I 7 ponti di Kaliningrad
o
Come assemblare un genoma

Fabrizio Mafessoni, Universita' di Trieste, Corso di Genomica Computazionale 2025/2026
Laurea Specialistica in Genomica Funzionale

Scopo ultimo del corso: saper partire da sequenze di DNA fino ad assemblare interi genomi e caratterizzarli funzionalmente

Dal sequenziatore..

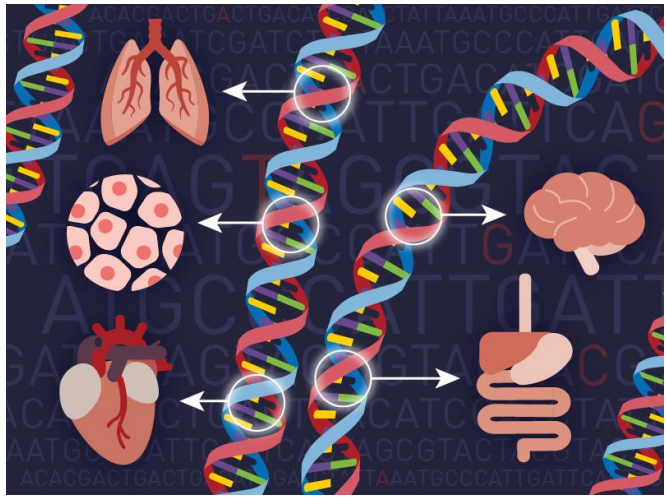
..al genoma..

..alla funzione

sequenze

assemblaggio

mappatura
e
genotipizzazione



Formati di file bioinformatici

.fastq

.fasta, .bam, .vcf

.bed, .gff

L’ assemblaggio

Dal sequenziatore..

sequenze

assemblaggio

..al genoma

Formati di file bioinformatici

.fastq

.fasta

Cosa vedremo oggi

- Perché è difficile assemblare
- Algoritmi per assemblare:
  - Overlap-graphs
  - De Brujin graphs
- Concetti generali delle scienze computazionali (“Teoria dei grafi”, algoritmi polinomiali ed intrattabili)
- Applicazioni specifiche a tipologie di dati di sequenziamento e problemi dei “genomi veri”
  - Il problema delle ripetizioni
  - Scaffolding
  - Assemblare con sequenze lunghe e corte

La prossima lezione

- Pratica di assemblaggio di gneomi con strumenti bioinformatici

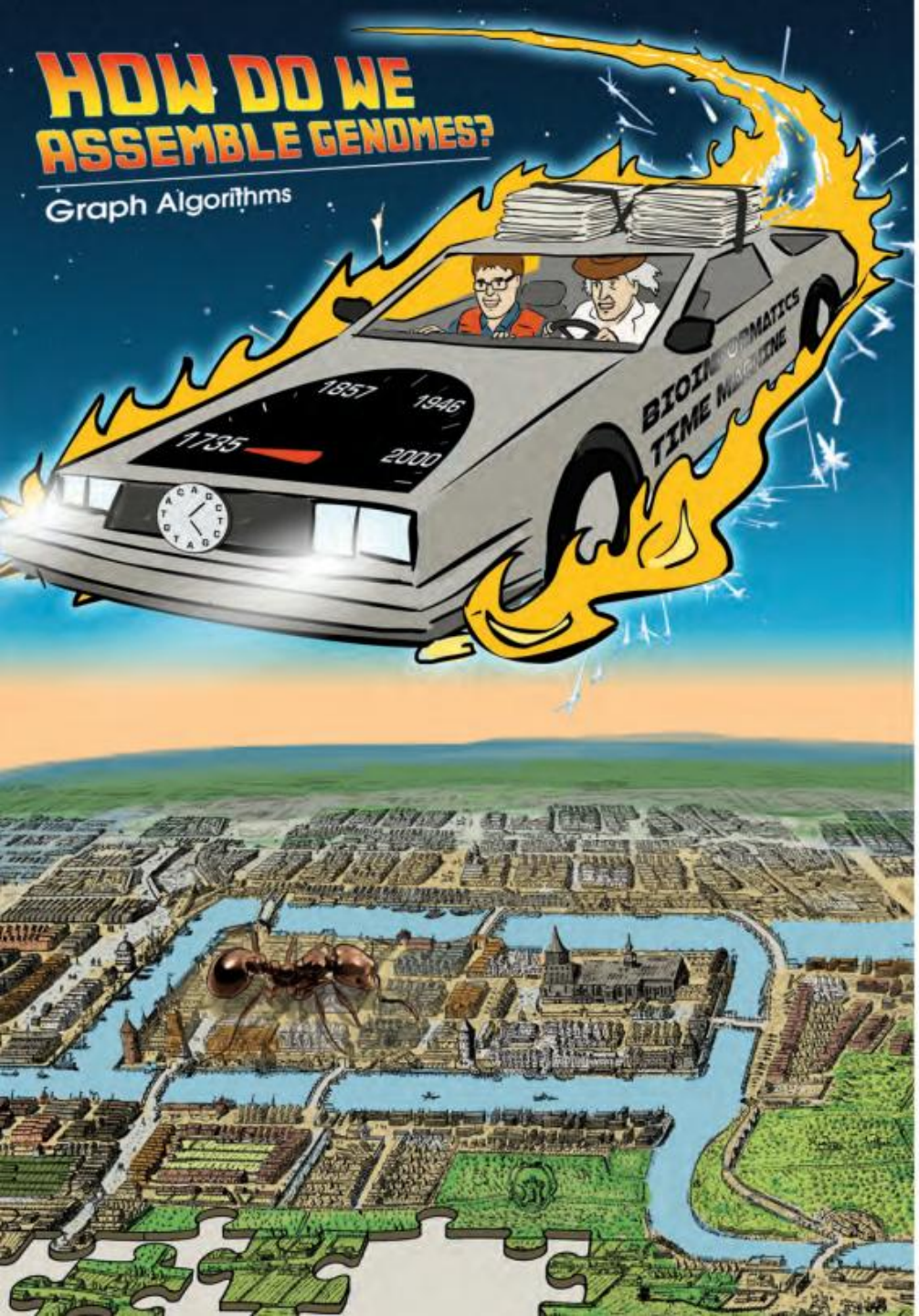
Cosa vedremo oggi

- Perché è difficile assemblare
- Algoritmi per assemblare:
  - Overlap-graphs
  - De Brujin graphs
- Concetti generali delle scienze computazionali (“Teoria dei grafi”, algoritmi polinomiali ed intrattabili)
- Applicazioni specifiche a tipologie di dati di sequenziamento e problemi dei “genomi veri”
  - Il problema delle ripetizioni
  - Scaffolding
  - Assemblare con sequenze lunghe e corte

Importanti in altri ambiti, ad esempio pangenomica, genomica comparativa (ricostruzione di sintenia e ortologia), reti metaboliche e di regolazione genica, struttura 3D del genoma, etc.

La prossima lezione

- Pratica di assemblaggio di gneomi con strumenti bioinformatici



Materiale didattico

La lezione è basata su questo capitolo dal libro «Bioinformatics Algorithms» e un paio di review allegate su Moodle.

Li e Durbin, 2024, Nature Reviews Genetics

Rise e Green, 2019, Annu.Rev.Anim.Bio.

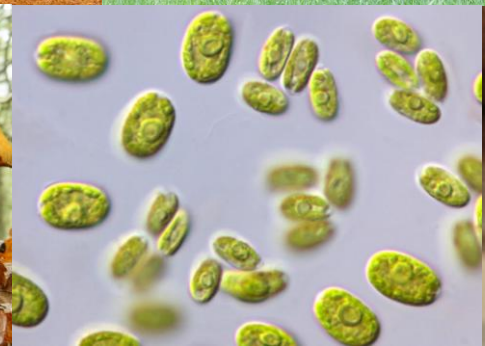



© Burrard-Lucas.com








Argomento difficile..ma alla base degli sforzi internazionali per comprendere tutto questo

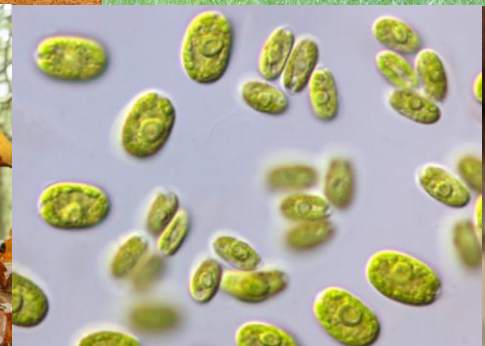












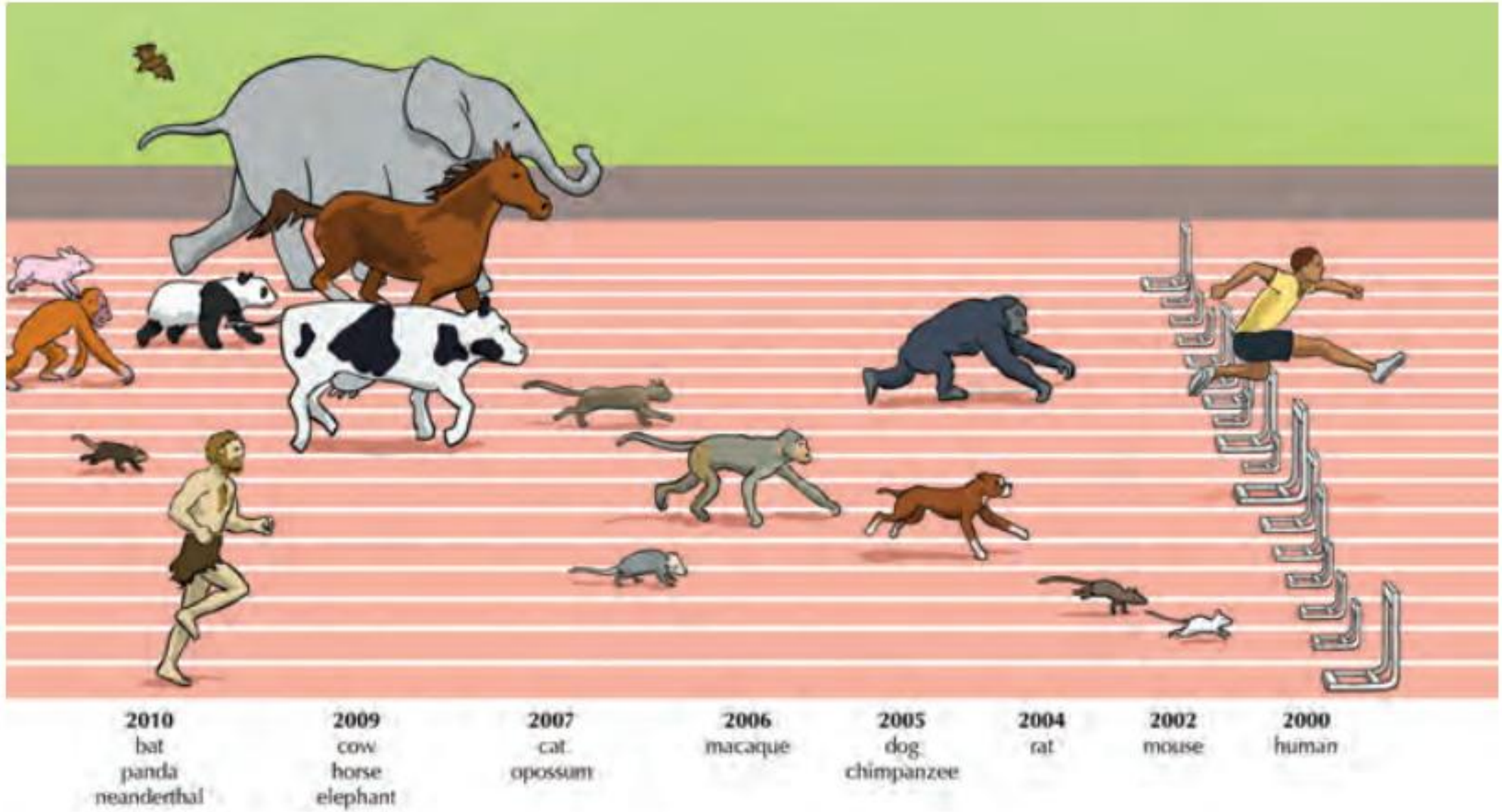



FIGURE 3.3 The first mammals with sequenced genomes.

Library schematic	Output
Illumina P5 P7 	Sequenze corte (50-250 bp) e relativamente accurate. Economico quindi alta coverage. Ad oggi ancora il 75% circa dei dati presenti.

Modificato da Rise e Green 2019, Ann.Rev.Anim.Bio.

Modificato da Rise e Green 2019, Ann.Rev.Anim.Bio.

Library schematic	Output
Illumina 	Sequenze corte (50-250 bp) e relativamente accurate. Economico quindi alta coverage. Ad oggi ancora il 75% circa dei dati presenti.
PacBio 	Sequenze HiFi accurate (errore < 0.5%) e lunghe >10kb.
Oxford Nanopore 	Sequenze ultra-lunghe di anche >100kb ma con errore <10%

Modificato da Rise e Green 2019, Ann.Rev.Anim.Bio.

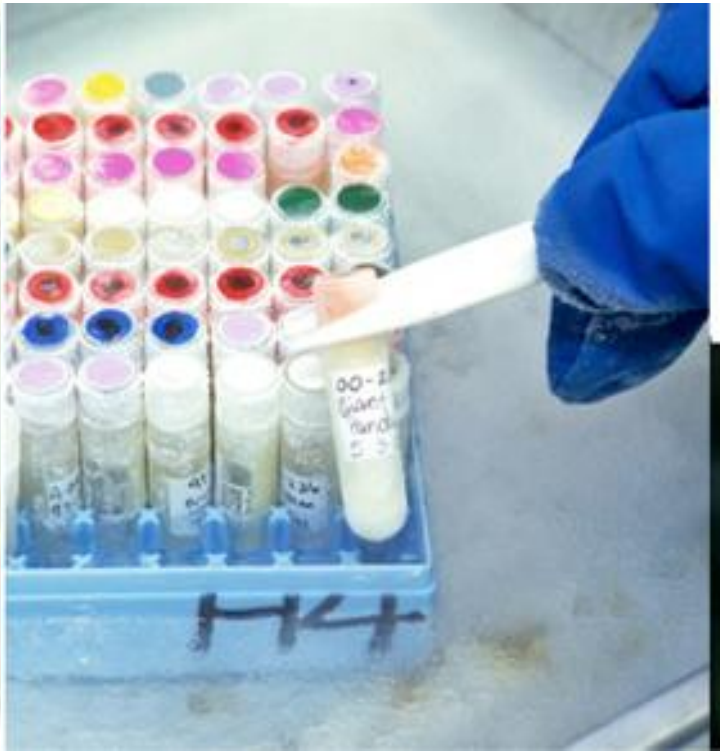
Library schematic	Output
Illumina 	Sequenze corte (50-250 bp) e relativamente accurate. Economico quindi alta coverage. Ad oggi ancora il 75% circa dei dati presenti.
PacBio 	Sequenze HiFi accurate (errore < 0.5%) e lunghe >10kb.
Oxford Nanopore 	Sequenze ultra-lunghe di anche >100kb ma con errore <10%
Hi-C ed altre tecnologie	Informazioni a lungo raggio, fino a 10Mb


T2T-CHM13
The complete sequence of a human genome


The perplexing figure behind a crucial virus database p. 332
Regulatory reforms to advance psychedelic therapies p. 347
A compact galaxy in the early Universe p. 416
Science
$15
28 APRIL 2023
SPECIAL ISSUE
science.org
AAAS
ZOONOMIA
Diverse genomes reveal mammalian secrets p. 356


Zoonomia project


EARTH
BIOGENOME
PROJECT
sequencing life for the future of life




GENOME 10K

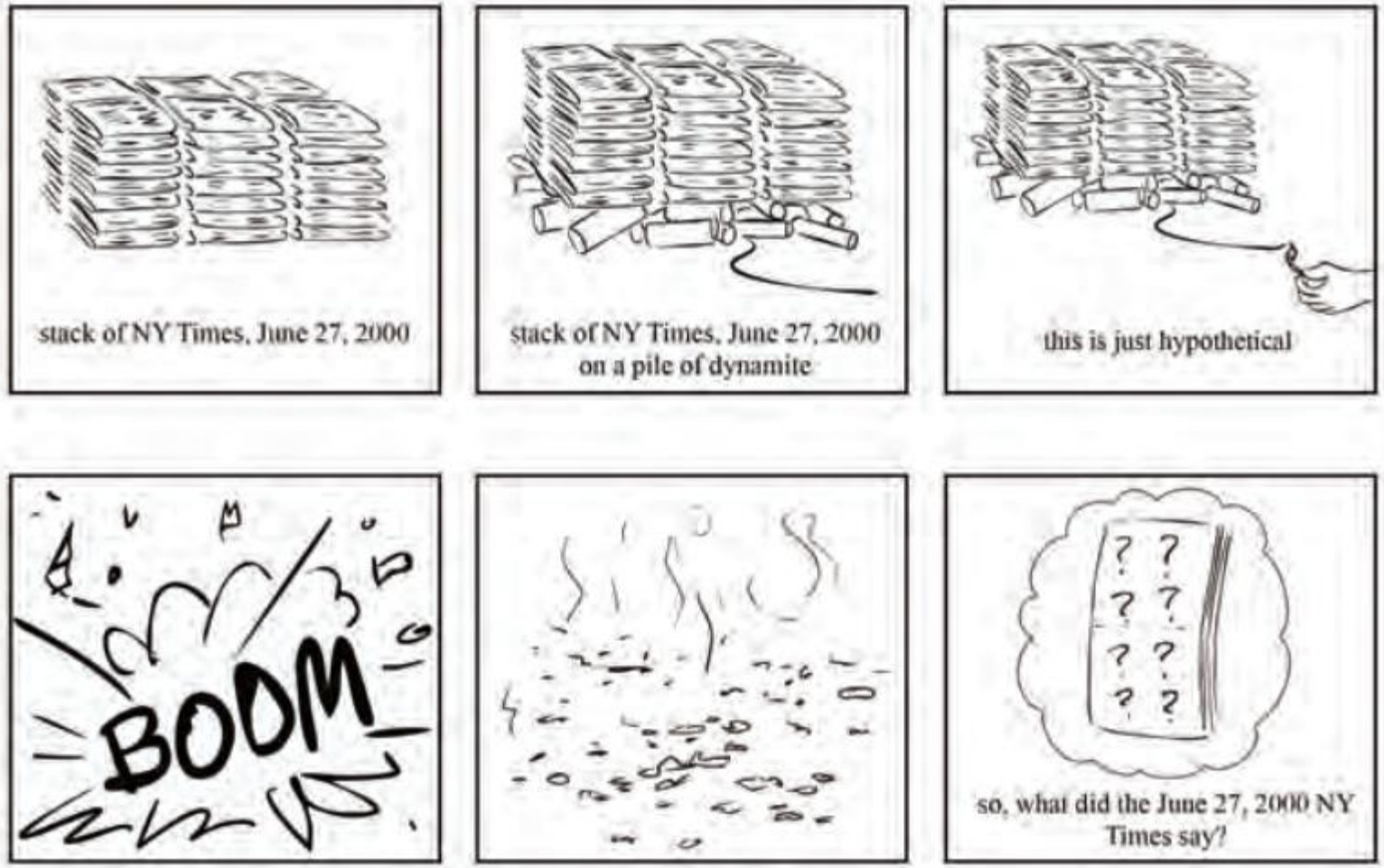


FIGURE 3.1 Don't try this at home! Crazy as it may seem, the Newspaper Problem serves as an analogy for the computational framework of genome assembly.

