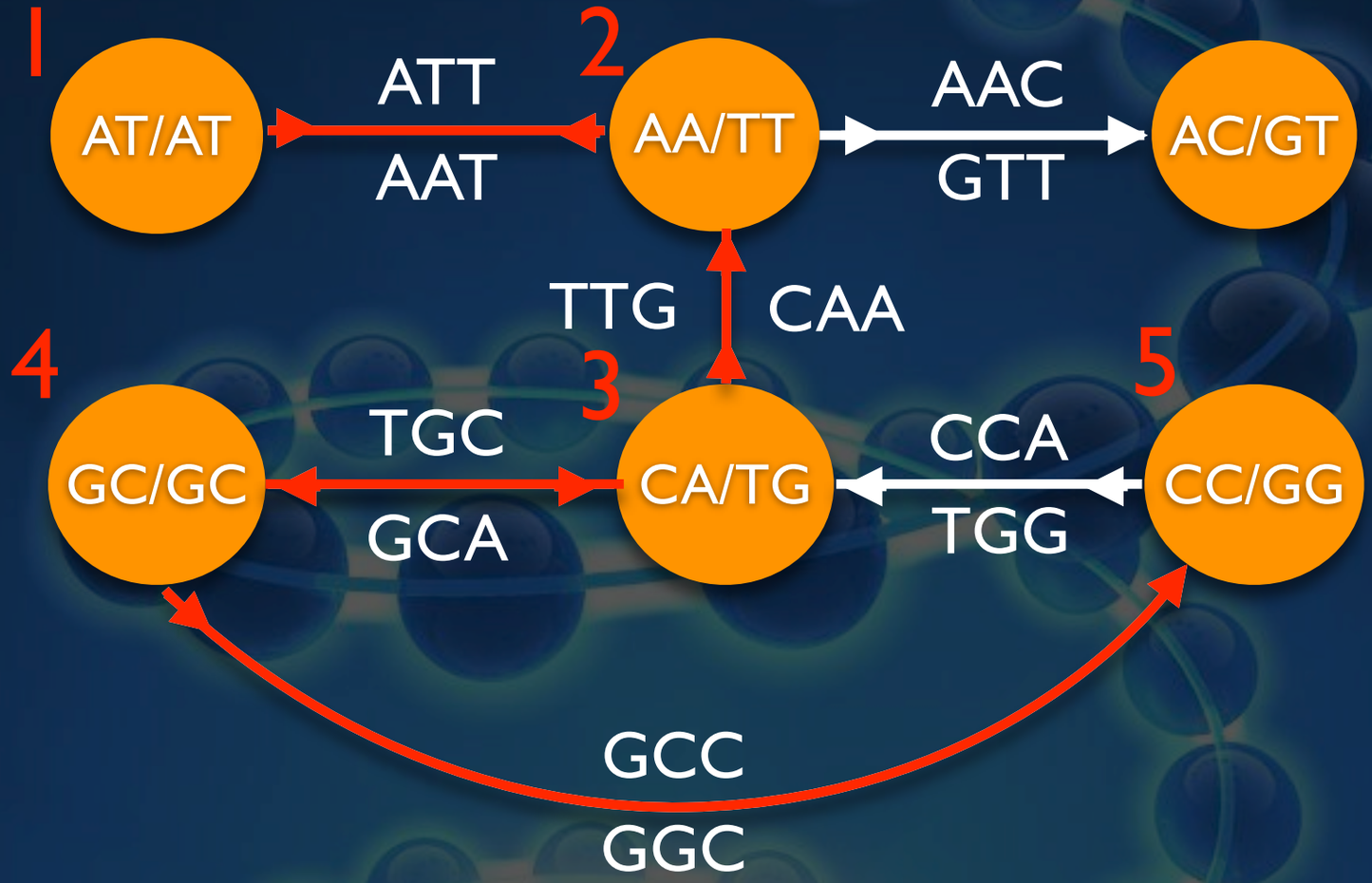


A T T G C

G C A A T

Esempio

ATT/AAT, TTG/CAA, TGC/GCA, GCC/GGC, CCA/TGG, CAA/TTG, AAC/GTT

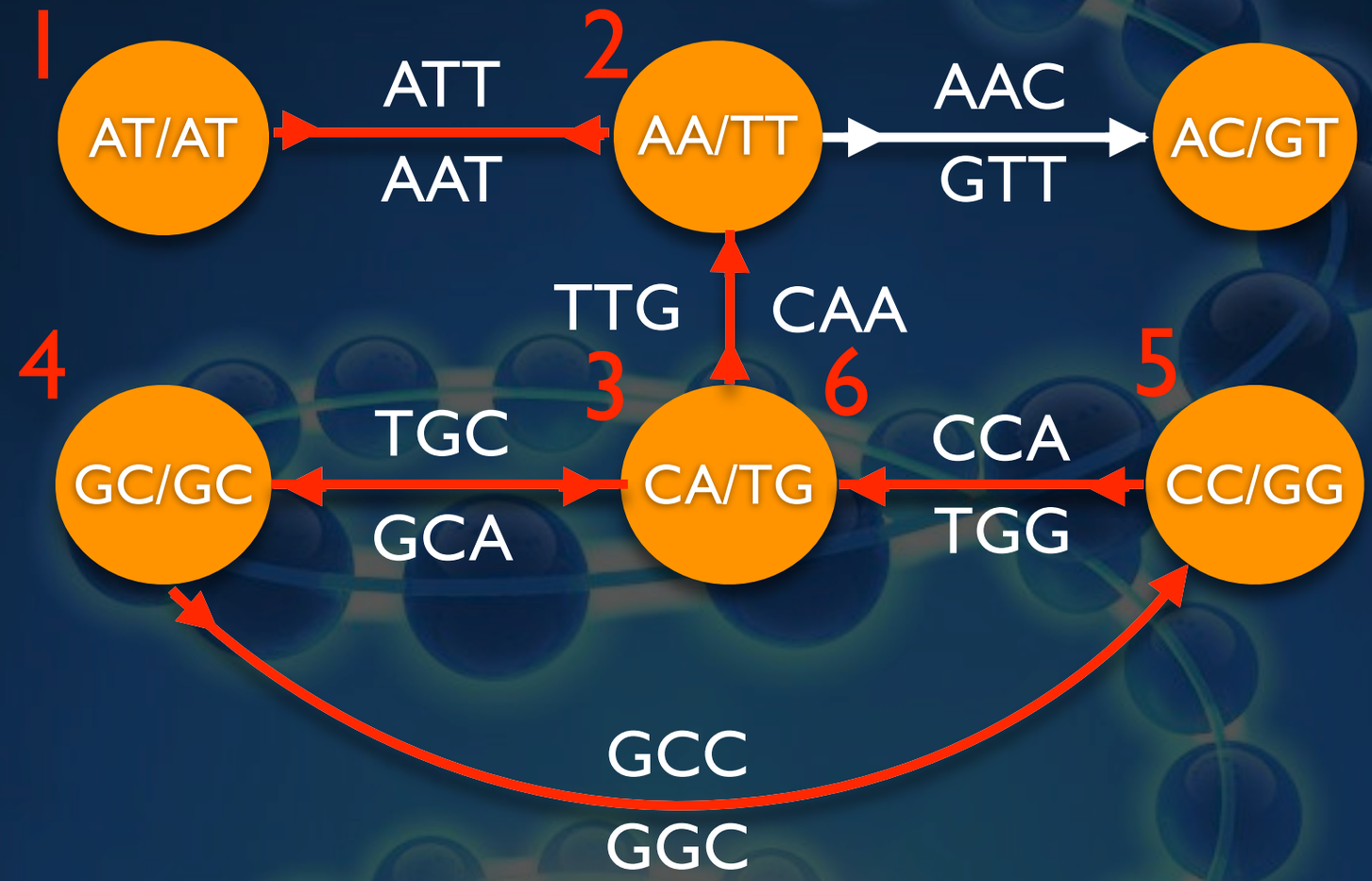


A T T G C C

G G C A A T

Esempio

ATT/AAT, TTG/CAA, TGC/GCA, GCC/GGC, CCA/TGG, CAA/TTG, AAC/GTT

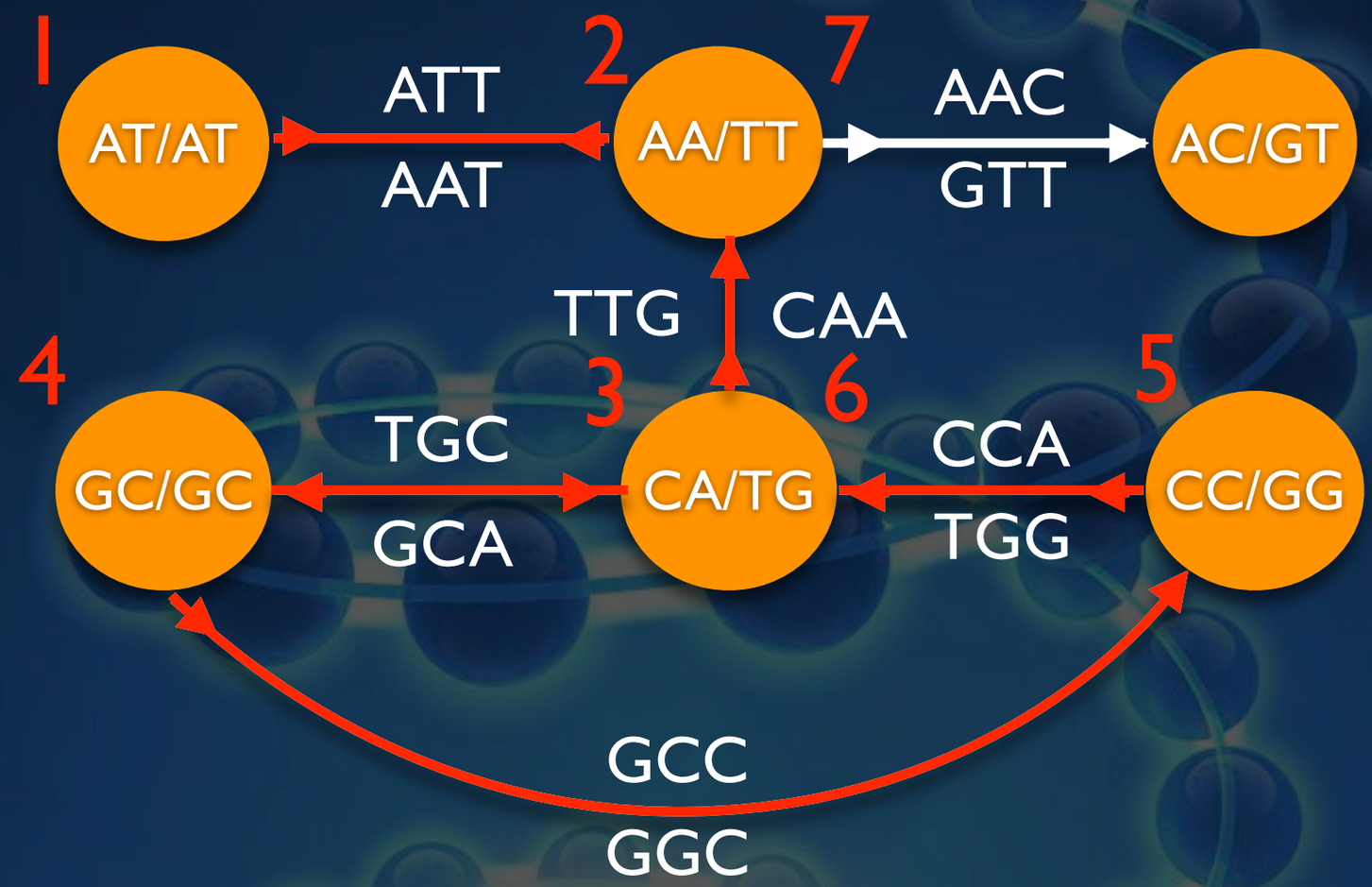


A T T G C C A