Genomic DNA

Reads

Draft Genome Sequence

Closed Genome Sequence

Figura da https://training.galaxyproject.org/training-material/topics/assembly/tutorials/general-introduction/slides-plain.html

Caratteristiche del Newspaper Problem

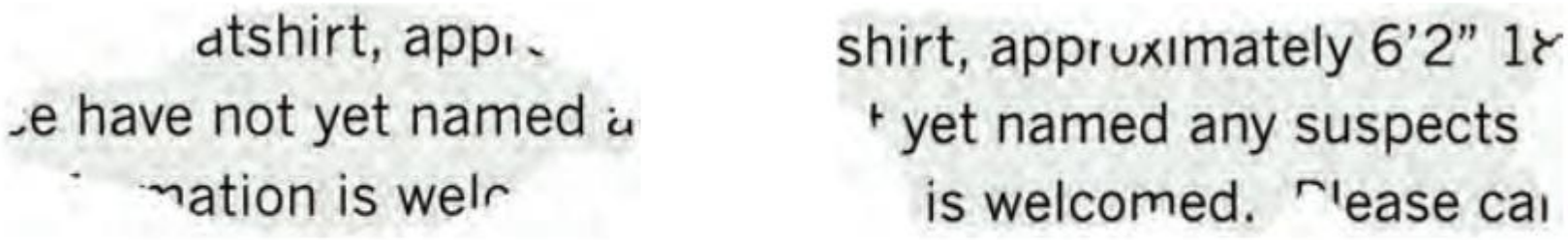


FIGURE 3.2 In the Newspaper Problem, we need to use overlapping shreds of paper to figure out the news.

- Copie multiple del giornale, quindi alcune frasi possono essere ripetute
- Nell'esplosione alcune parti possono essere andate distrutte
- Alcune lettere possono essere lette erroneamente

Shakespearomics

- ***Reads***

```
ds, Romans, count
ns, countrymen, le
Friends, Rom
send me your ears;
crymen, lend me
```

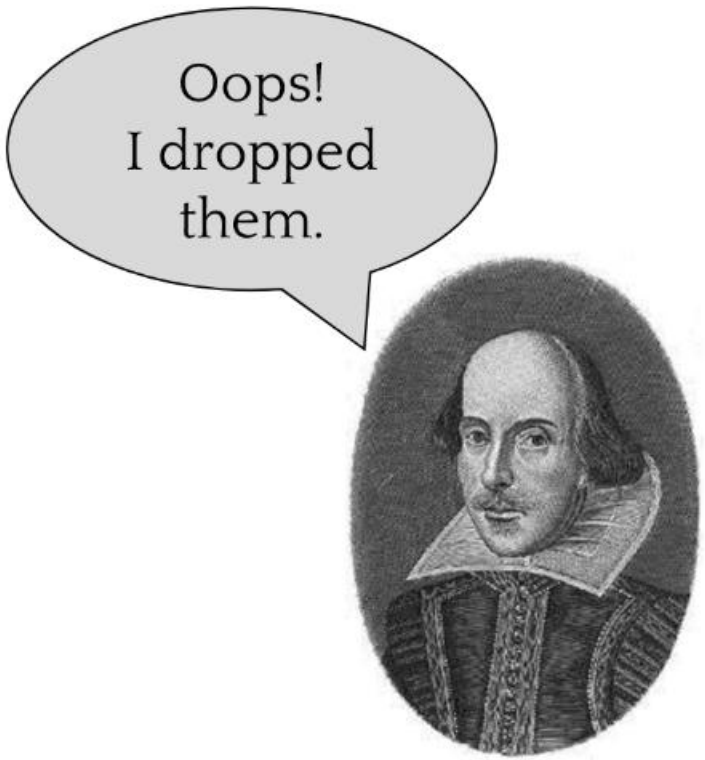


Figura da https://training.galaxyproject.org/training-material/topics/assembly/tutorials/general-introduction/slides-plain.html

Shakespearomics

***Strategia 1**: cerchiamo le parti di parola sovrapposte (overlaps)*

- ***Reads***

```
ds, Romans, count
ns, countrymen, le
Friends, Rom
send me your ears;
crymen, lend me
```

- ***Overlaps***

```
Friends, Rom
      ds, Romans, count
            ns, countrymen, le
                  crymen, lend me
                        send me your ears;
```

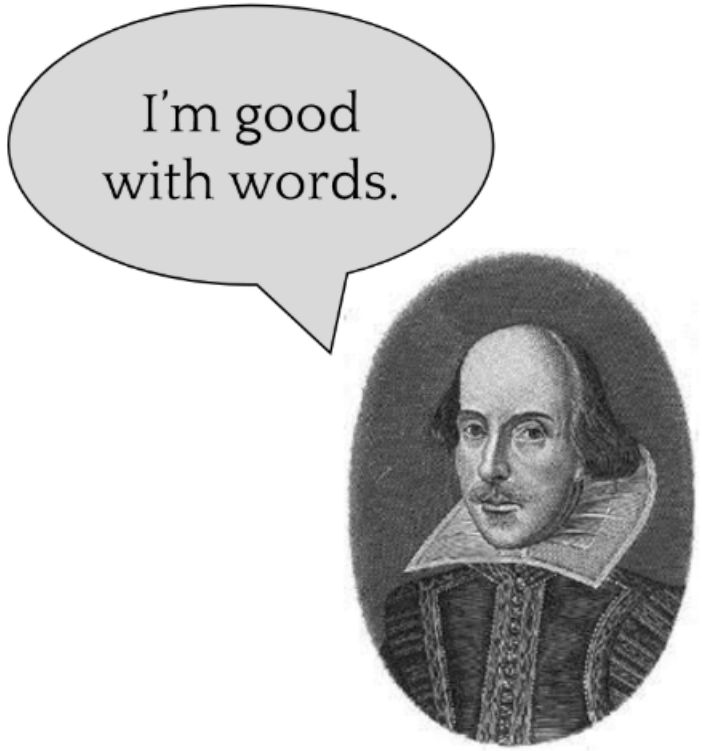


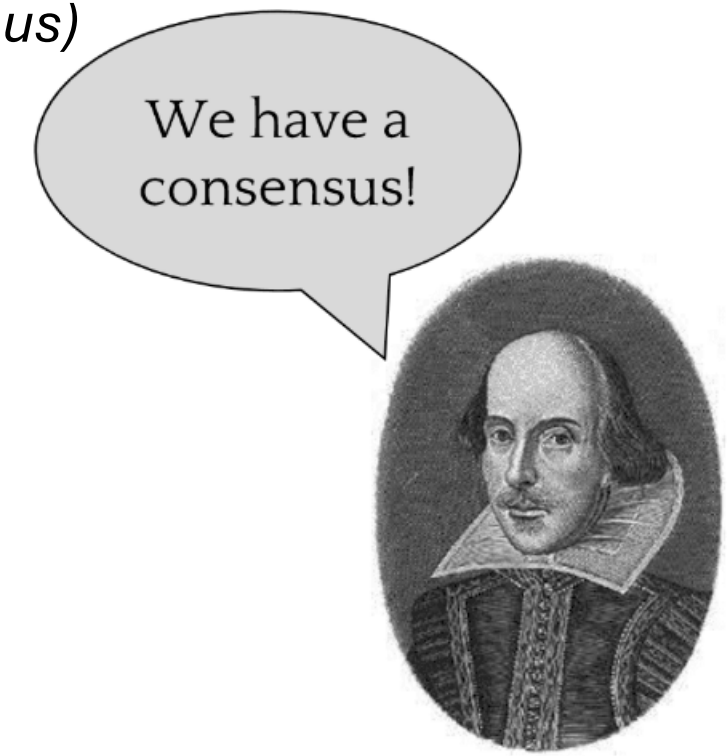
Figura da https://training.galaxyproject.org/training-material/topics/assembly/tutorials/general-introduction/slides-plain.html

Shakespearomics

***Strategia 1**: cerchiamo le parti di parola sovrapposte (overlaps) e correggiamo gli errori prendendo la lettera piu' comune (consensus)*

- ***Reads***

```
ds, Romans, count
ns, countrymen, le
Friends, Rom
send me your ears;
crymen, lend me
```



- ***Overlaps***

```
Friends, Rom
      ds, Romans, count
            ns, countrymen, le
                  crymen, lend me
                        send me your ears;
```

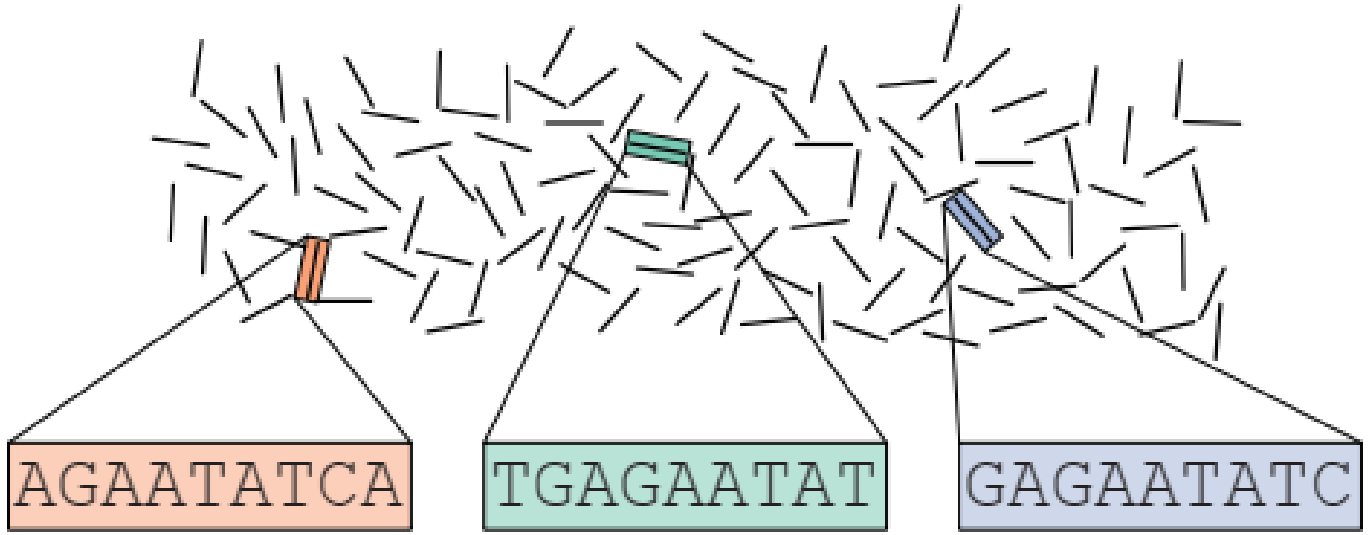
- ***Majority consensus***

```
Friends, Romans, countrymen, lend me your ears;
```

Multiple identical copies of a genome

Shatter the genome into reads

Sequence the reads



Assemble the genome using overlapping reads

AGAATATCA
GAGAATATC
TGAGAATAT
...TGAGAATATCA...

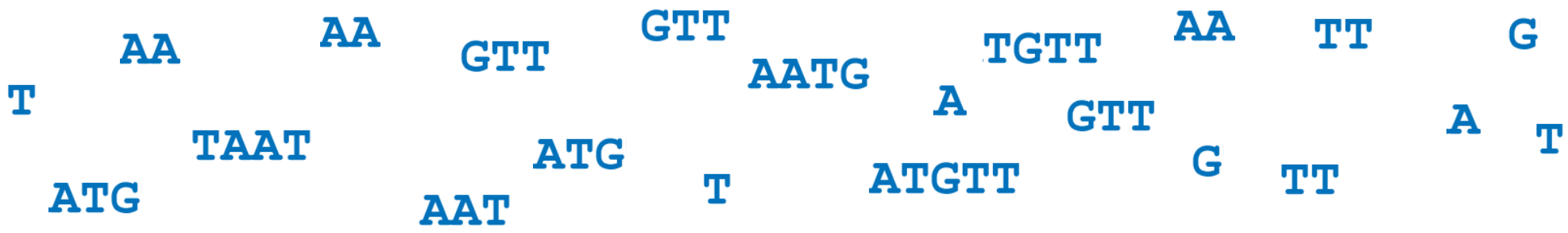
Evviva, abbiamo risolto il problema! Basta trovare gli overlap e correggere gli errori con un consenso. Lezione finita!

Evviva, abbiamo risolto il problema! Basta trovare gli overlap e correggere gli errori con un consenso. Lezione finita!

Ma non è così semplice..

La ricostruzione di una stringa

- Assemblare equivale a ricostruire una stringa (ad esempio **TAATGTT**) a partire da dei suoi frammenti:



La ricostruzione di una stringa

- Assemblare equivale a ricostruire una stringa (ad esempio **TAATGTT**) a partire da dei suoi frammenti:

- Per iniziare, semplifichiamo il problema assumendo che:
  - I suoi frammenti abbiano tutti la stessa lunghezza
  - Ogni frammento sia ripetuto una sola volta

La ricostruzione di una stringa

- Assemblare equivale a ricostruire una stringa (ad esempio **TAATGTT**) a partire da dei suoi frammenti:

- Per iniziare, semplifichiamo il problema assumendo che:
  - I suoi frammenti abbiano tutti la stessa lunghezza
  - Ogni frammento sia ripetuto una sola volta

AAT ATG GTT TAA TGT

Un esempio semplice TAATGTT

- Possiamo suddividerla nei 3-meri

```
AAT ATG GTT TAA TGT
```

- Da dove iniziare?

Un esempio semplice TAATGTT

- Possiamo suddividerla nei 3-meri

AAT ATG GTT TAA TGT

Un esempio semplice TAATGTT

- Possiamo suddividerla nei 3-meri

AAT ATG GTT TAA TGT

- Da dove iniziare? Nessun 3-mero finisce con TA, quindi TAA deve essere all'inizio.

TAA

Un esempio semplice **TAATGTT**

- Possiamo suddividerla nei 3-meri

AAT ATG GTT TAA TGT

- Da dove iniziare? Nessun 3-mero finisce con TA, quindi TAA deve essere all'inizio.

TAA

TAA

AA**T**

Un esempio semplice **TAATGTT**

- Possiamo suddividerla nei 3-meri

AAT ATG GTT TAA TGT

- Da dove iniziare? Nessun 3-mero finisce con TA, quindi TAA deve essere all'inizio.

TAA

TAA
AA**T**

TAA
AA**T**
AT**G**

Un esempio semplice **TAATGTT**

- Possiamo suddividerla nei 3-meri

AAT ATG GTT TAA TGT

- Da dove iniziare? Nessun 3-mero finisce con TA, quindi TAA deve essere all'inizio.

TAA

TAA
AA**T**

TAA
AA**T**
AT**G**

TAA
AA**T**
AT**G**
TG**T**

Un esempio semplice **TAATGTT**

- Possiamo suddividerla nei 3-meri

AAT ATG GTT TAA TGT

- Da dove iniziare? Nessun 3-mero finisce con TA, quindi TAA deve essere all'inizio.

TAA

TAA
AA**T**

TAA
AA**T**
AT**G**

TAA
AA**T**
AT**G**
TG**T**

TAA
AA**T**
AT**G**
TG**T**
GT**T**
TAATGTT

Il nostro primo assemblaggio di DNA!

Un esempio leggermente piu’ realistico

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

- Cominciamo ad assemblare i notri 3meri da TAA come prima, visto che nessuno termina con TA.

TAA

Un esempio leggermente piu’ realistico

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

- Cominciamo ad assemblare i notri 3meri da TAA come prima, visto che nessuno termina con TA.
- Il prossimo 3mero dovrebbe cominciare con AA, quindi abbiamo solo AAT. E poi, AAT puo’ essere esteso solo da ATG.

```
TAA
 AAT
  ATG
TAATG
```

Un esempio leggermente piu’ realistico

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

- Cominciamo ad assemblare i notri 3meri da TAA come prima, visto che nessuno termina con TA.
- Il prossimo 3mero dovrebbe cominciare con AA, quindi abbiamo solo AAT. E poi, AAT puo’ essere esteso solo da ATG.
- Ora possiamo scegliere vari che iniziano con TG. Avete un preferito?

```
TAA
 AAT
  ATG
TAATG
```

Un esempio leggermente piu’ realistico

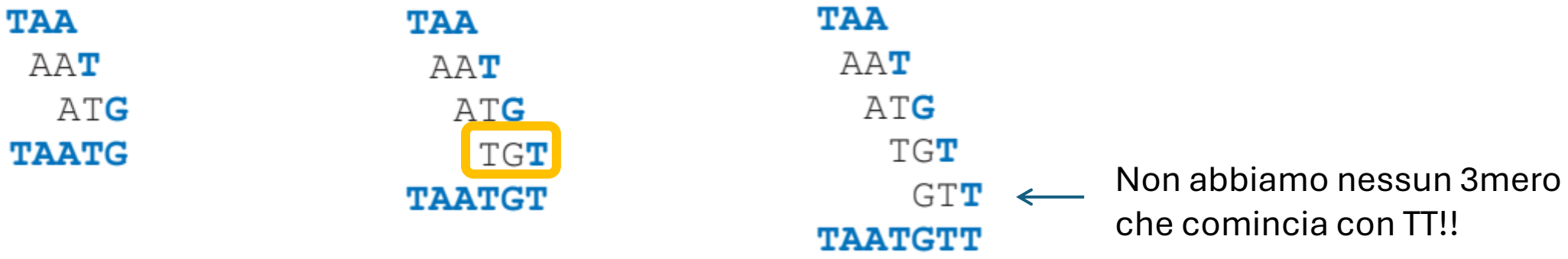
AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

- Cominciamo ad assemblare i notri 3meri da TAA come prima, visto che nessuno termina con TA.
- Il prossimo 3mero dovrebbe cominciare con AA, quindi abbiamo solo AAT. E poi, AAT puo’ essere esteso solo da ATG.
- Ora possiamo scegliere vari che iniziano con TG. Avete un preferito? Scegliamo TGT.

Un esempio leggermente piu' realistico

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

- Cominciamo ad assemblare i notri 3meri da TAA come prima, visto che nessuno termina con TA.
- Il prossimo 3mero dovrebbe cominciare con AA, quindi abbiamo solo AAT. E poi, AAT puo' essere esteso solo da ATG.
- Ora possiamo scegliere vari che iniziano con TG. Avete un preferito? Scegliamo TGT. A poi continuiamo..oopps, bloccati!!!



Un esempio leggermente piu’ realistico

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

- Torniamo indietro

Un esempio leggermente piu’ realistico

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

- Torniamo indietro
- Proseguendo, possiamo completare l’assemblaggio della sequenza

TAA
AAT
ATG
TAATG

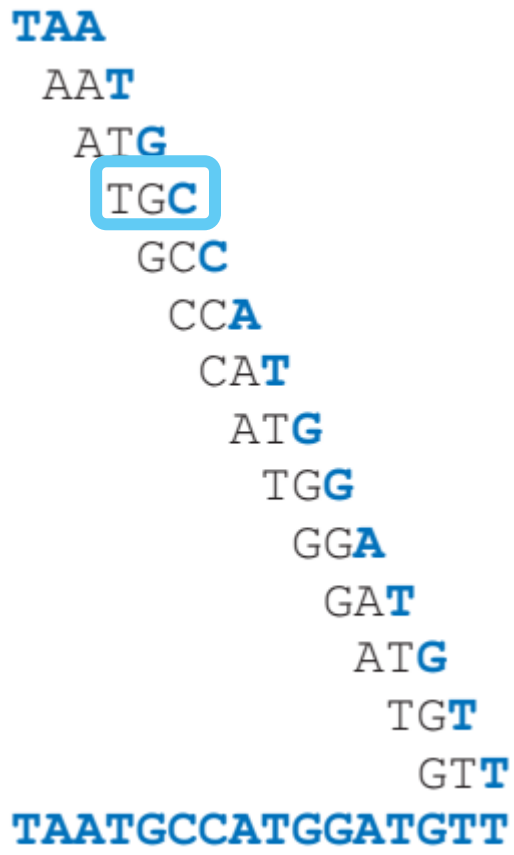
TAA
AAT
ATG
TGC
TAATGC

TAA
AAT
ATG
TGC
GCC
CCA
CAT
ATG
TGG
GGA
GAT
ATG
TGT
GTT
TAATGCCATGGATGTT

Un esempio leggermente piu’ realistico

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

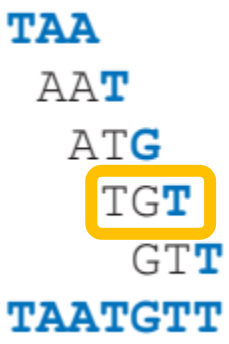
- Perché questa sequenza ci ha causato questo problema? Perché abbiamo dovuto riniziare?

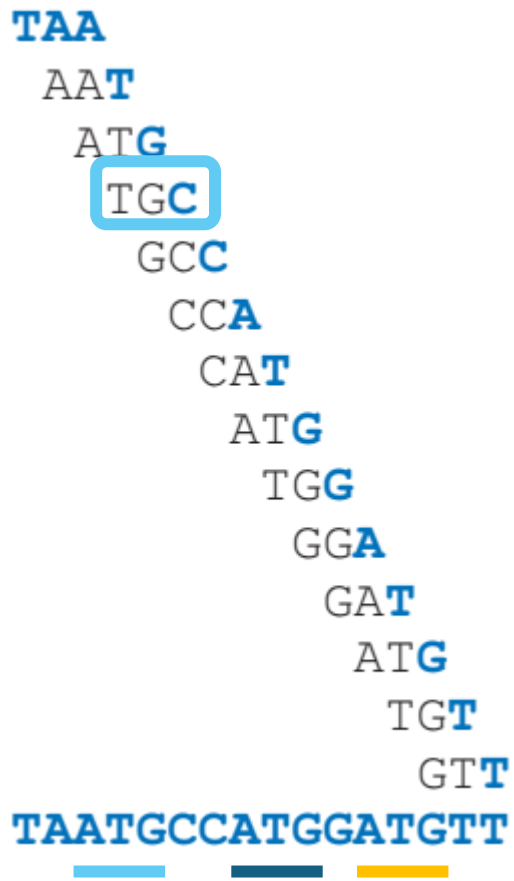


Un esempio leggermente piu’ realistico

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

- Perché questa sequenza ci ha causato questo problema? Perché abbiamo dovuto riniziare?
- Il trimero ATG (e quindi ATG) era ripetuto tre volte, portandoci a percorsi alternativi.





Un esempio leggermente piu’ realistico

```
AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT
```

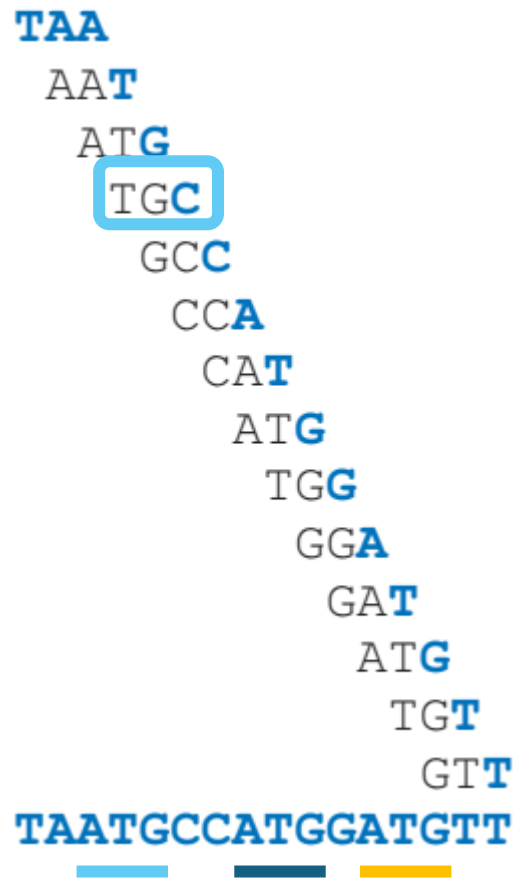
- Perché questa sequenza ci ha causato questo problema? Perché abbiamo dovuto riniziare?
- Il trimero ATG (e quindi ATG) era ripetuto tre volte, portandoci a percorsi alternativi.
- Per casi semplici è risolvibile (as esempio nel nostro esempio o nel gioco Triazzle).

Immaginate però per milioni e milioni

di ripetizioni..

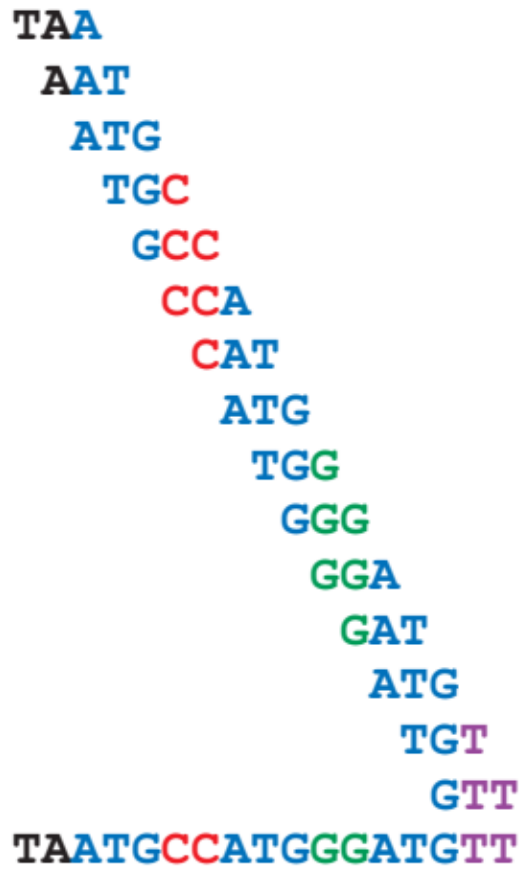
Come risolvere efficientemente il nostro esempio leggermente piu’ realistico?

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT



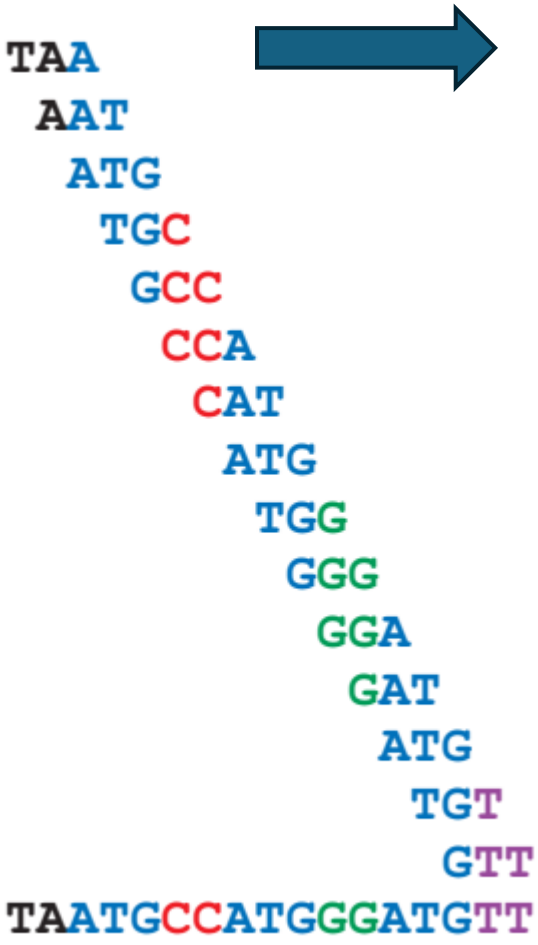
L’assemblaggio come un percorso nell’ “overlap graph”

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

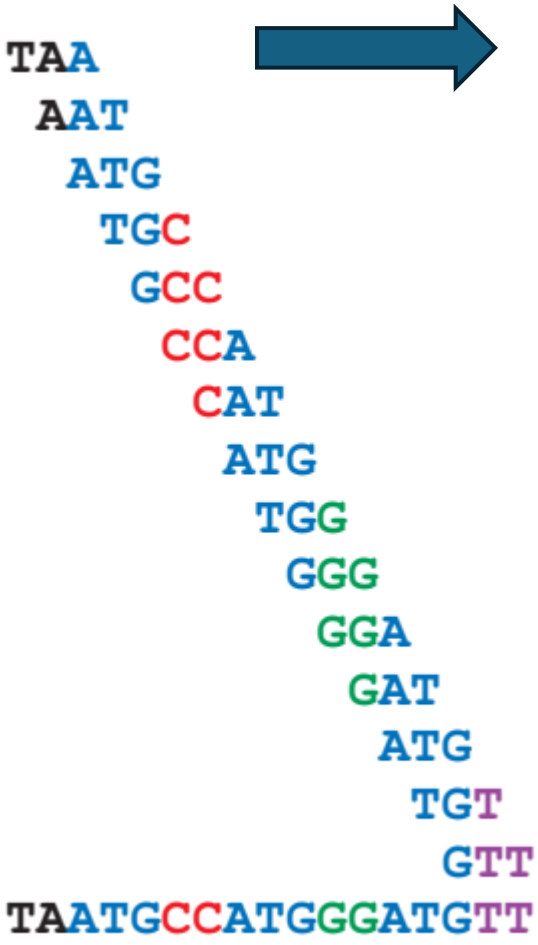


Qualche definizione/sinonimo:
path: *percorso*
overlap graph: *grafo delle sovrapposizioni*

Possiamo rappresentare la nostra stringa come un “overlap graph” che unisce i 3meri “sovrapposti”



Possiamo rappresentare la nostra stringa come un “overlap graph” che unisce i 3meri “sovrapposti”

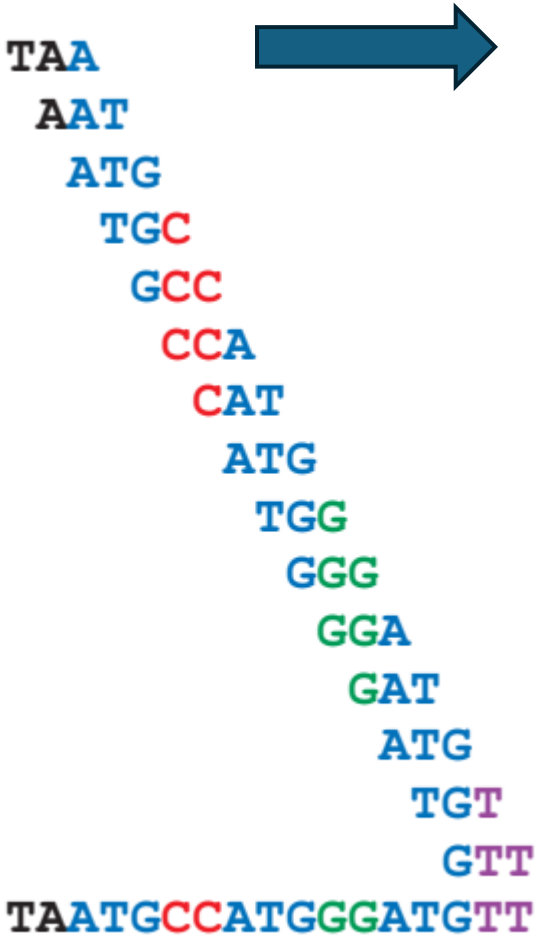


La nostra stringa unisce trimeri sovrapposti di due caratteri.
Con trimeri sovrapposti di due nucleotidi definiamo trimeri il cui suffisso del primo (gli ultimi due caratteri) siano uguali al prefisso del successivo.

$\text{SUFFIX}(\mathbf{TAA}) = \mathbf{AA}$

$\text{PREFIX}(\mathbf{AAT}) = \mathbf{AA}$

Possiamo rappresentare la nostra stringa come un “overlap graph” che unisce i 3meri “sovrapposti”



La nostra stringa unisce trimeri sovrapposti di due caratteri.
Con trimeri sovrapposti di due nucleotidi definiamo trimeri il cui suffisso del primo (gli ultimi due caratteri) siano uguali al prefisso del successivo.

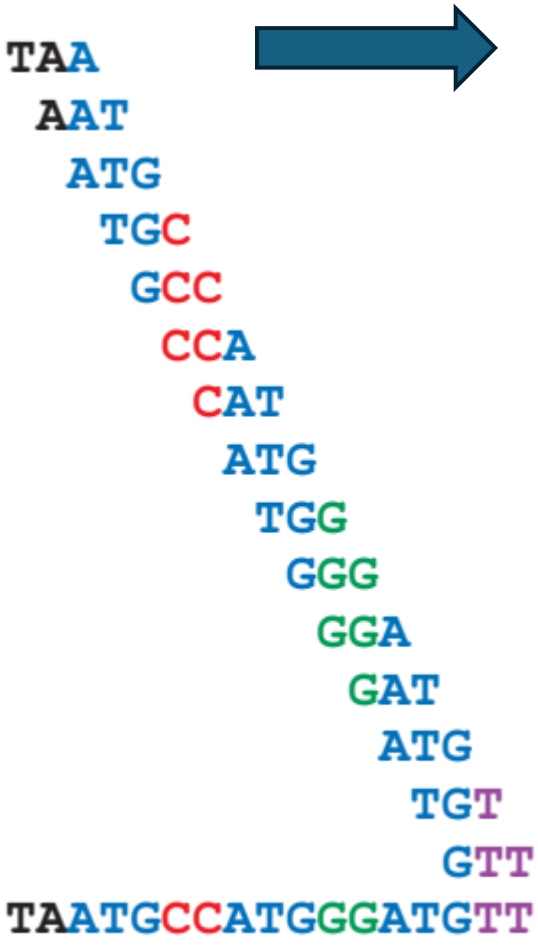
$$\text{SUFFIX}(\mathbf{TAA}) = \mathbf{AA}$$
$$\text{PREFIX}(\mathbf{AAT}) = \mathbf{AA}$$

Nota: Nel nostro grafo:

- i **nodi** sono i trimeri
- gli **archi** (le frecce) sono gli overlap/sovrapposizioni

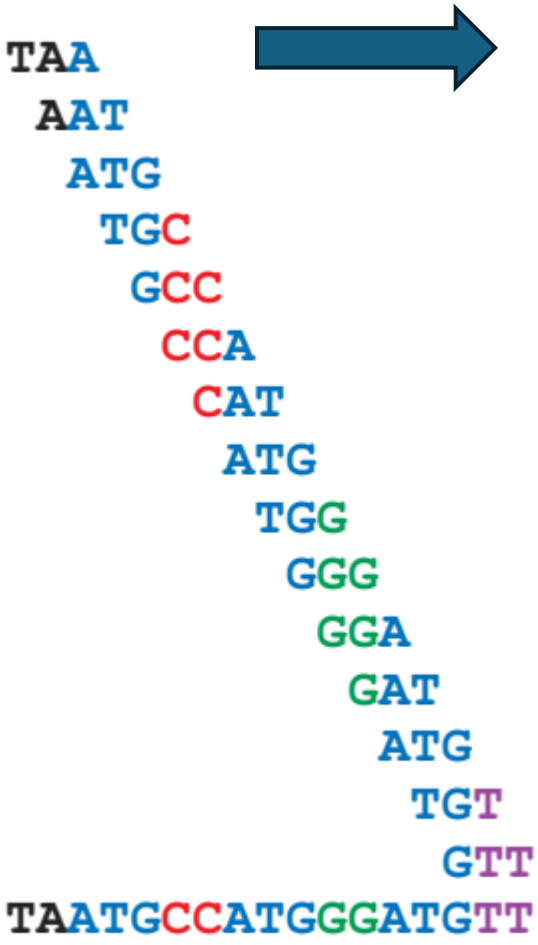
A volte in «teoria dei grafi» si parla anche di «vertici» (nodi) e «spigoli» (archi).
Il nostro grafo è inoltre «diretto» (cioè ha frecce unidirezionali), mentre un grafo indiretto è bidirezionale.

Possiamo rappresentare la nostra stringa come un “overlap graph” che unisce i 3meri “sovrapposti”

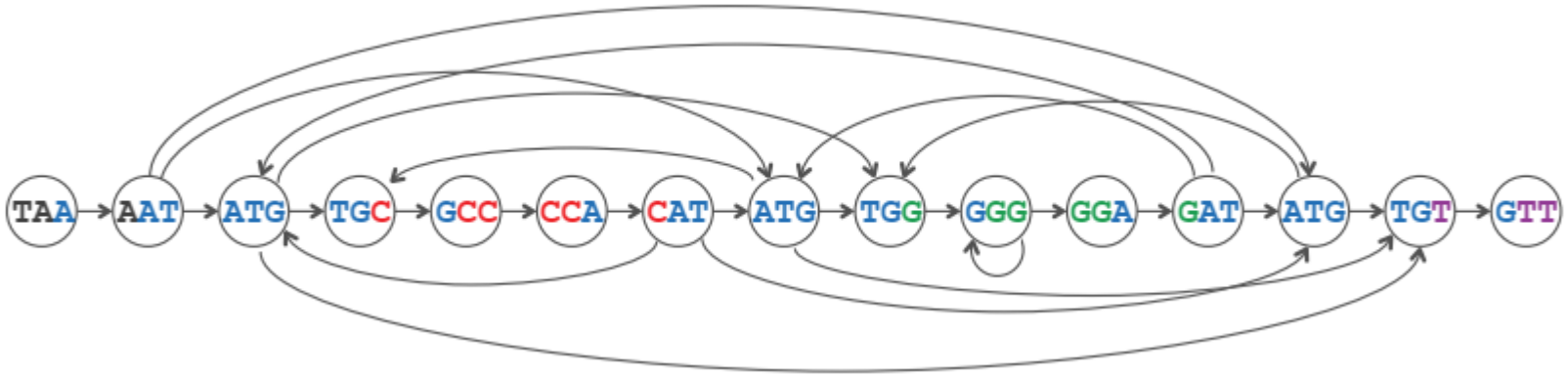


Ma questi non erano tutti i possibili percorsi. E noi non conosciamo il genoma in anticipo!