T G G C A A T

Esempio

ATT/AAT, TTG/CAA, TGC/GCA, GCC/GGC, CCA/TGG, CAA/TTG, AAC/GTT

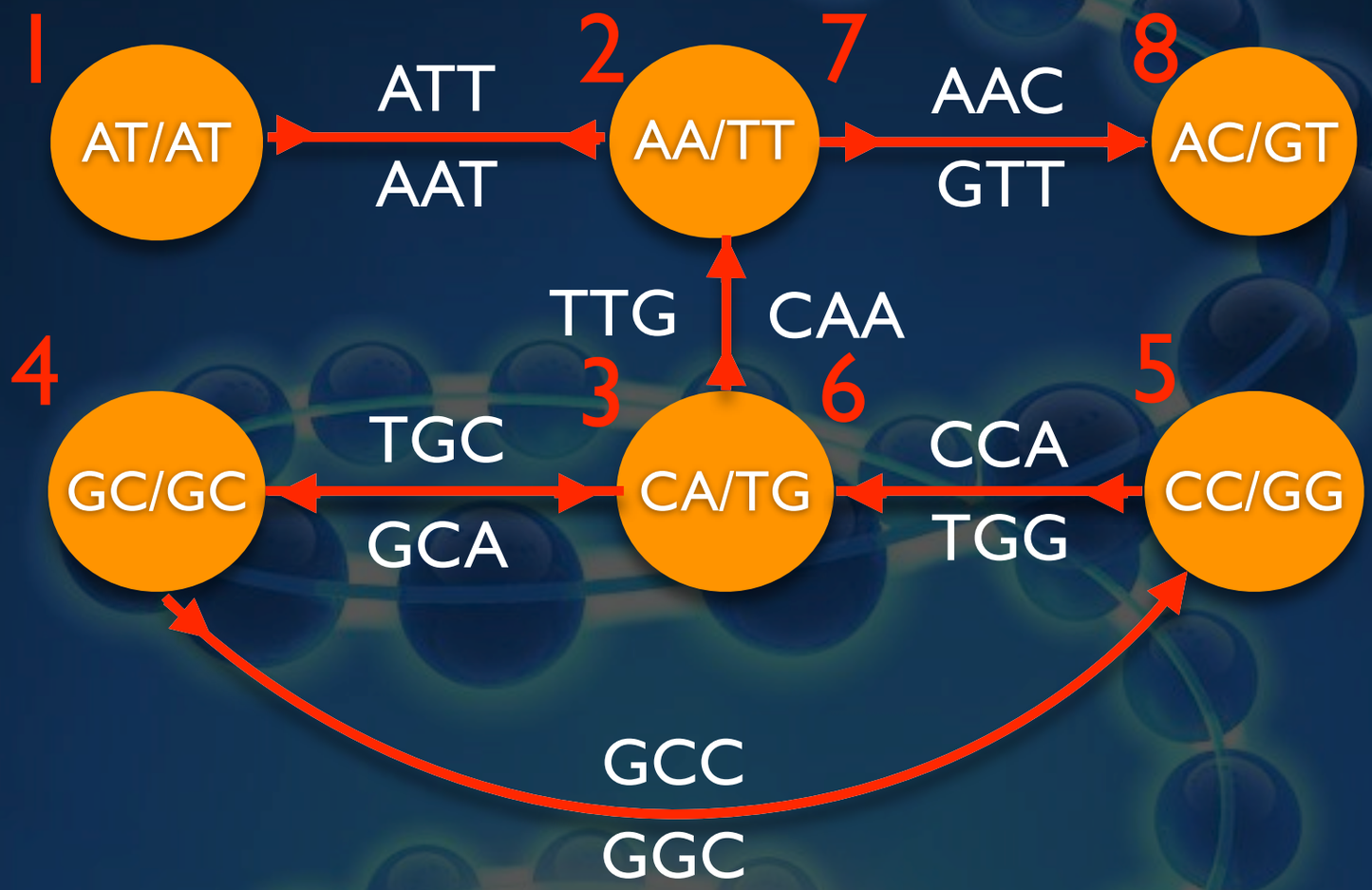


A T T G C C A A

T T G G C A A T

Esempio

ATT/AAT, TTG/CAA, TGC/GCA, GCC/GGC, CCA/TGG, CAA/TTG, AAC/GTT



A T T G C C A A C
G T T G G C A A T

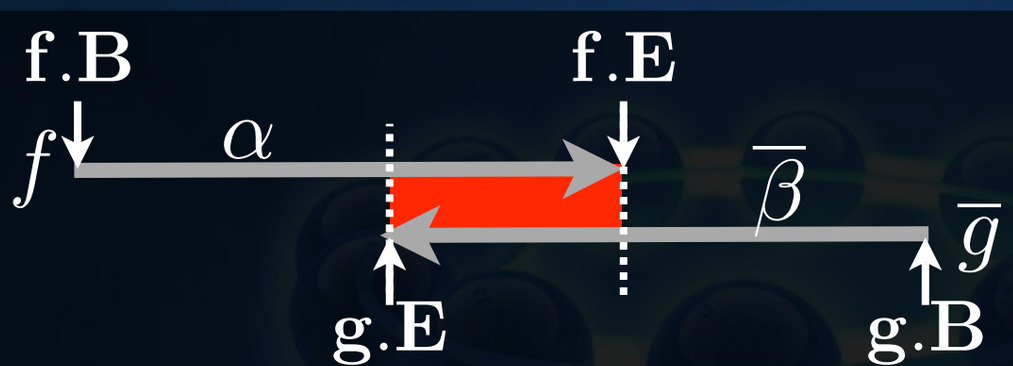
Complessità

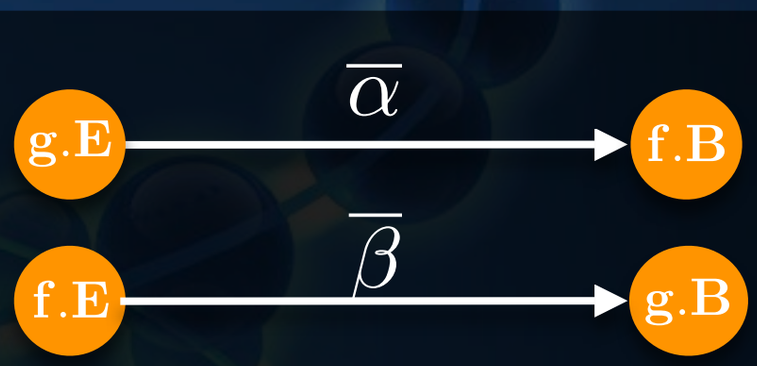
- Chinese walk = Cammino che attraversa tutti gli archi almeno una volta
- Chinese Postman Problem:
  - Trovare un Chinese walk di costo minimo (Chinese Postman Tour)
- Chinese Postman Tour corrisponde al più breve assembly di tutti i k-mer
- $\mathrm{CPP} \in \mathcal{P}$ per grafi bidiretti