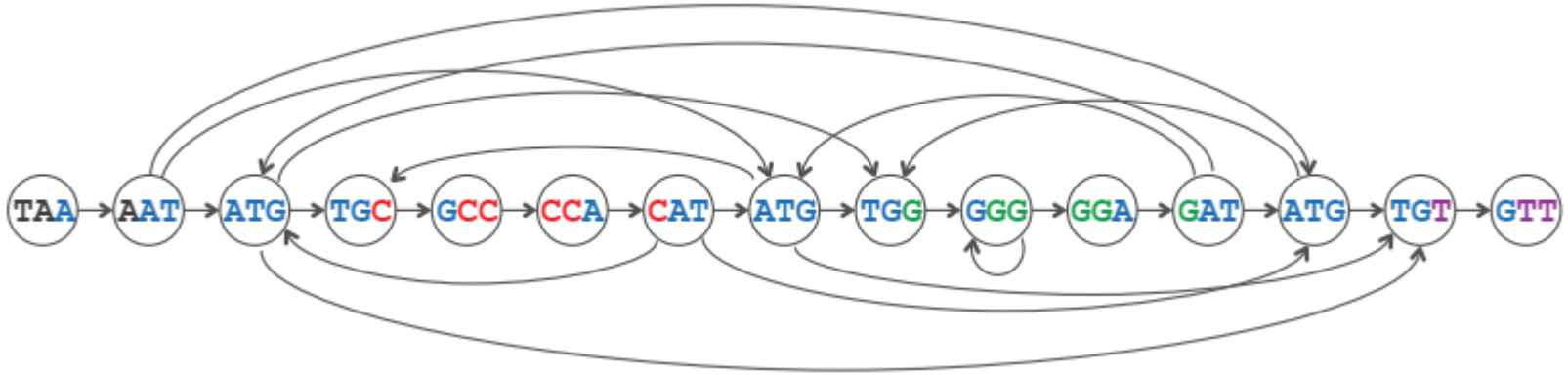
Possiamo rappresentare la nostra stringa come un “overlap graph” che unisce i 3meri “sovrapposti”



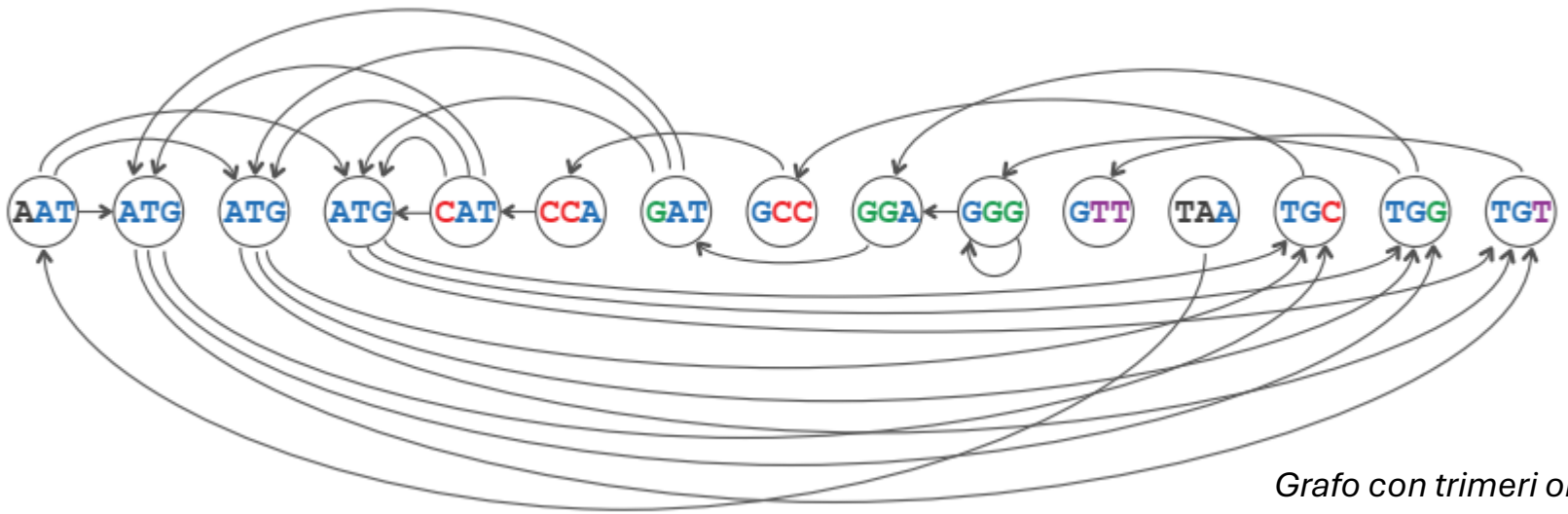
Quindi dobbiamo unire tutti i possibili casi in cui il prefisso di un trimero è uguale al suffisso di un altro trimero. Cosa otteniamo?



Voi potreste dirmi «OK, basta seguire le freccine dritte e dimenticarsi delle altre per trovare la soluzione»..

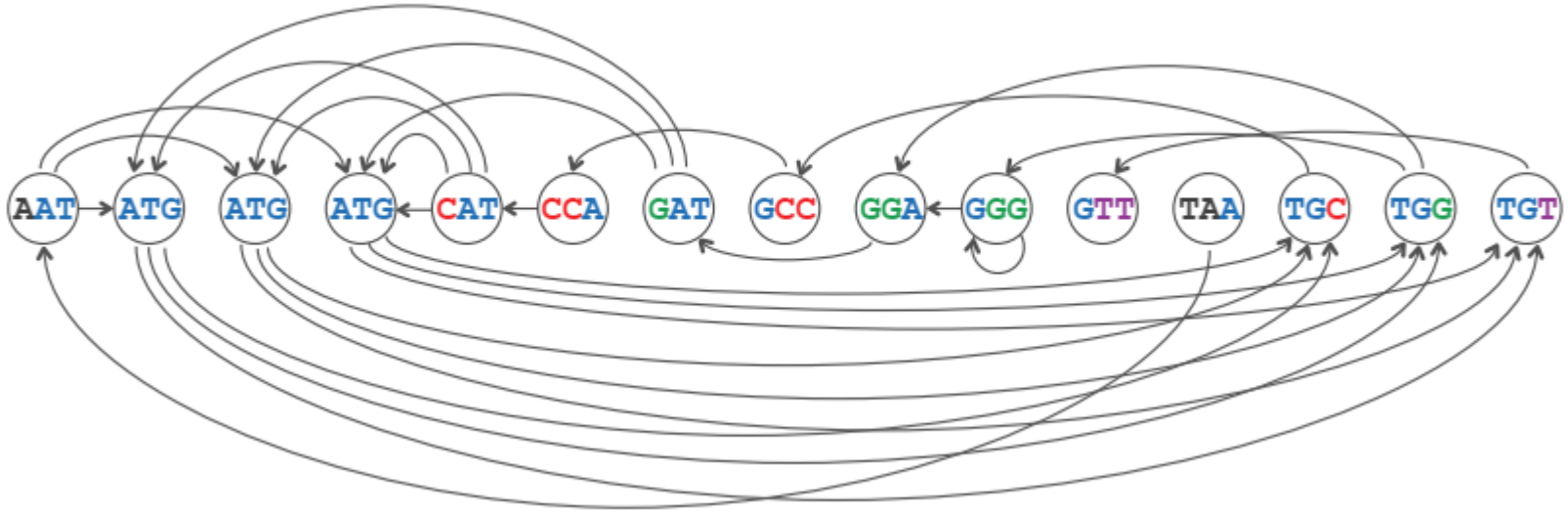


..tuttavia non sappiamo a priori l'ordine dei trimeri. Quindi la nostra situazione è piu' simile a:

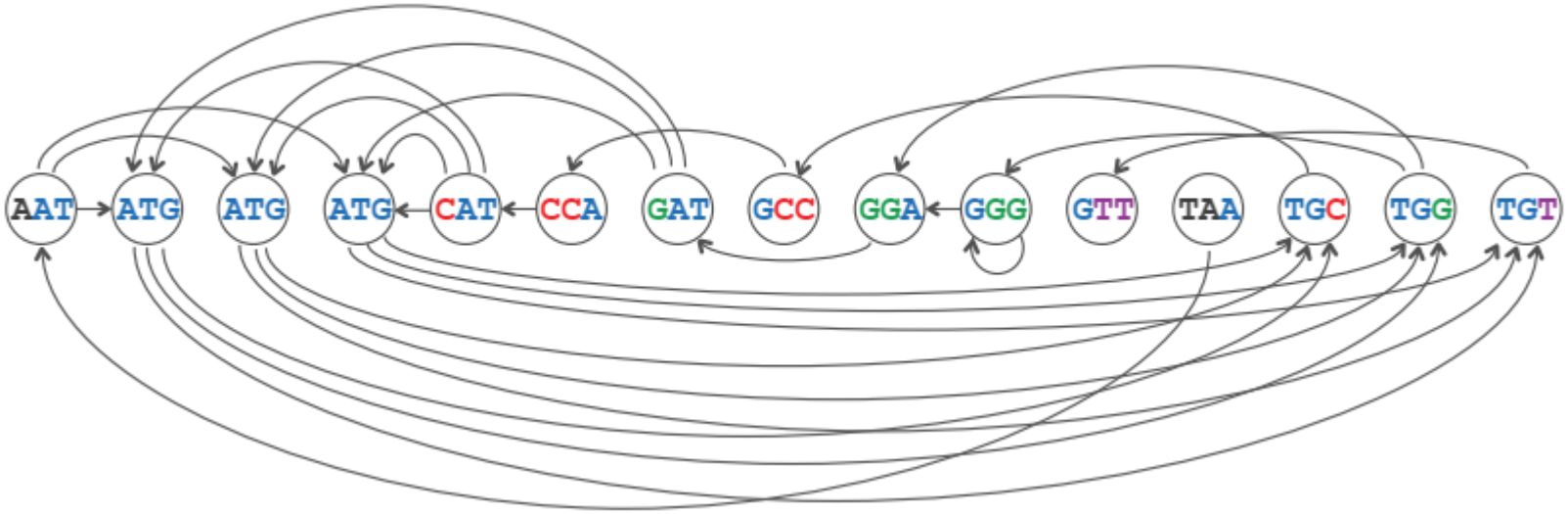


Grafo con trimeri ordinati alfabeticamente

Una definizione formale del nostro problema: **Trovare il percorso «giusto*» nell' «overlap graph».**

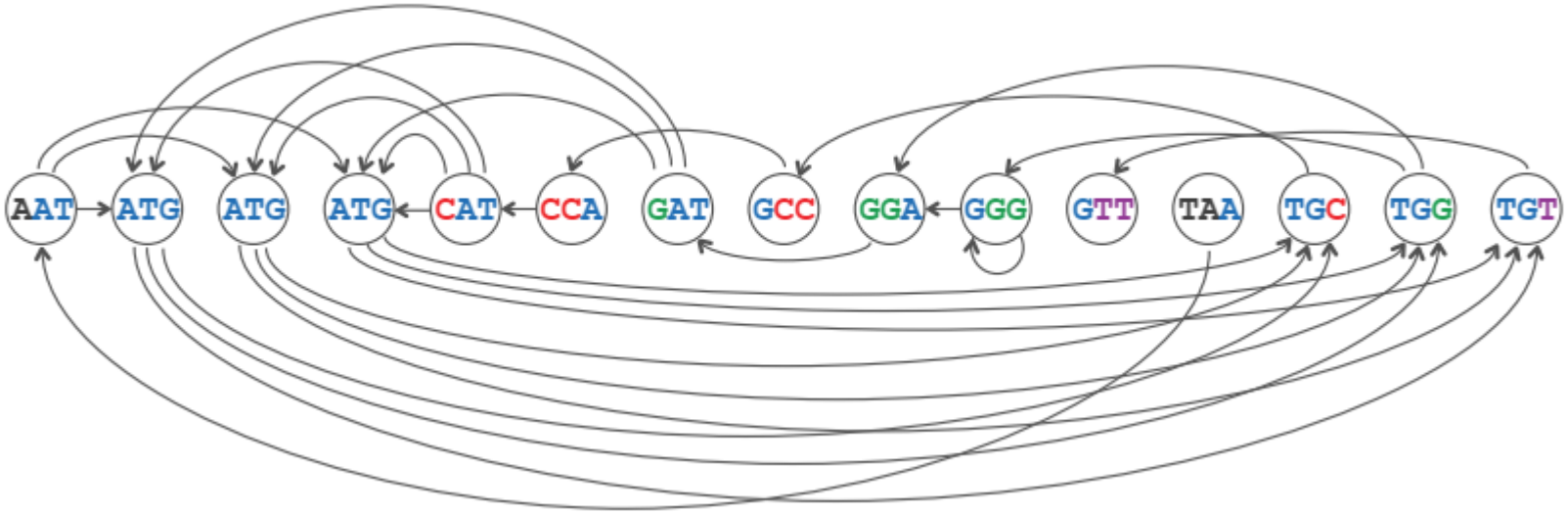


Una definizione formale del nostro problema: **Trovare il percorso «giusto*» nell' «overlap graph».**



*** se siete interessati ai termini:**
«giusto»: anche chiamato percorso Hamiltoniano, un percorso che passa per tutti i nodi una volta sola.

Una definizione formale del nostro problema: **Trovare il percorso «giusto*» nell' «overlap graph».**



E abbiamo già il nostro algoritmo per trovare le soluzioni:

1) Troviamo un trimero iniziale (che ha un prefisso che non è suffisso di nessun altro trimero).
2) Connettiamo il nostro trimero ad uno sovrapposto (la cui fine o «suffisso» sia uguale al prefisso del trimero successivo)
3) Ripetiamo per tentativi finché troviamo un percorso «giusto» (Hamiltoniano)

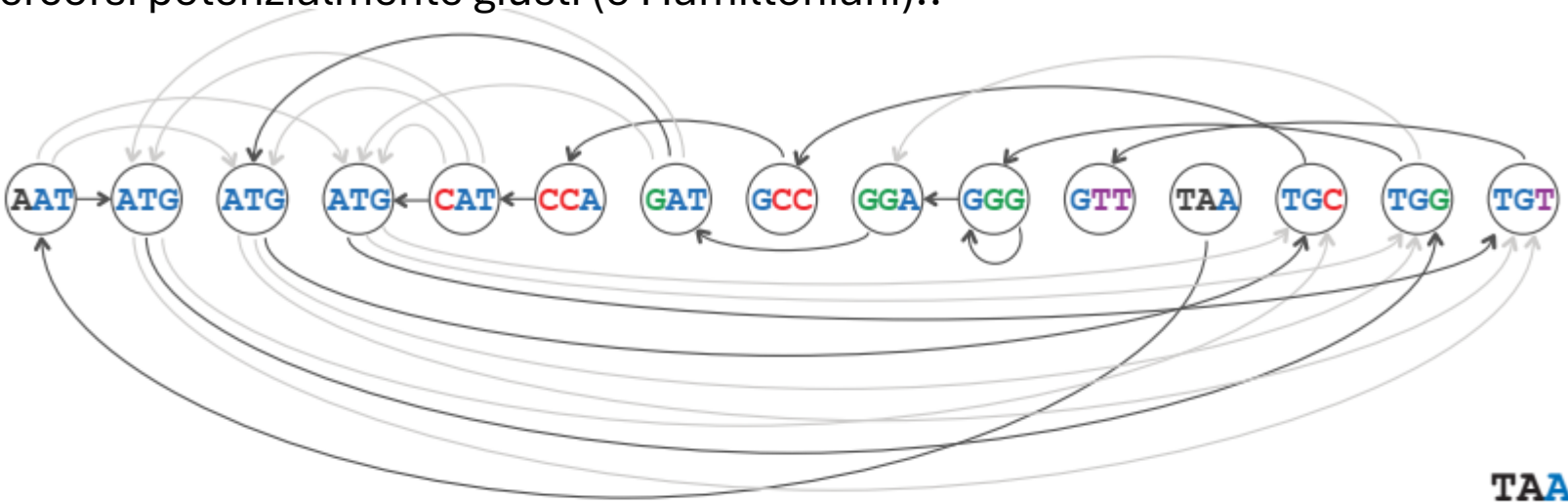
Nell'esempio:

TAA, perché nessun altro trimero finisce con TA

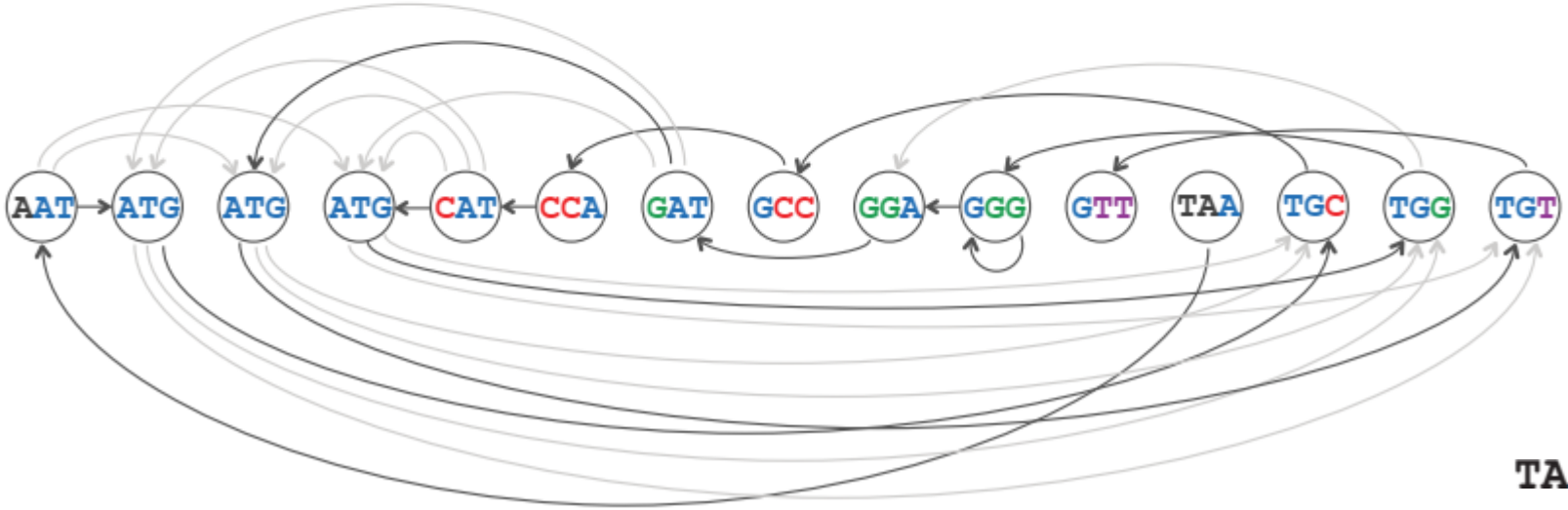
Suffisso(TAA)=AA=Prefisso(AAT) quindi AAT segue TAA

*** se siete interessati ai termini:**
«giusto»: anche chiamato percorso Hamiltoniano, un percorso che passa per tutti i nodi una volta sola.