String Graph

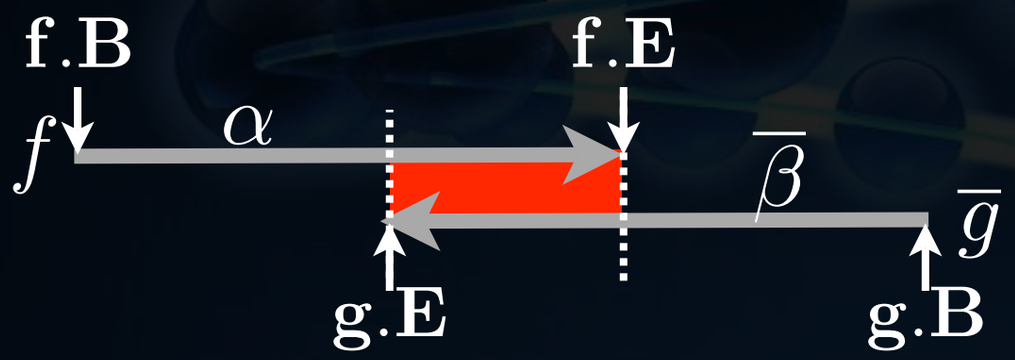
- Assunzioni:
  - No contaminazioni/chimere
  - Collezione substring-free
  - Errori al di sotto di una soglia ϵ (2.5%)
- Grafo:
  - $\mathcal{N} = \{f.B, f.E | f \in \mathcal{F}\}$
  - Per ogni $f, g \in \mathcal{F}$ che hanno un overlap (massimale) si hanno due archi orientati etichettati con le sequenze non in comune