Ecco le soluzioni (seguendo il nostro algoritmo). E nel nostro caso c'erano persino due percorsi potenzialmente giusti (o Hamiltoniani)!!



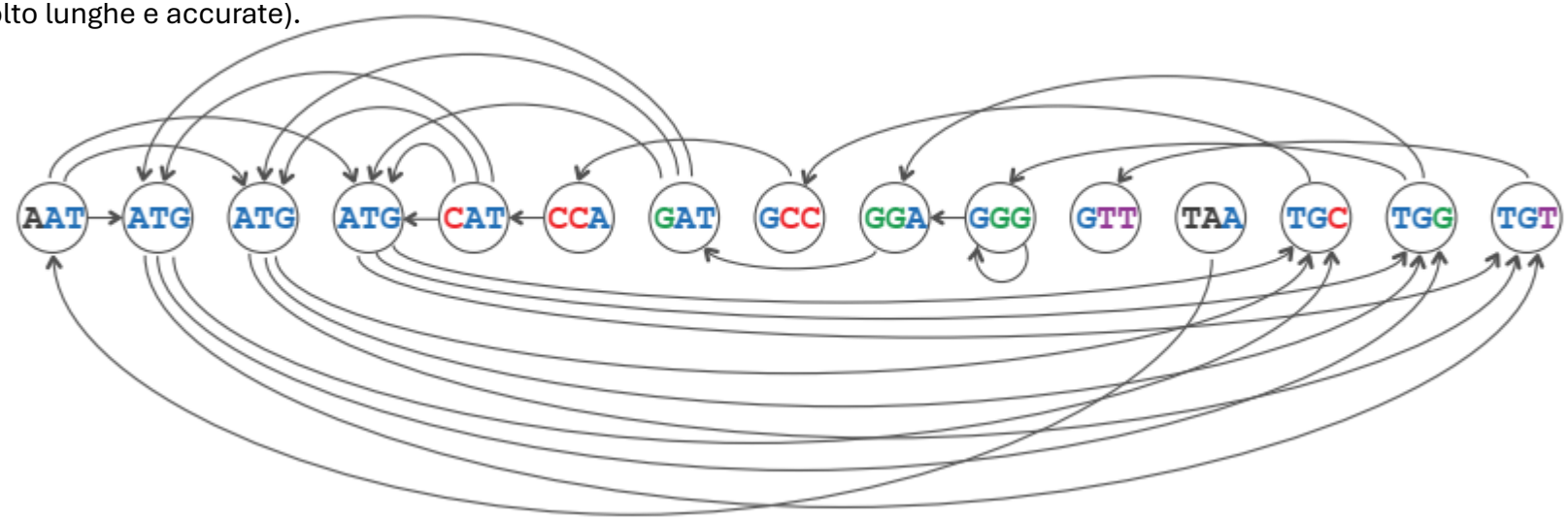
TAATGCCATGGGATGTT



TAATGGGATGCCATGTT

Gli «overlap graphs» sono alla base di vari assemblatori moderni

(specializzati in sequenze molto lunghe e accurate).



ARTICLES
https://doi.org/10.1038/s41592-020-01056-5
nature methods
Check for updates

Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm

Haoyu Cheng[1,2], Gregory T. Concepcion[3], Xiaowen Feng[1,2], Haowen Zhang[4] and Heng Li[1,2]

Method

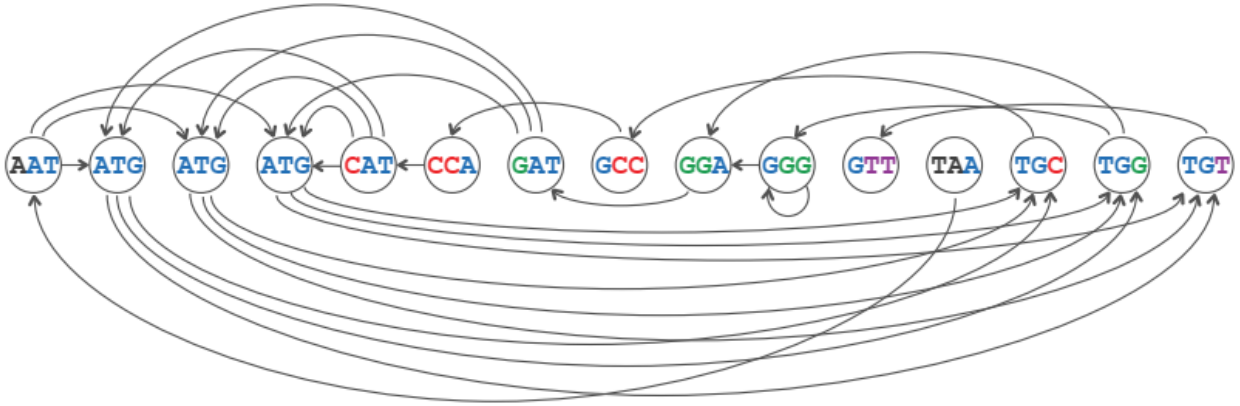
HiCanu: accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads

Sergey Nurk,[1,6] Brian P. Walenz,[1,6] Arang Rhie,[1] Mitchell R. Vollger,[2] Glennis A. Logsdon,[2] Robert Grothe,[3] Karen H. Miga,[4] Evan E. Eichler,[2,5] Adam M. Phillippy,[1] and Sergey Koren[1]

Svantaggi degli overlap-graphs

Tuttavia abbiamo anche visto dei problemi negli overlap graph:

- molteplici percorsi Hamiltoniani/soluzioni potenzialmente giuste
- **grafi complessi**, che diventano rapidamente* (con sequenze corte e genomi lunghi/ripetuti) **intrattabili**



Svantaggi degli overlap-graphs

Tuttavia abbiamo anche visto dei problemi negli overlap graph:

- molteplici percorsi Hamiltoniani/soluzioni potenzialmente giuste
- **grafi complessi**, che diventano rapidamente* (con sequenze corte e genomi lunghi/ripetuti) **intrattabili**

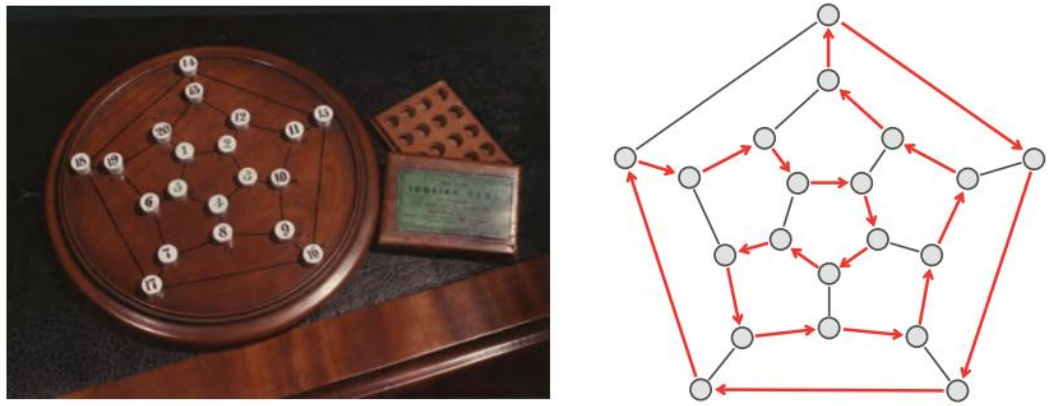
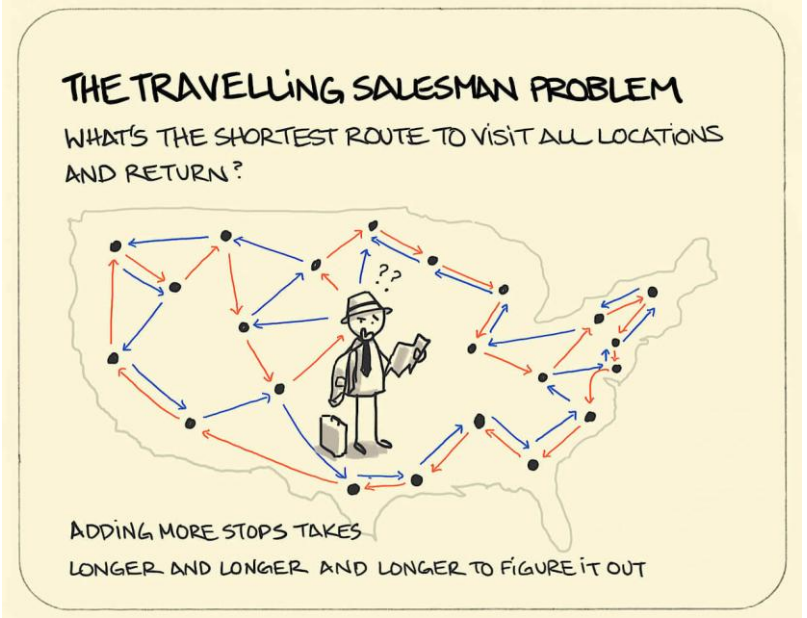


FIGURE 3.45 (Left) Hamilton's icosian game, with a winning placement of the pegs shown. (Right) The winning placement of the pegs can be represented as a Hamiltonian cycle in a graph. Each node in this graph represents a peghole on the board, and an edge connects two pegholes that are connected by a line segment on the board.



** Trovare un percorso Hamiltoniano ad un overlap-graph è un problema NP-complete o intrattabile, cioè al crescere dei dati il tempo per risolverlo cresce esponenzialmente e non è mai stato trovato un algoritmo per risolverlo in tempo polinomiale! Se lo trovate vincerete 1 milione di dollari dal Clay Mathematics Institute!*

Proviamo a riformulare il problema

Ripartiamo dalla nostra sequenza

TAATGCCATGGGATGTT

estrando i trimeri

TAA AAT ATG TGC GCC CCA CAT ATG TGG GGG GGA GAT ATG TGT GTT

Proviamo a riformulare il problema

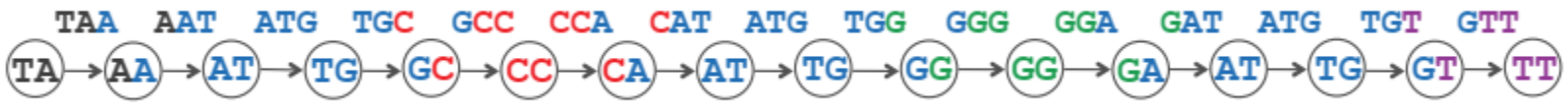
Ripartiamo dalla nostra sequenza

TAATGCCATGGGATGTT

estrando i trimeri

TAA AAT ATG TGC GCC CCA CAT ATG TGG GGG GGA GAT ATG TGT GTT

Ora invece di assegnare i 3-meri ai nodi assegnamoli agli archi, mentre ai nodi assegnamo i 2-meri sovrapposti



Proviamo a riformulare il problema

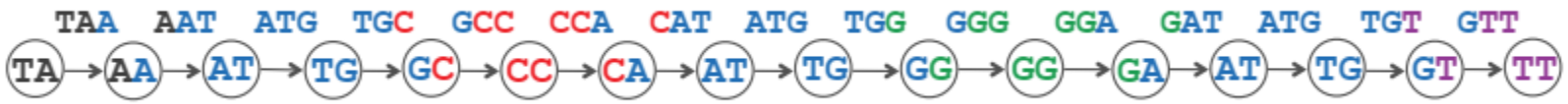
Ripartiamo dalla nostra sequenza

TAATGCCATGGGATGTT

estrando i trimeri

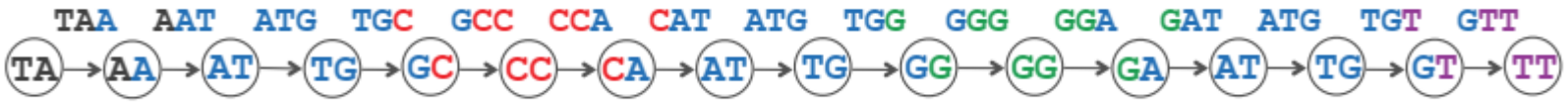
TAA AAT ATG TGC GCC CCA CAT ATG TGG GGG GGA GAT ATG TGT GTT

Ora invece di assegnare i 3-meri ai nodi assegnamoli agli archi, mentre ai nodi assegnamo i 2-meri sovrapposti

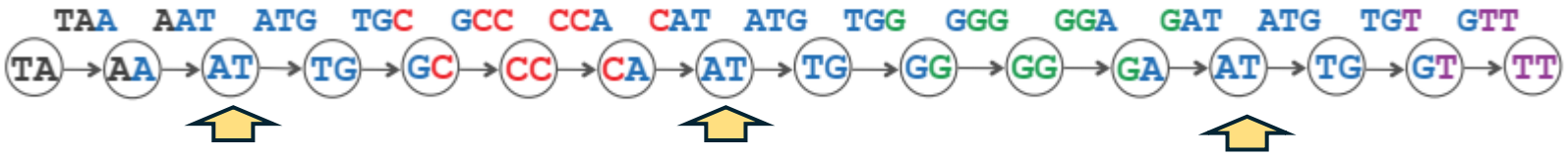


Prima (nell'overlap-graph) avevamo invece

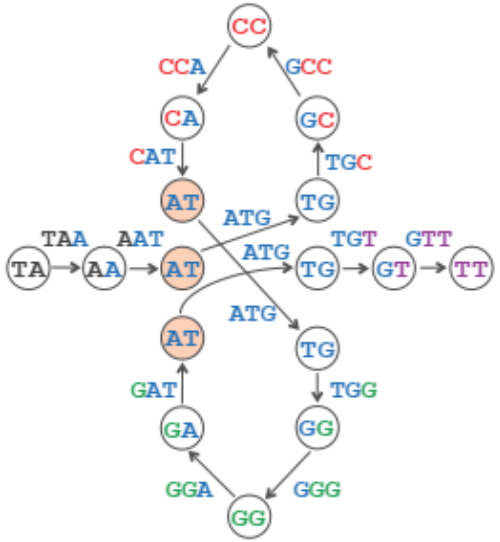
Proviamo a riformulare il problema

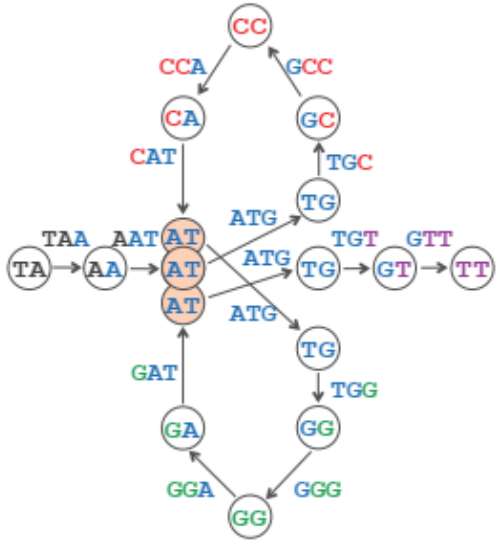


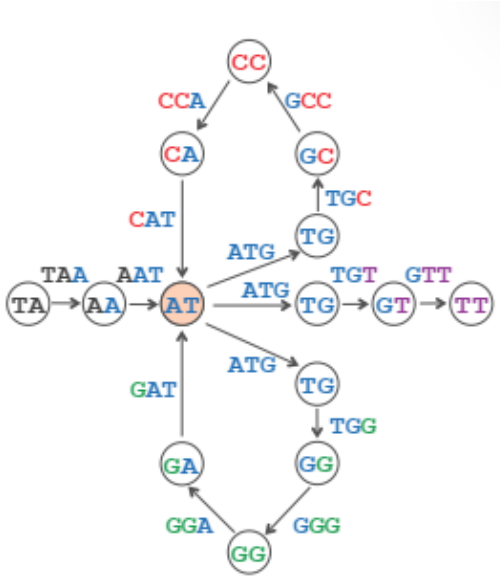
Proviamo a riformulare il problema



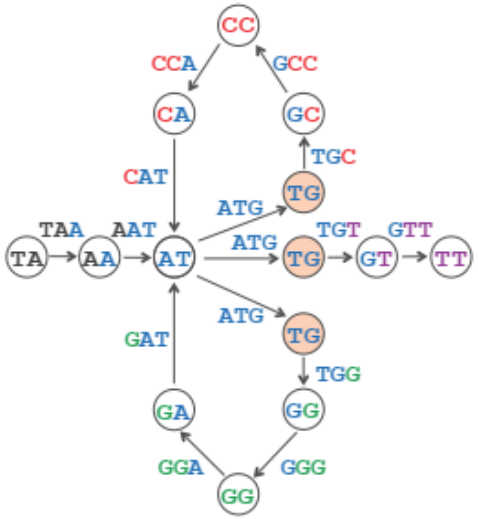
Ora, **«incolliamo»** i nodi uguali. Partiamo da **AT:**

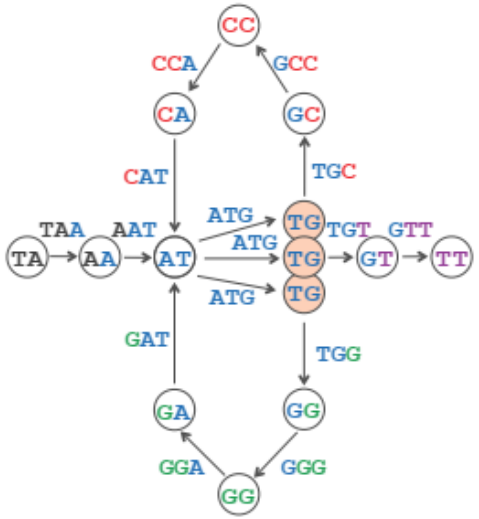


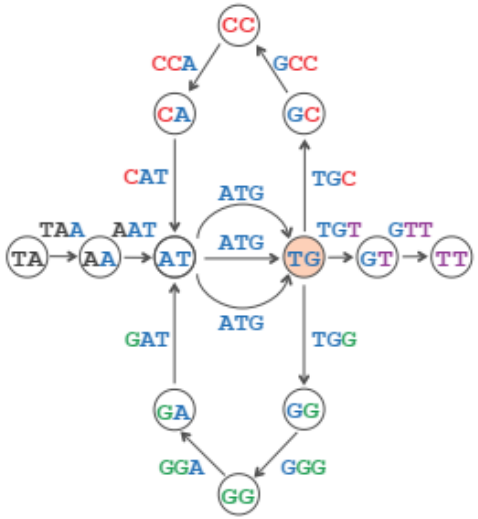




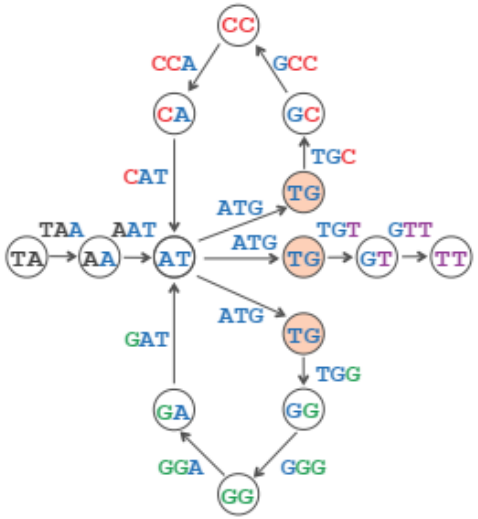
Proseguiamo **«incollando» TG..**

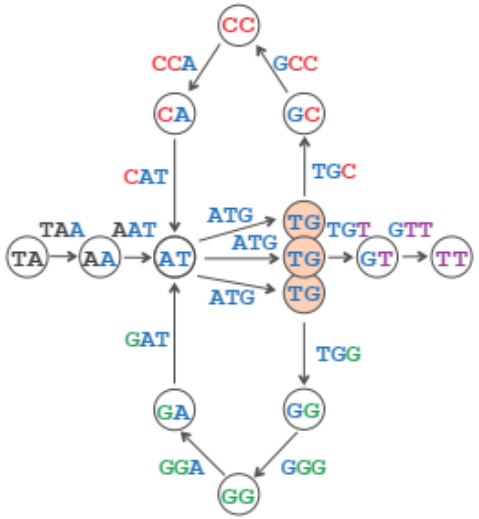


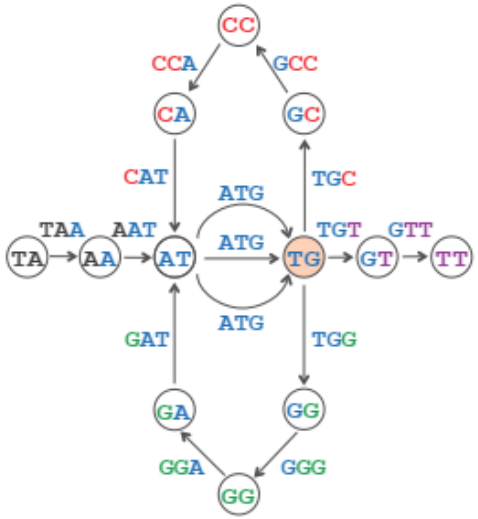




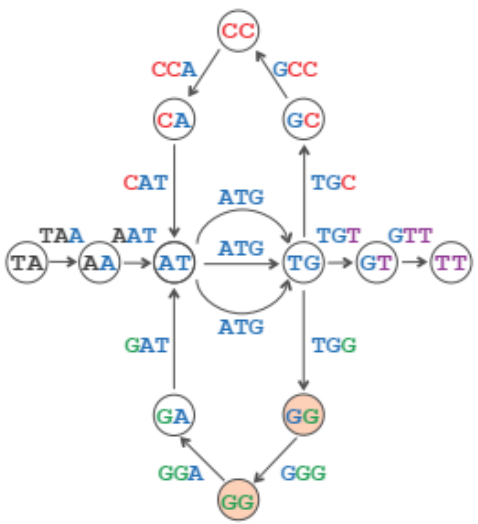
Proseguiamo **«incollando» TG..**

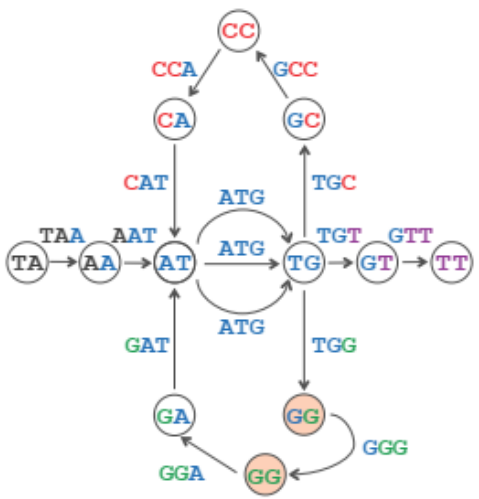


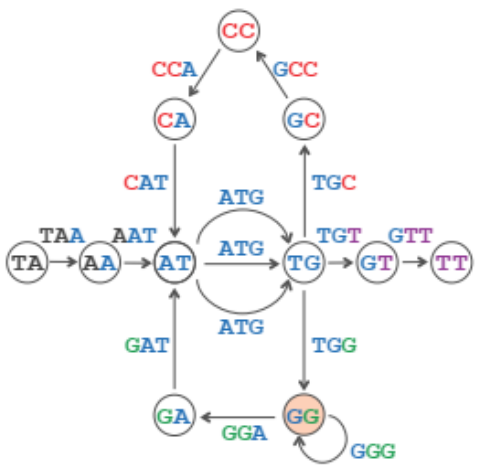




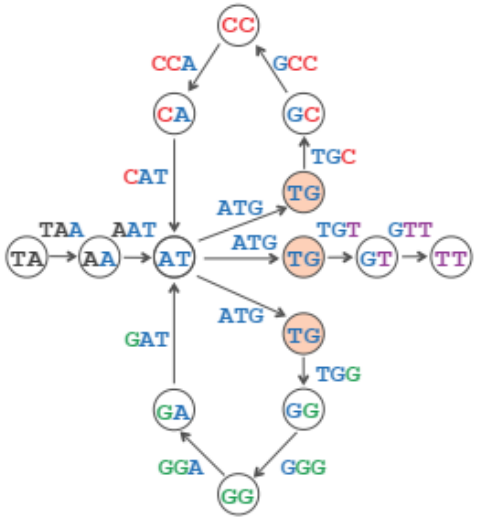
e poi **GG:**

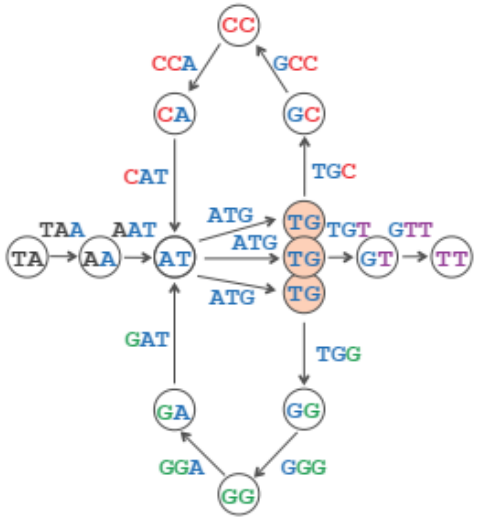


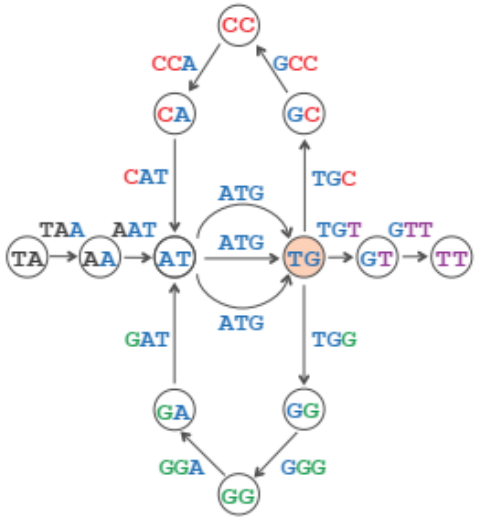




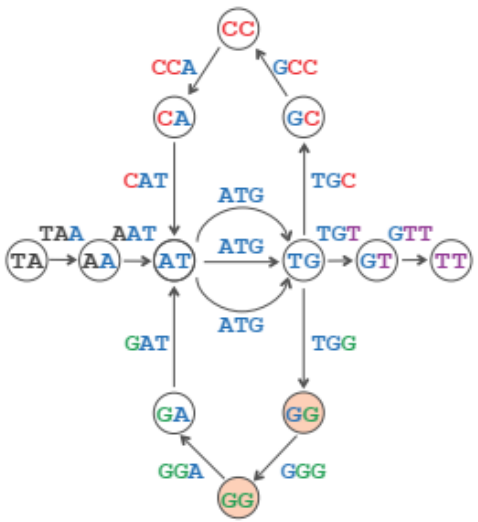
Proseguiamo **«incollando» TG..**

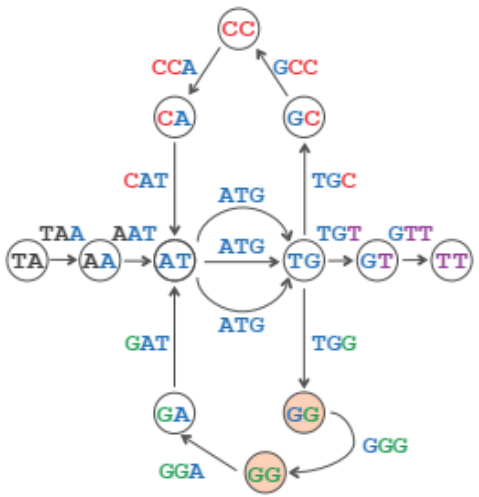


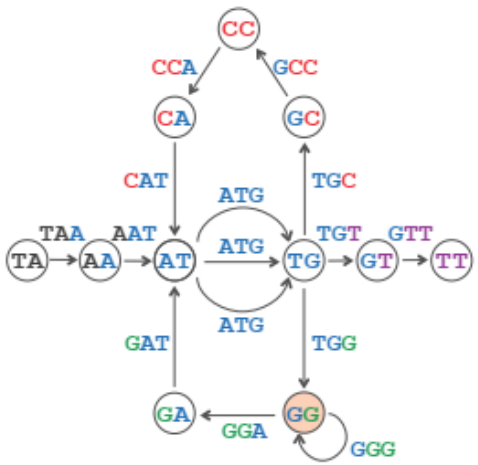




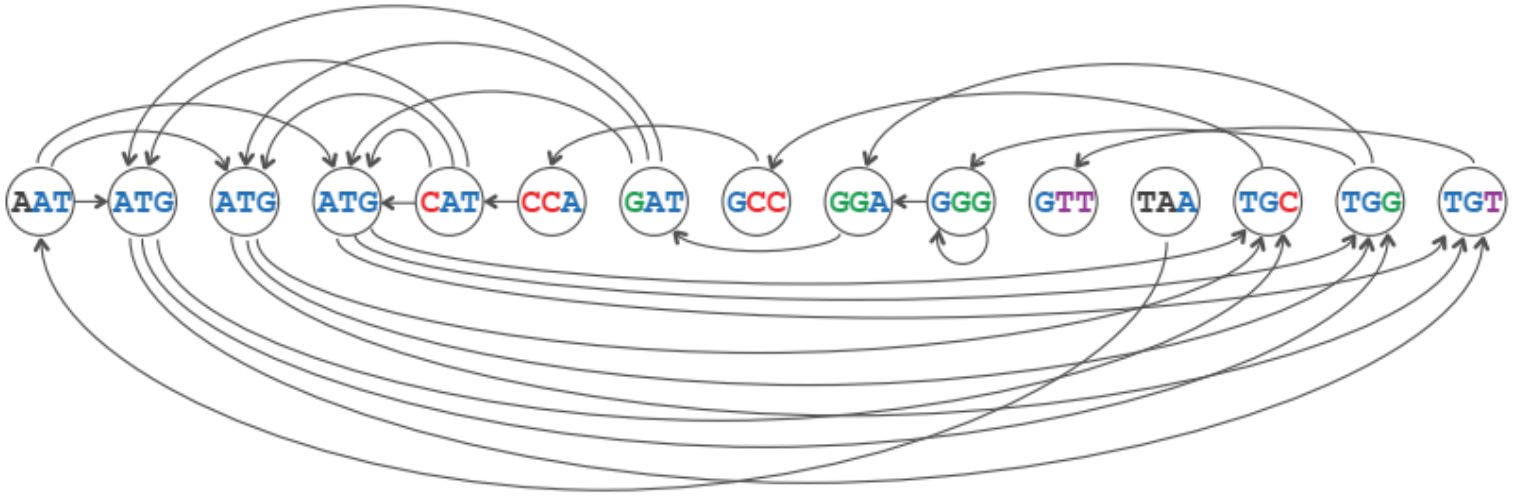
e poi **GG.**



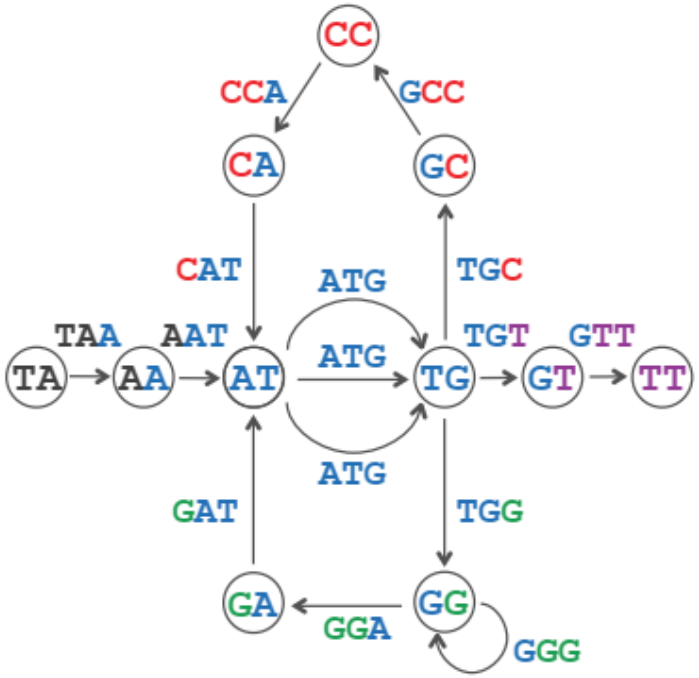




..e allora?!

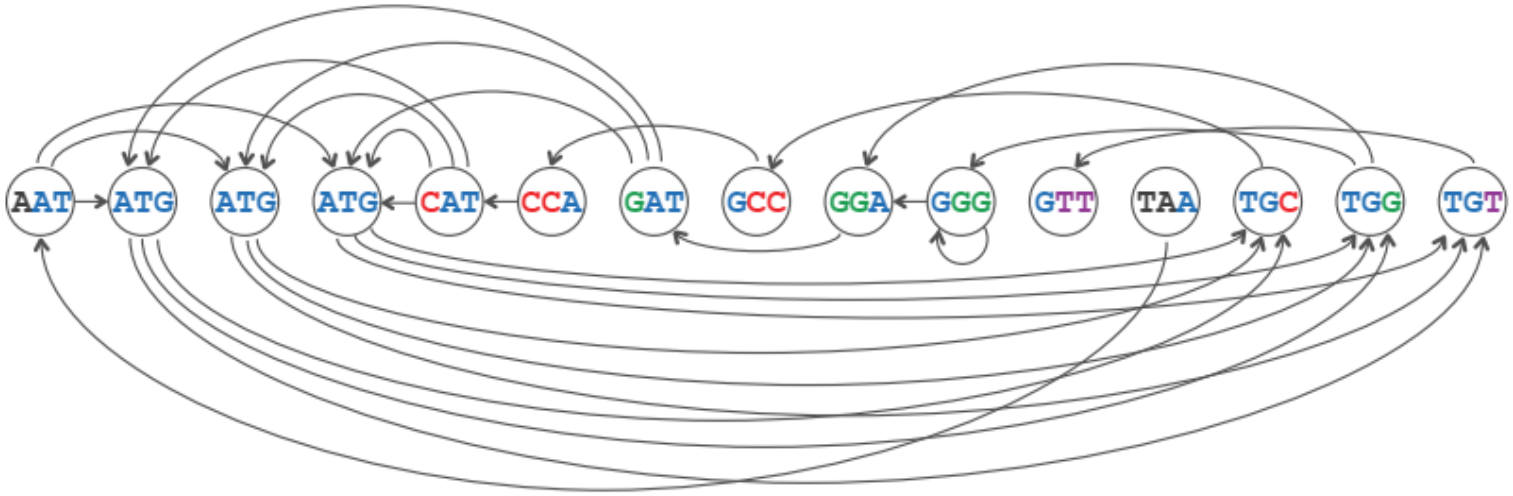


Overlap graph



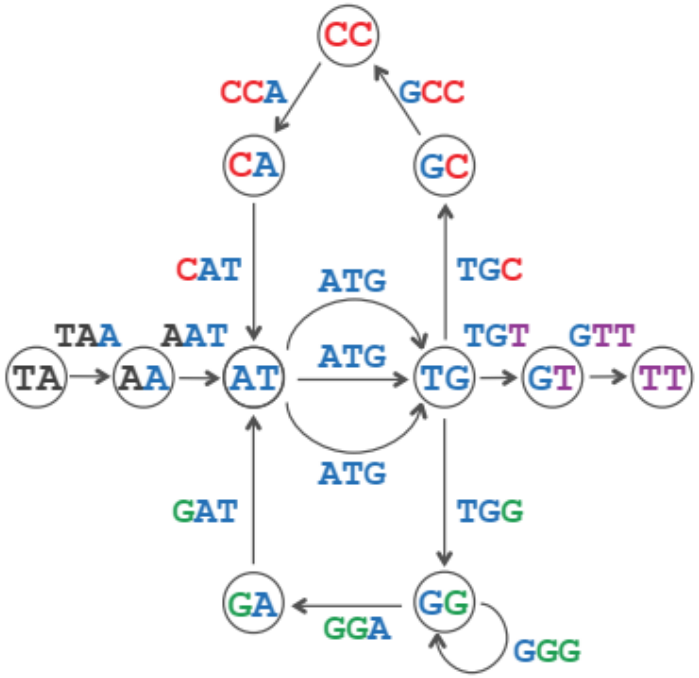
De Brujin graph

FIGURE 3.17 The overlap graph (top) and de Bruijn graph (bottom) for the same collection of 3-mers.



Overlap graph

Voi quale preferireste risolvere?