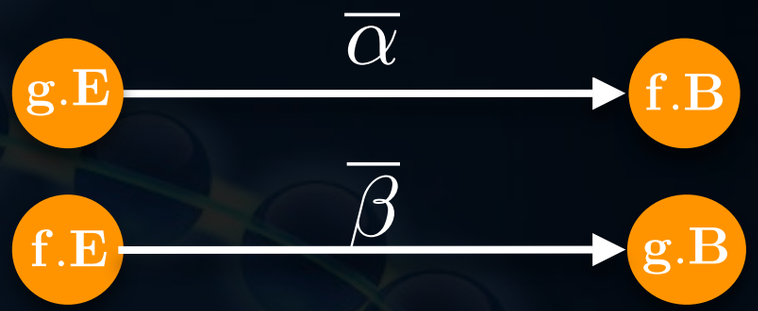
Archi

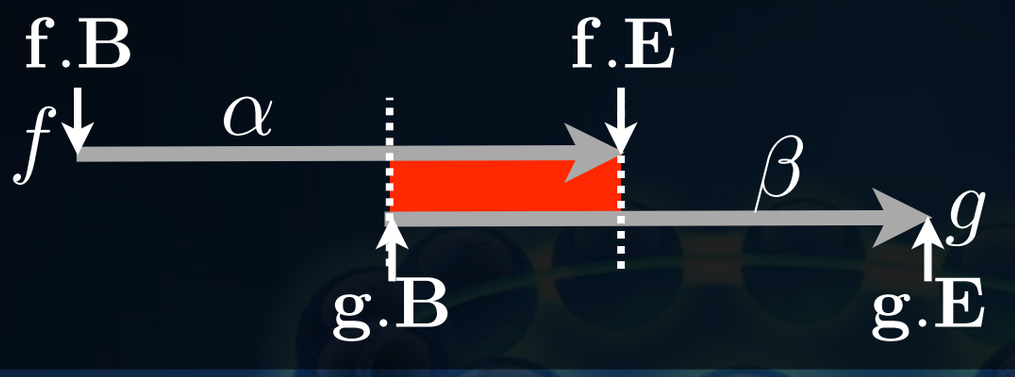


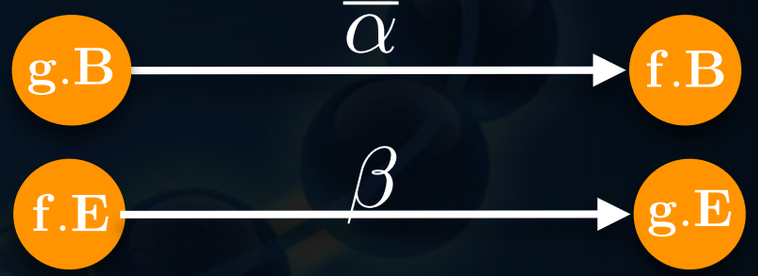


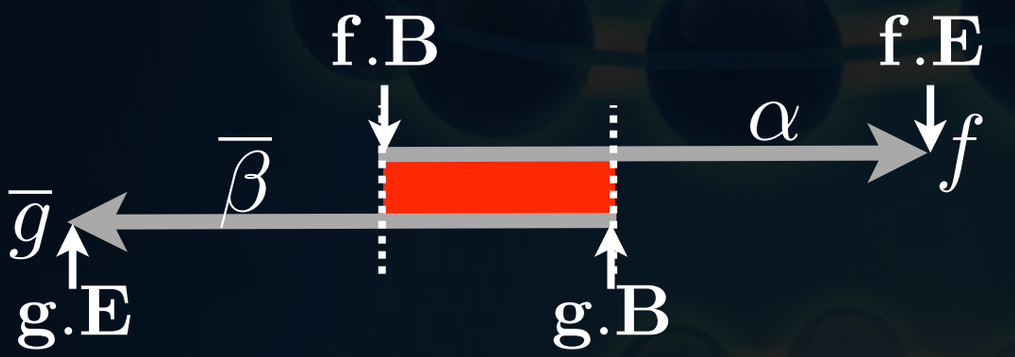
Archi

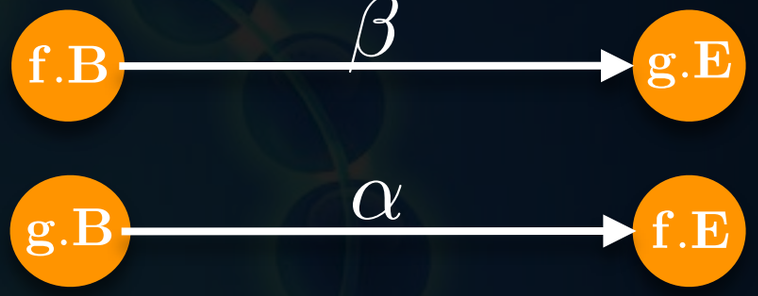

f.B
f.E
f
α
β
g
g.E
g.B

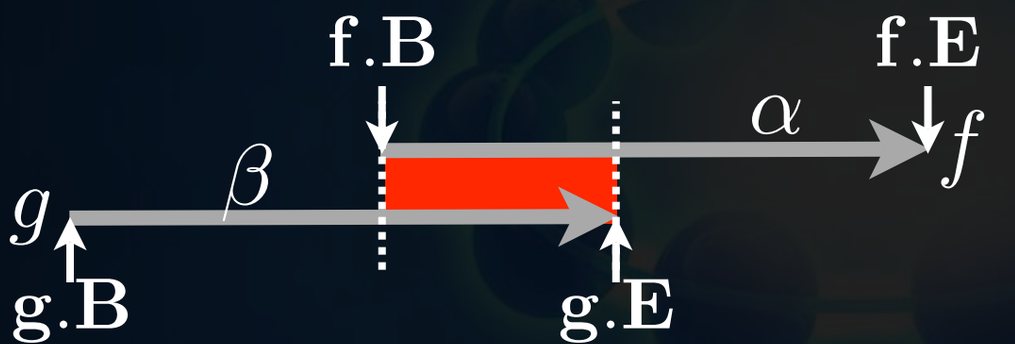

g.E
α
f.B
f.E
β
g.B

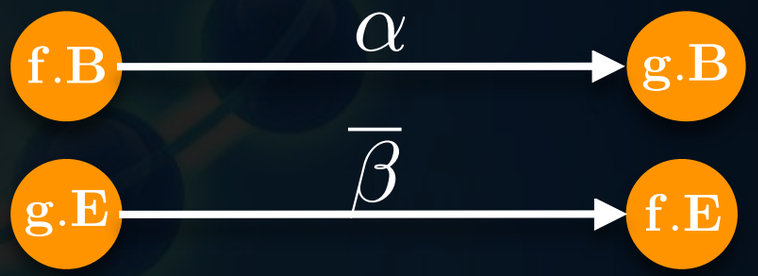

f.B
f.E
f
α
β
g
g.B
g.E


g.B
α
f.B
f.E
β
g.E


f.B
f.E
α
f
β
g
g.E
g.B


f.B
β
g.E
g.B
α
f.E

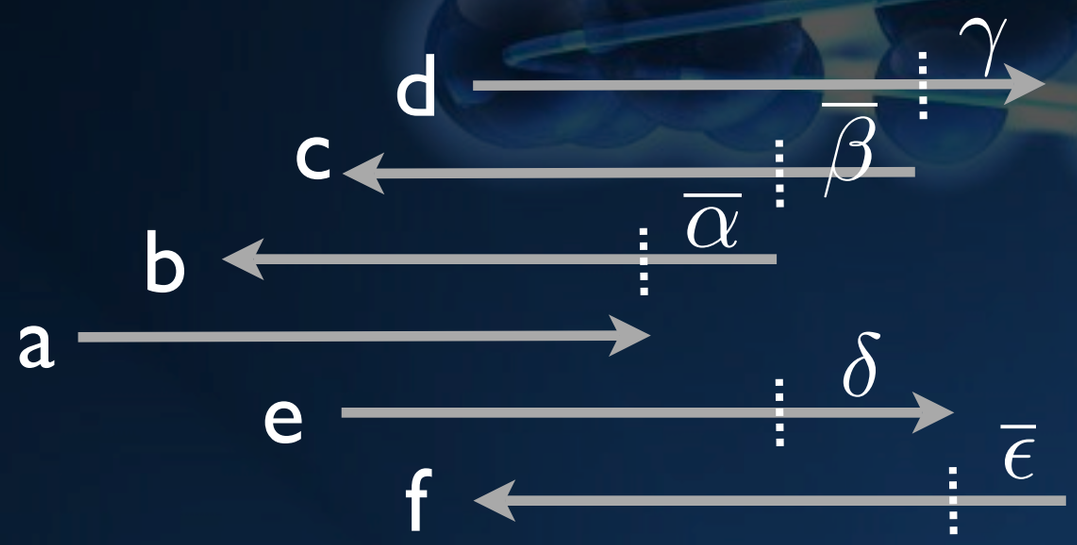