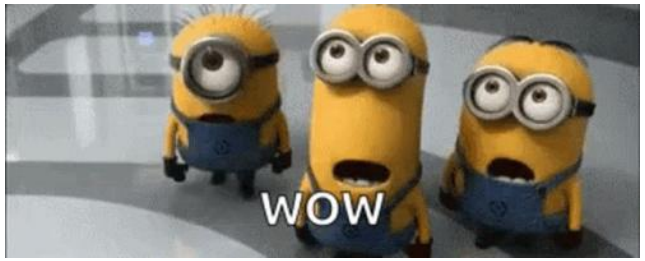


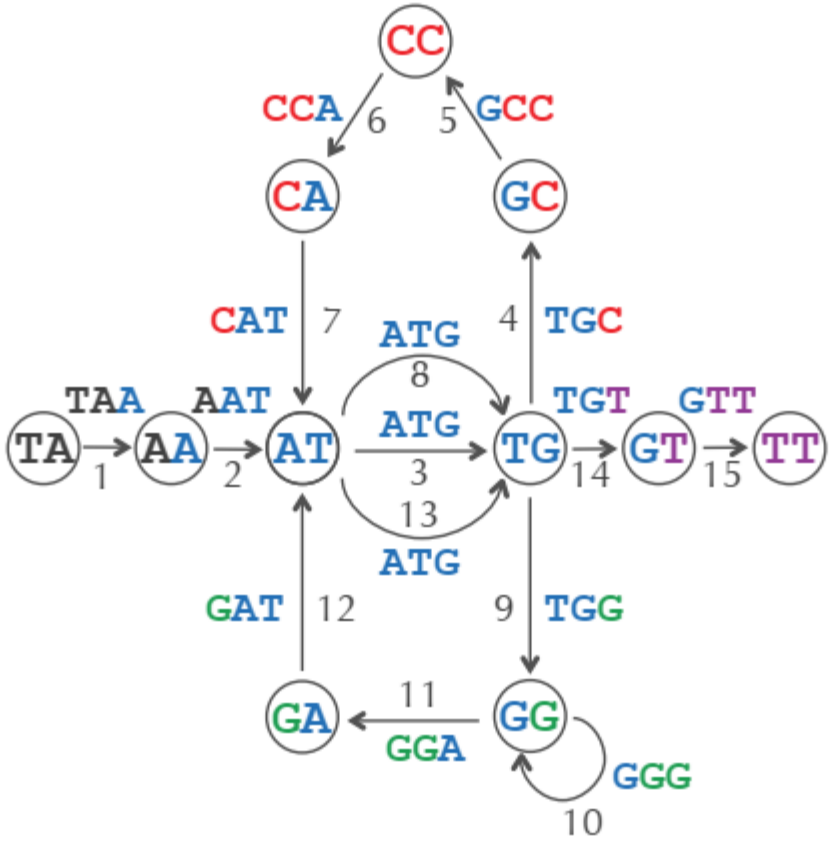
De Brujin graph

FIGURE 3.17 The overlap graph (top) and de Bruijn graph (bottom) for the same collection of 3-mers.

I grafi di De Brujin sono apparentemente piu' «semplici»*

*possono essere risolti
in tempo breve (polinomiale)

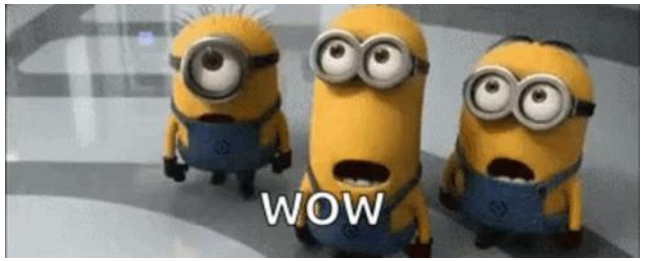


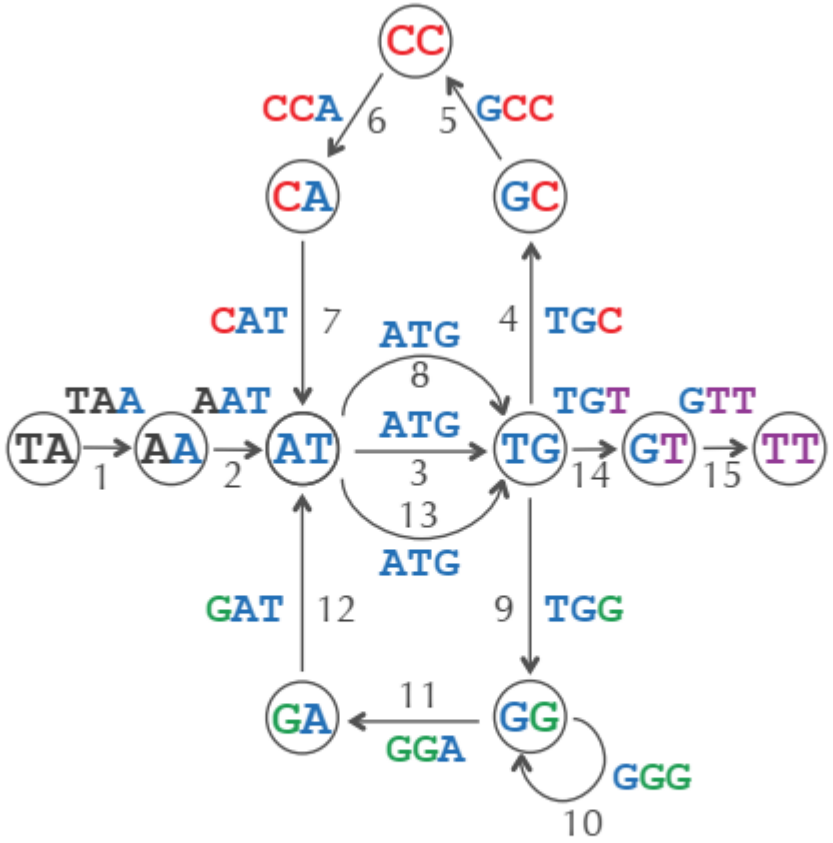


Il percorso per ricostruire la sequenza **TAATGCCATGGGATGTT** è indicato dai numeri dall'1 al 15. Notate che il percorso non passa mai dalla stessa freccia.

I grafi di De Brujin sono apparentemente piu' «semplici»*

*possono essere risolti
in tempo breve (polinomiale)

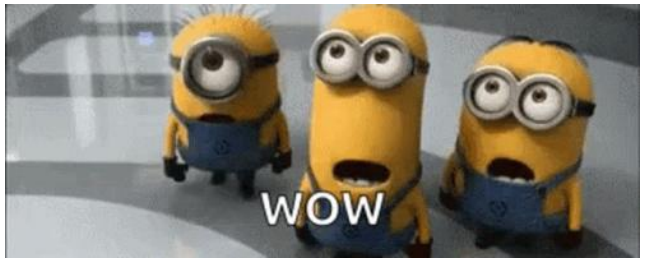




Il percorso per ricostruire la sequenza **TAATGCCATGGGATGTT** è indicato dai numeri dall'1 al 15. Notate che il percorso non passa mai dalla stessa freccia.

I grafi di De Brujin sono apparentemente piu' «semplici»*

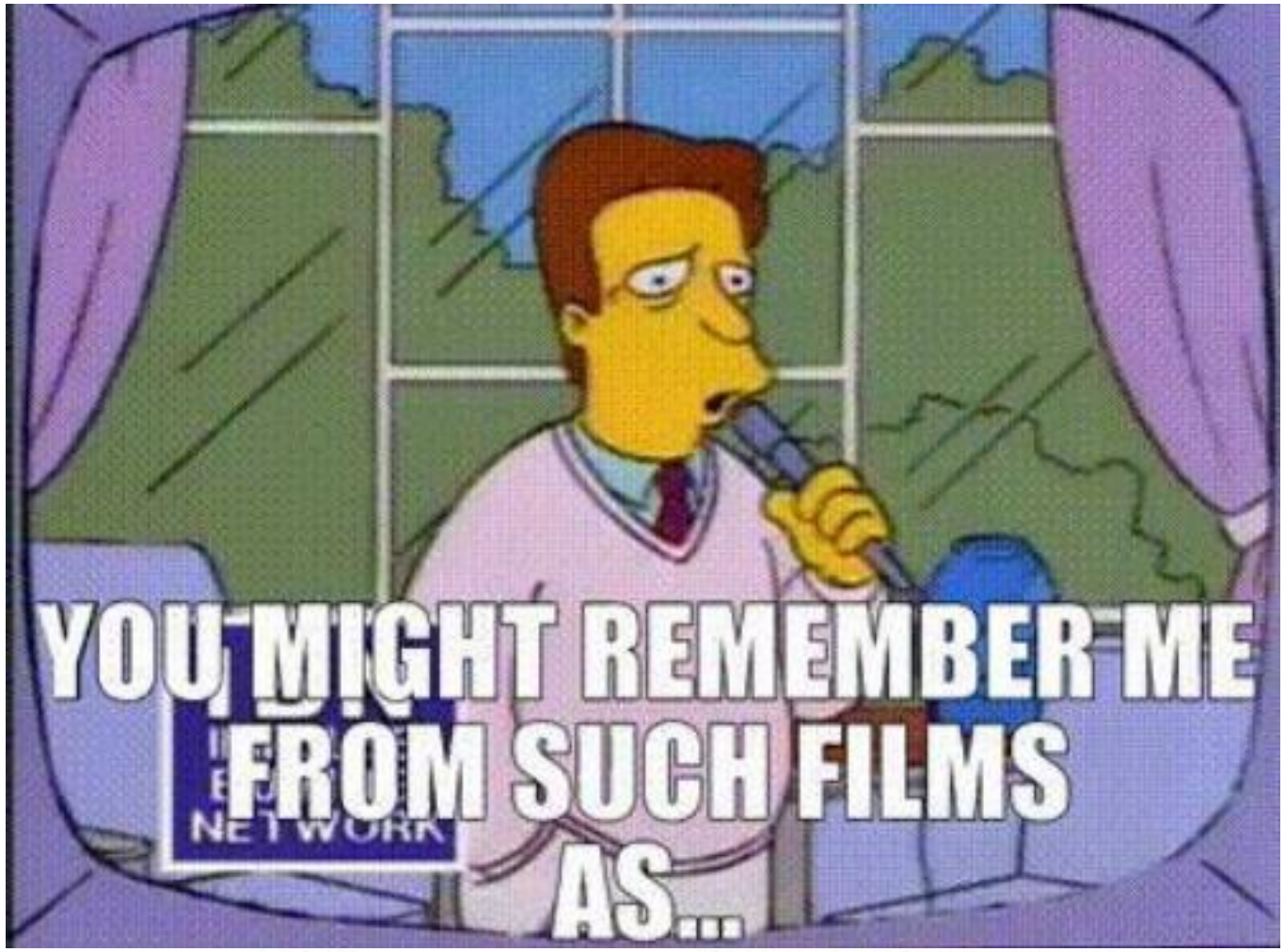
*possono essere risolti
in tempo breve (polinomiale)



(Parentesi storica che non dovete necessariamente ricordare ma che è carina)

FIGURE 3.47 Leonhard Euler (left), William Hamilton (center), and Nicolaas de Bruijn (right).

Euler (1707-1783)

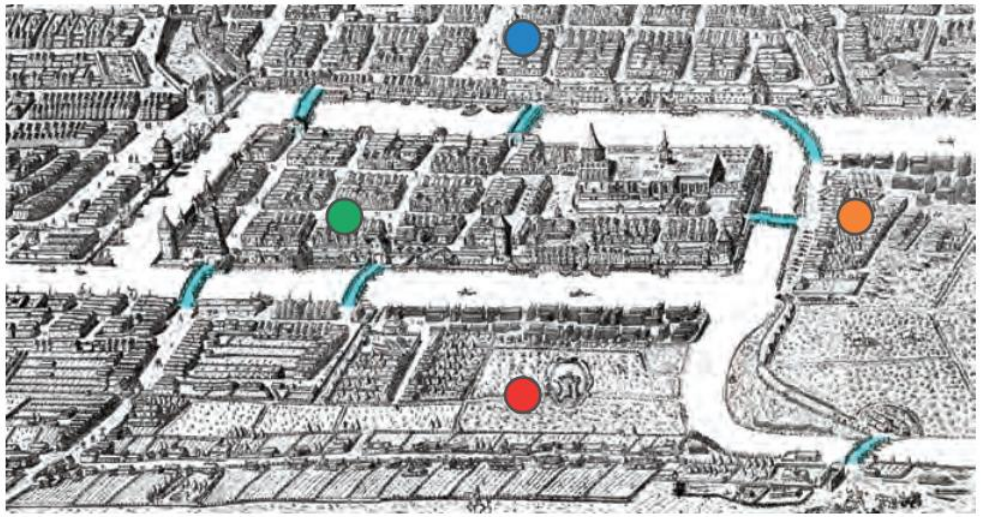


..la notazione *f(x)* per le funzioni;

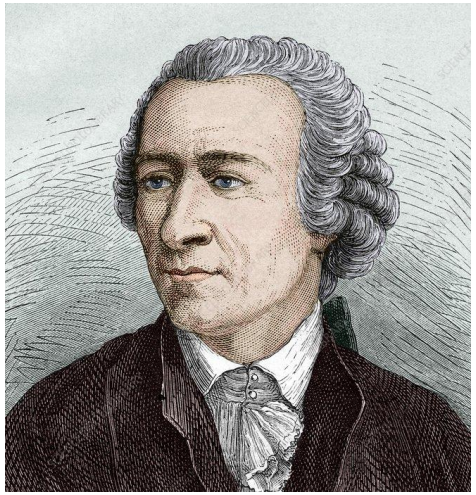
..la notazione *i* per la radice quadrata di -1;

..essere diventato cieco dall'occhio destro nel 1735, aver continuato a lavorare ininterrottamente, e dopo aver perso l'occhio sinistro nel 1766 aver commentato «Ora avrò meno distrazioni» ed aver continuato a pubblicare centinaia di pubblicazioni.

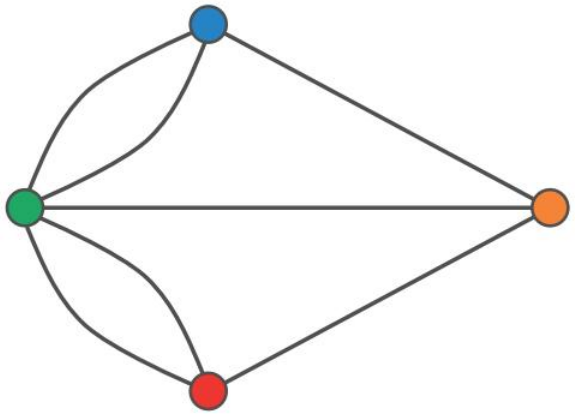
I 7 ponti di Königsberg



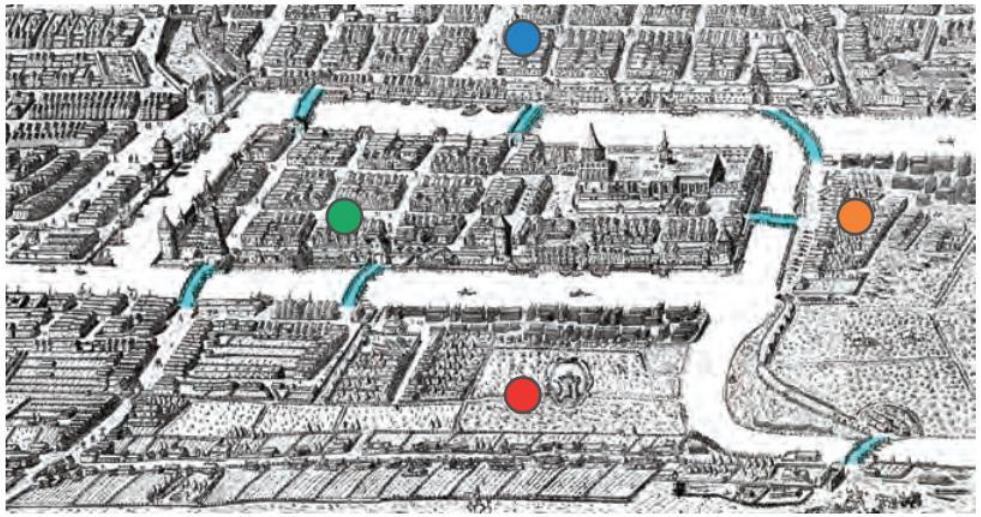
Come posso passare per i 7 ponti della città una sola volta e tornare indietro senza ripassare dallo stesso ponte?



I nodi indicano le **4 aree della citta'** da visitare, e gli archi i **7 ponti**

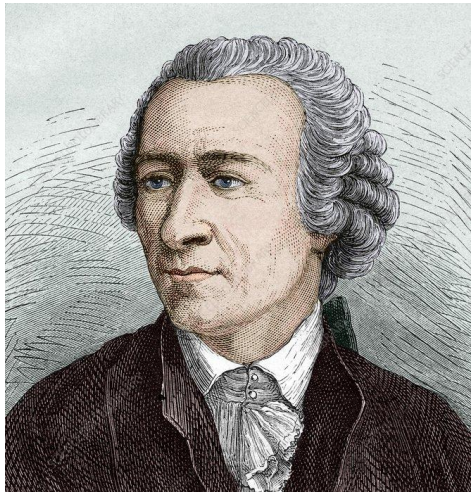


I 7 ponti di Königsberg

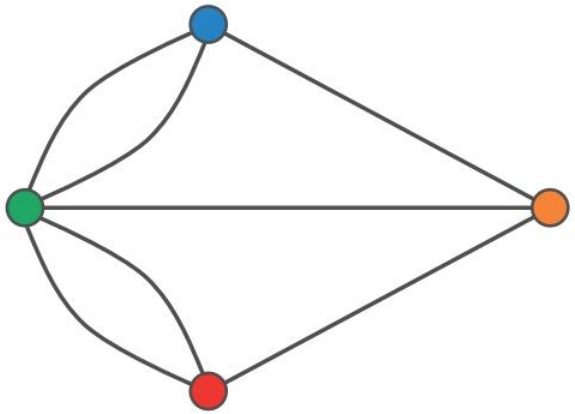


Il ciclo Euleriano

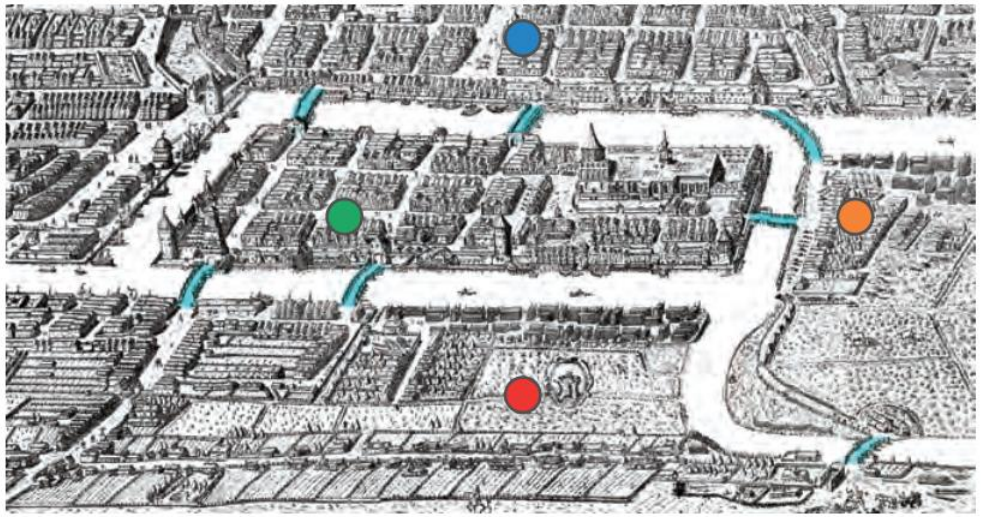
Come posso passare per i 7 ponti della città una sola volta e tornare indietro senza ripassare dallo stesso ponte?