f.B
f.E
α
f
β
g
g.B
g.E


f.B
α
g.B
g.E
β
f.E

Cammini

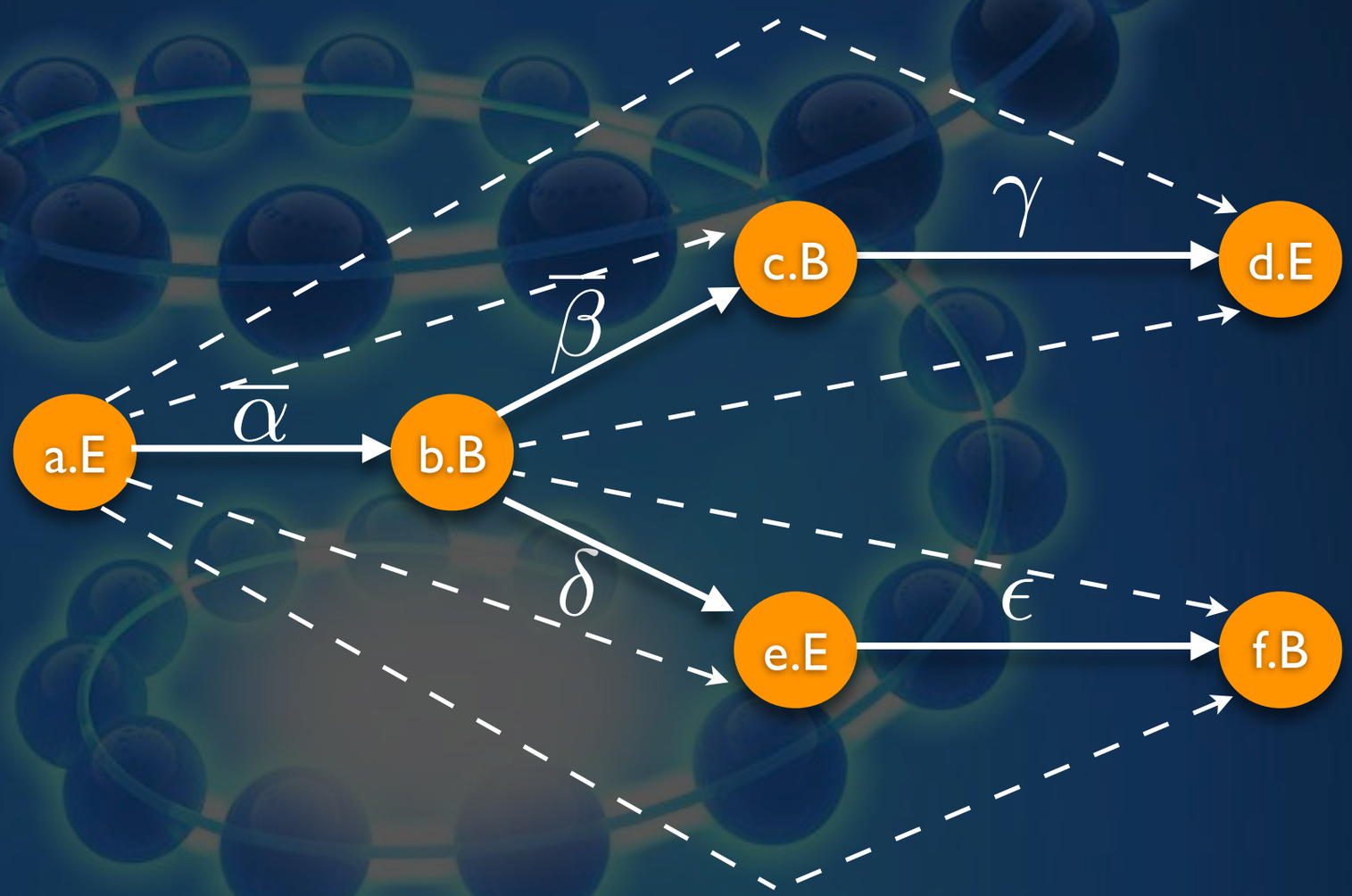
- Ogni cammino $v_1, v_2, \ldots, v_m$ rappresenta una sequenza contigua tale che:
  - Se $v_i = f_k.E$ il frammento f_k partecipa nel verso originario, altrimenti complementato
  - E' formata dalla sequenza di v_1 (eventualmente complementata) seguita dalle etichette degli archi attraversati dal cammino
  - E' un assemblaggio valido (read-coherence)

Esempio

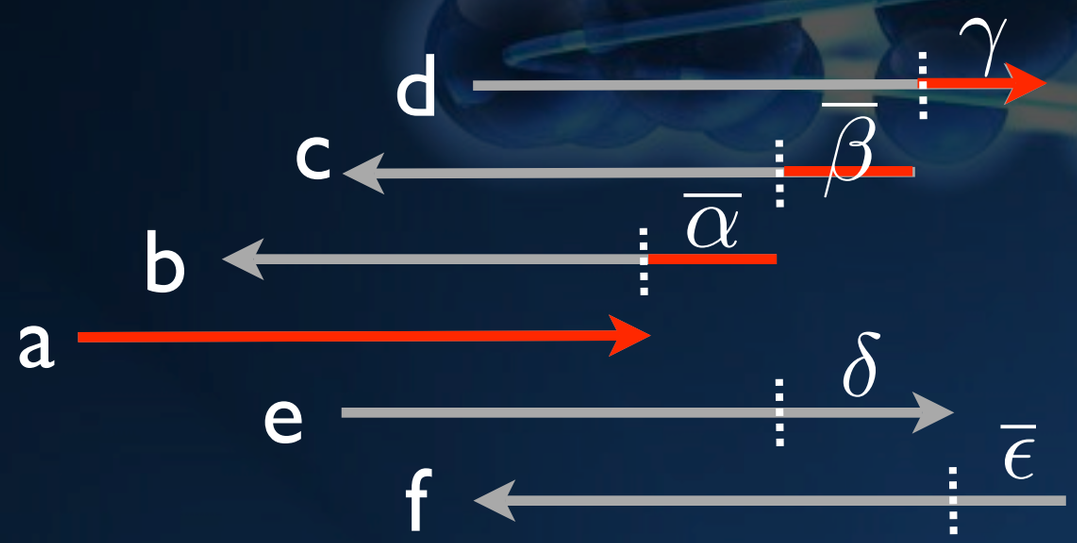


Costruzione solo per la porzione non overlapping a destra

Archi transitivi

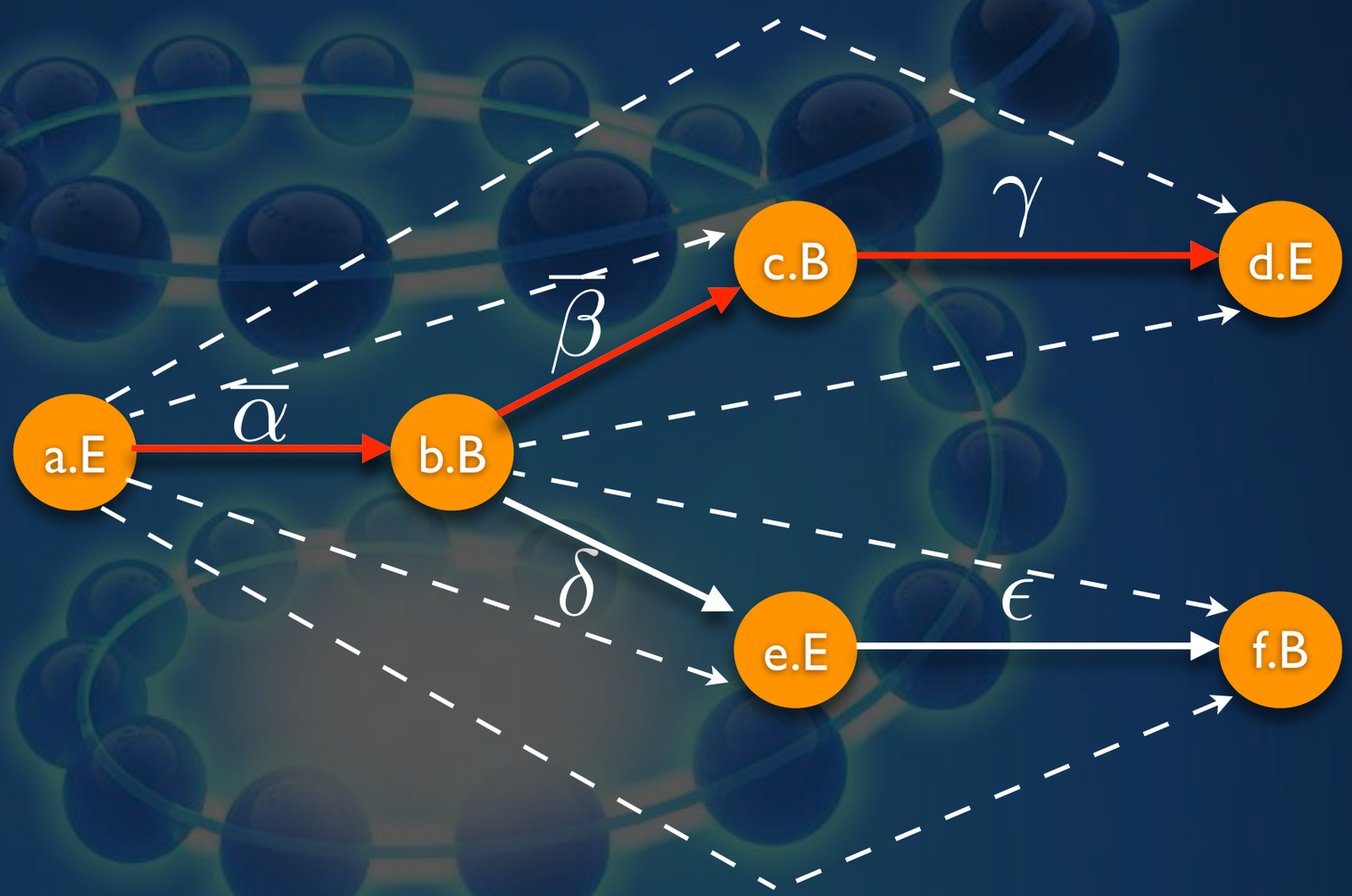


Esempio



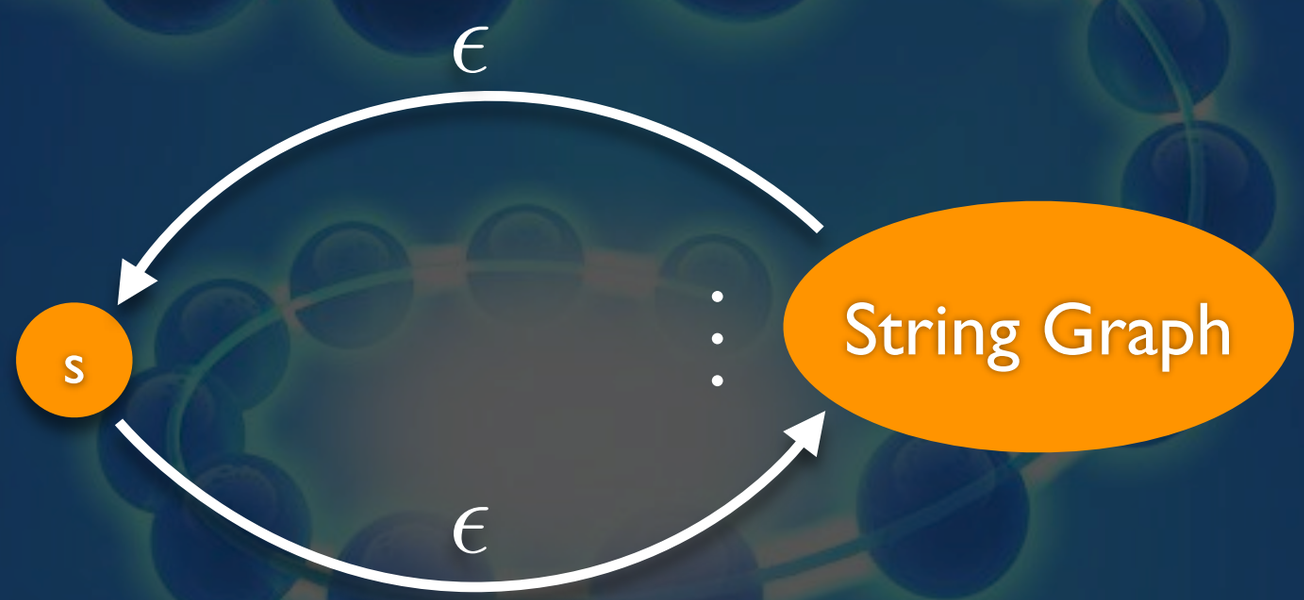
Costruzione solo per la porzione non overlapping a destra

Archi transitivi



Mancanza di Copertura

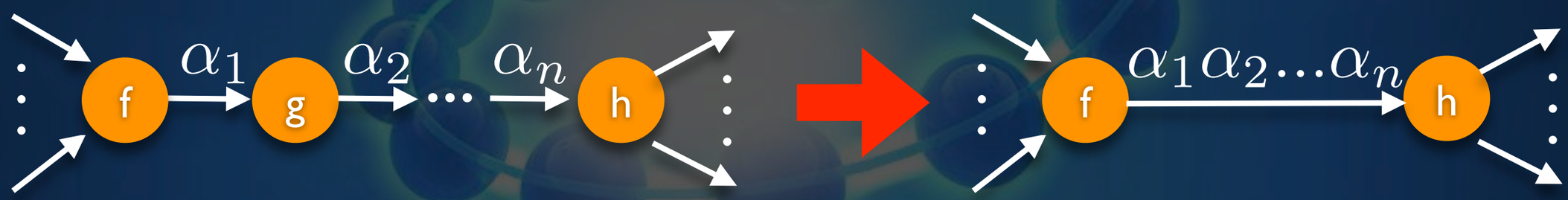
- Nodi con nessun arco uscente o entrante indicano mancanza di copertura
- Soluzione:
  - Si aggiunge un nodo fittizio s e per ogni nodo di giunzione v si aggiungono gli archi $(v, s), (s, v)$ etichettati con ϵ



Ottimizzazioni

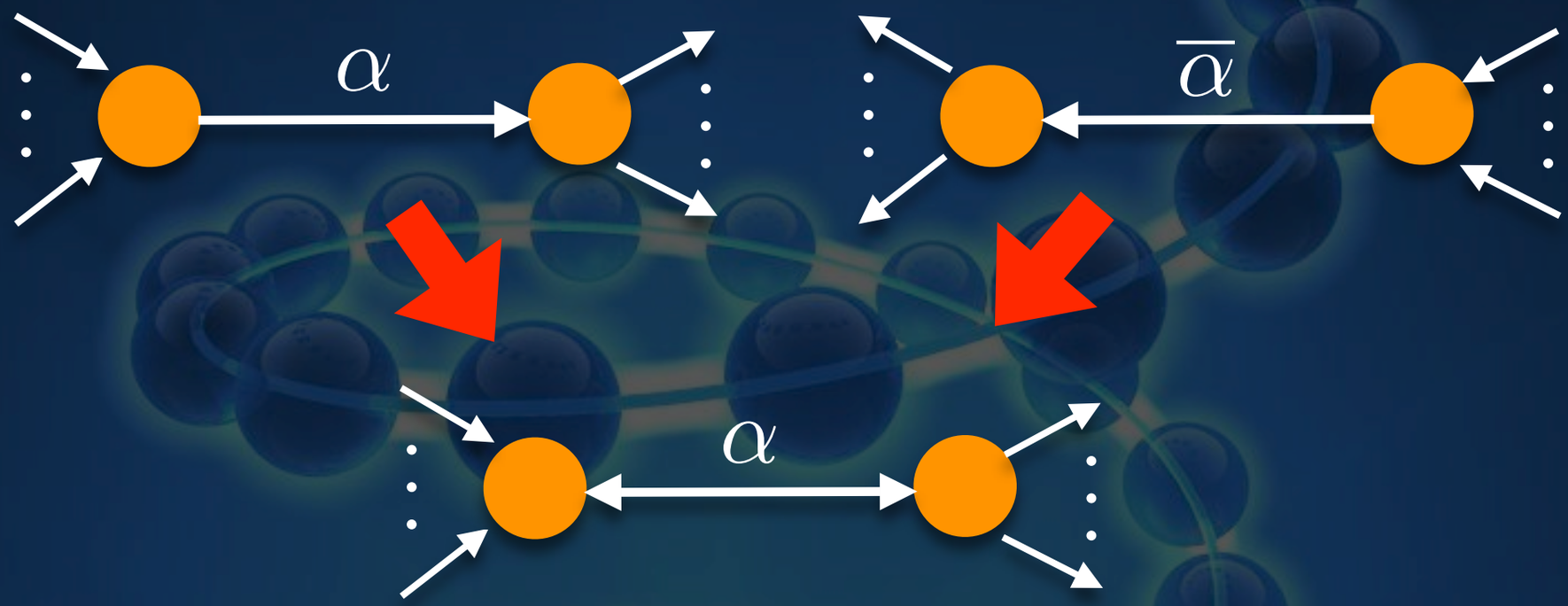
- Se c'è un overlap tra f e g, tra g e h e tra f ed h, l'arco (f, h) può essere eliminato

- Collassamento delle catene di archi che iniziano e finiscono con un nodo di giunzione e contengono solo nodi con un arco entrante e uscente



Ottimizzazioni (2)

- Unificazione di archi complementari in archi bidiretti



- Guadagno da 2 a 3 ordini di grandezza sul numero originario di frammenti e overlap

Funzione di Selezione

- Tramite procedimenti statistici si costruisce una funzione s che ad ogni arco e associa
  - $l(e)$ = numero minimo di attraversamenti
  - $u(e)$ = numero massimo di attraversamenti

Tipo	$l(e)$	$u(e)$
Required	1	∞
Optional	0	∞
Exact	1	1

Minimum s-Walk Problem

- Un s-Walk è un cammino che attraversa gli archi rispettando la funzione di selezione
- Problema MSWP:
  - Trovare un s-Walk di lunghezza minima
- *Teorema:* Il problema MSWP è NP-Arduo
- E' possibile costruire una funzione di selezione più specifica che restituisce un *traversal count*

String Graph vs De Bruijn

1. Preprocessing:
    - Il grafo di De Brujin necessita dei k-mers
    - Lo String Graph vuole una collezione substring-free
2. Read-coherence:
    - De Bruijn non è read-coherent, molte risorse impiegate per renderlo tale
    - String Graph lo è per costruzione

String Graph vs De Bruijn (2)

3. Orientamento:
   - Grafo di De Bruijn aggiunge k-mers complementari
   - String Graph lo risolve per costruzione
4. Ripetizioni:
   - Grafo di De Bruijn isola ripetizioni
   - String Graph localizza e quantifica le ripetizioni classificando gli archi

String Graph vs De Bruijn (3)

5. Errori
   - Grafo De Bruijn diventa un “groviglio” di nodi
   - String Graph li gestisce per costruzione
6. Mancanza di Copertura:
   - Grafo De Bruijn non se ne occupa
   - String Graph la gestisce aggiungendo un nodo fittizio

String Graph vs De Bruijn (4)

7. Complessità:
   - BSP e MSWP sono NP-Ardui
     - Possibilità di risoluzione efficiente se si considerano particolari classi di input
   - Grafo De Bruijn bidiretto è polinomiale