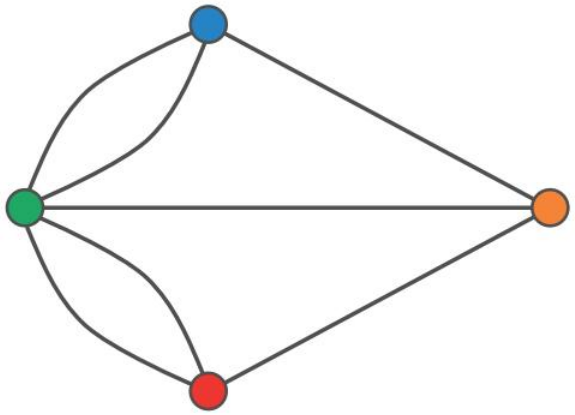


I nodi indicano le **4 aree della citta'** da visitare, e gli archi i **7 ponti**



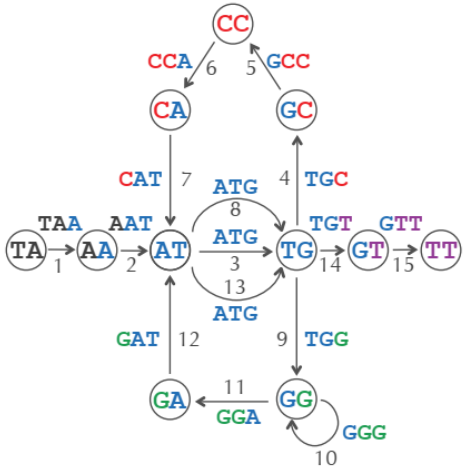
I 7 ponti di Königsberg

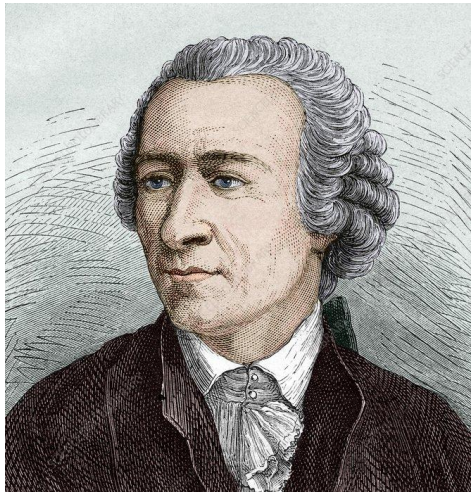




Il ciclo Euleriano

Come posso passare per i 7 ponti della città una sola volta e tornare indietro senza ripassare dallo stesso ponte?

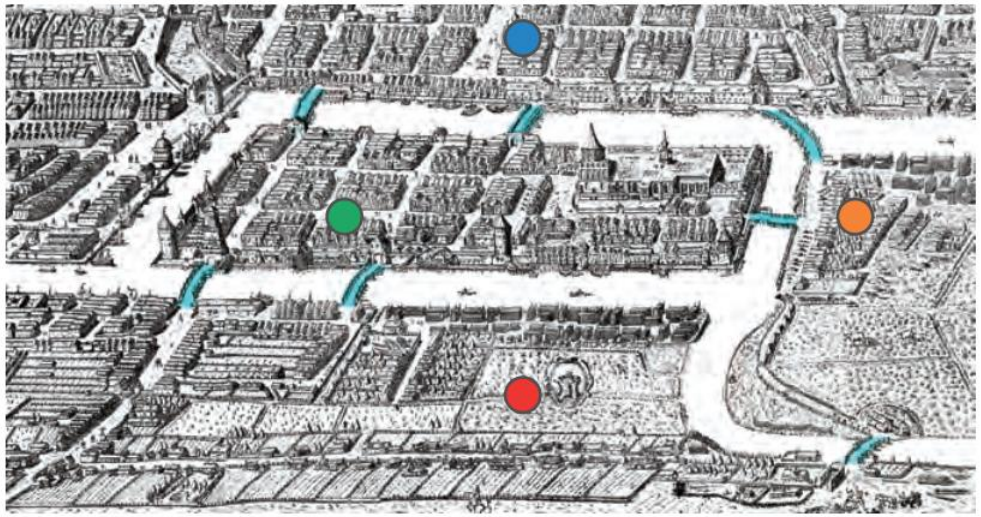


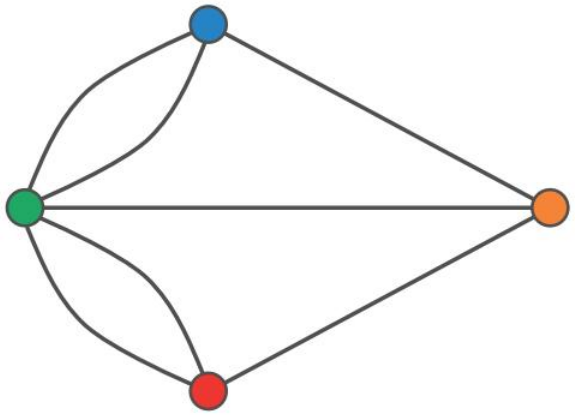


E' un problema simile al nostro!

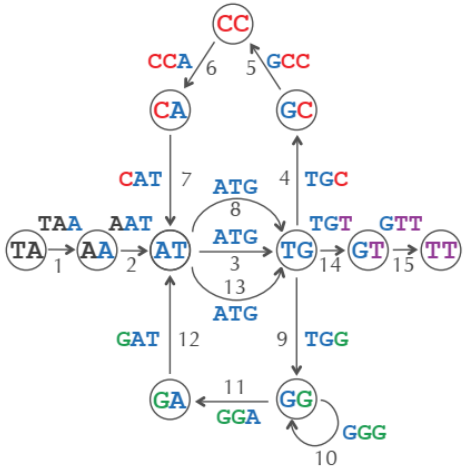
Il percorso per ricostruire la sequenza **TAATGCCATGGGATGTT** è indicato dai numeri dall'1 al 15.
Il percorso **non ripassa mai dalla stessa freccia**

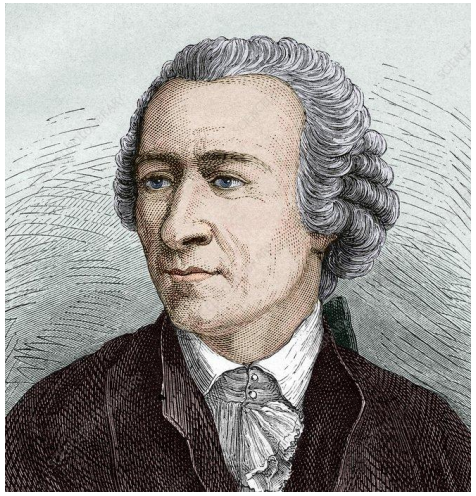
I 7 ponti di Königsberg





Il ciclo Euleriano





E' un problema simile al nostro!

Il percorso per ricostruire la sequenza **TAATGCCATGGGATGTT** è indicato dai numeri dall'1 al 15.

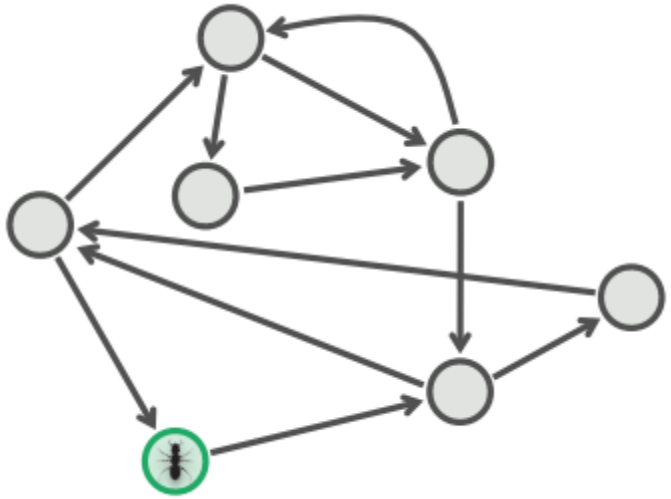
Il percorso **non ripassa mai dalla stessa freccia***

*il teorema di Eulero dimostra come sia sempre possibile trovare un ciclo Euleriano in un grafo «**bilanciato**», cioè dove per ogni nodo ci siano lo stesso numero di frecce che puntano dentro e fuori

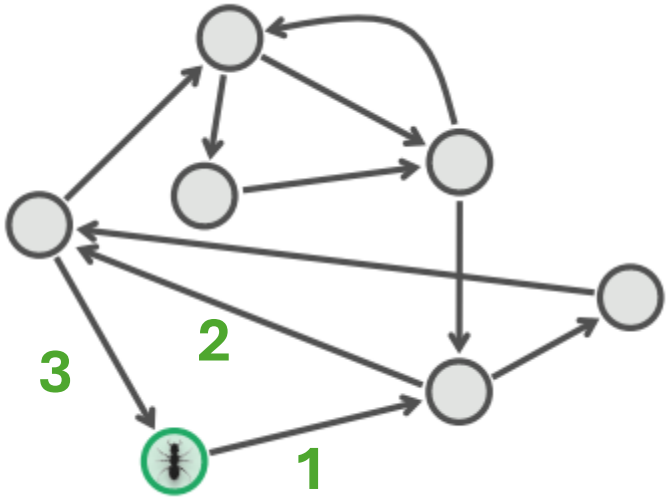
Ma come
lo troviamo
questo
percorso??

Ma come lo troviamo questo percorso??

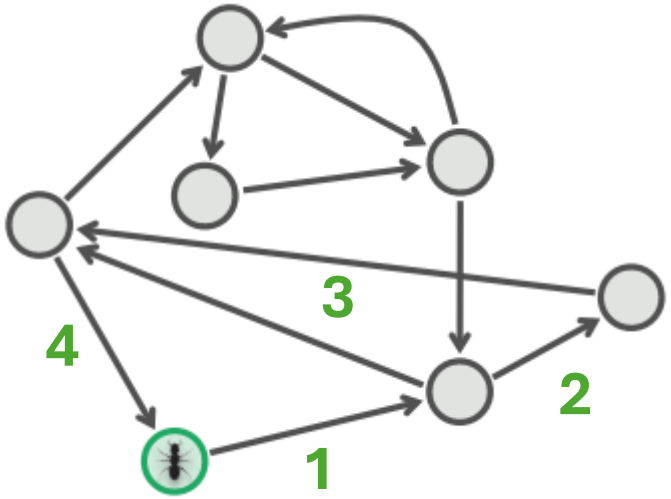
La formica Leo cerca il suo ciclo Euleriano per cercare da mangiare ovunque



La formica Leo cerca il suo ciclo Euleriano per cercare da mangiare ovunque

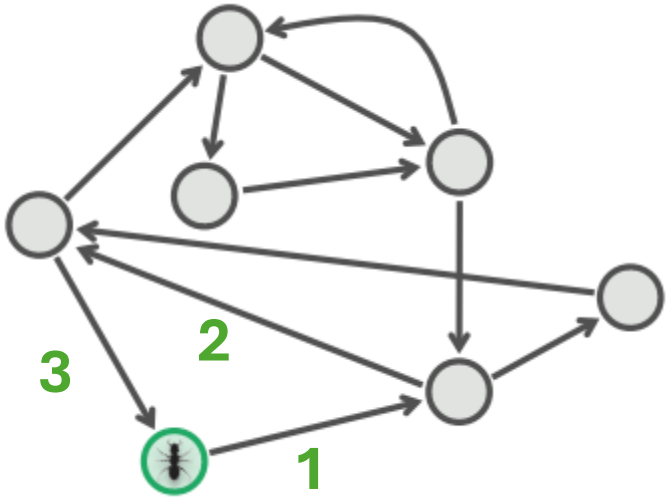


Ogni tanto si blocca..

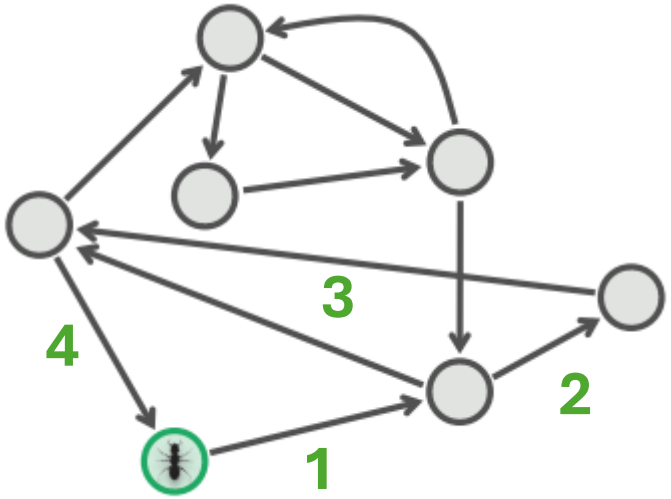


..e si blocca

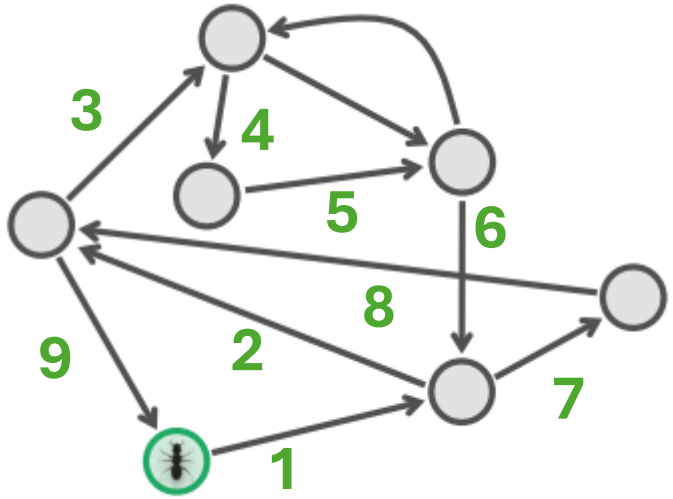
La formica Leo cerca il suo ciclo Euleriano per cercare da mangiare ovunque



Ogni tanto si blocca..



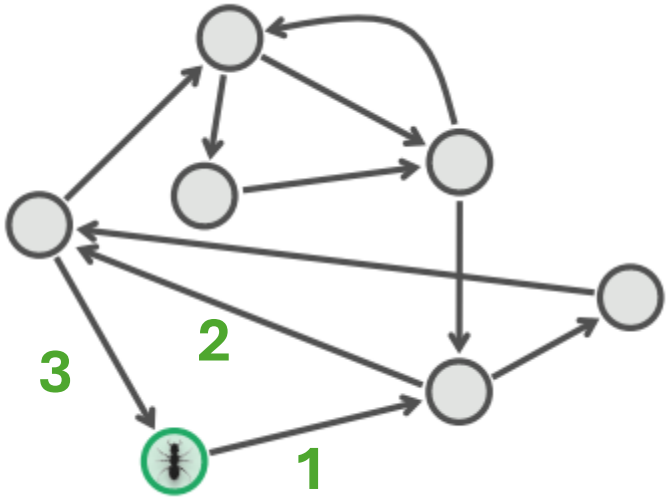
..e si blocca



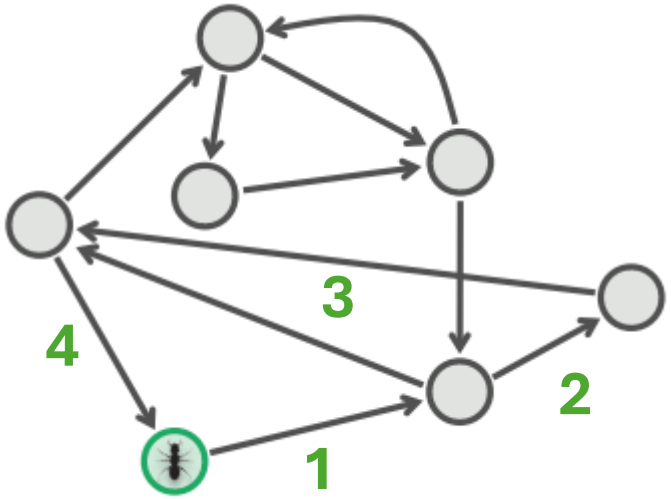
..e si riblocca...

Cosa notate?

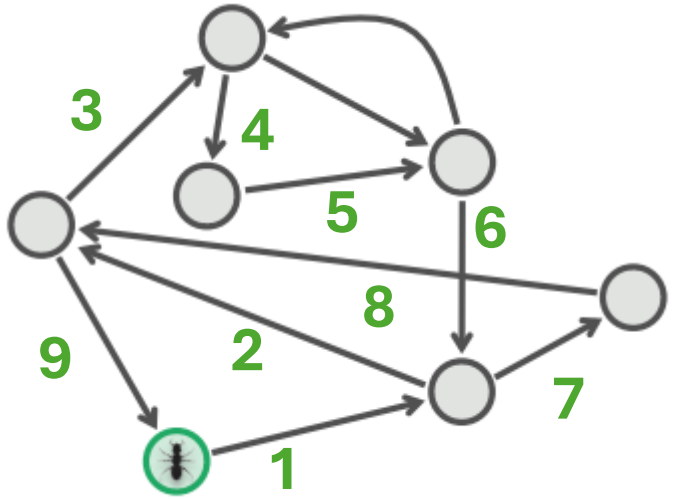
La formica Leo cerca il suo ciclo Euleriano per cercare da mangiare ovunque



Ogni tanto si blocca..



..e si blocca

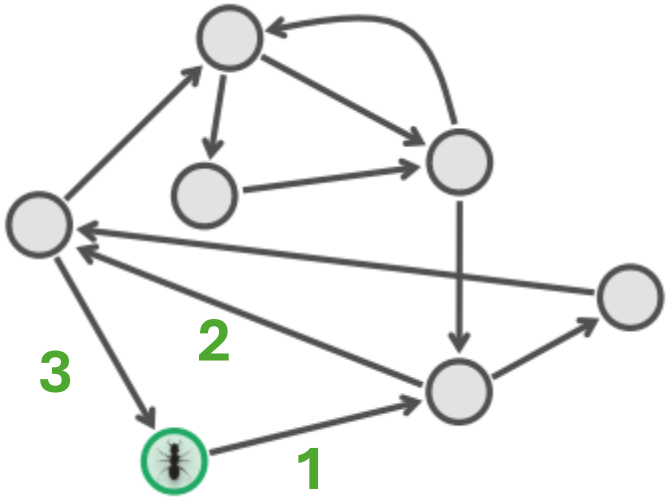


..e si riblocca...

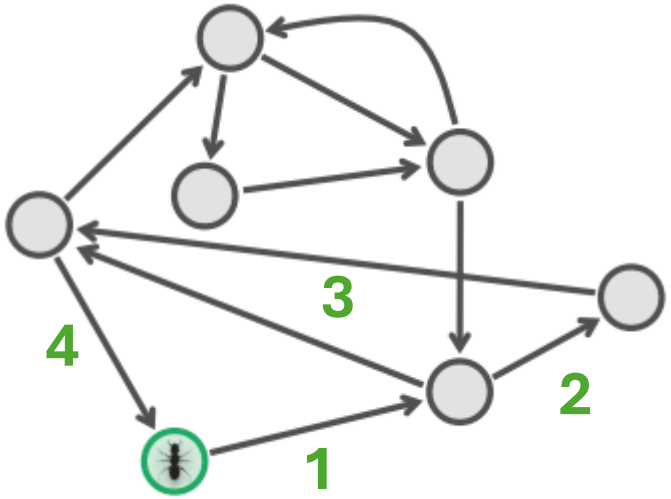
...tornando sempre al nodo di partenza!*

*se il grafo è bilanciato

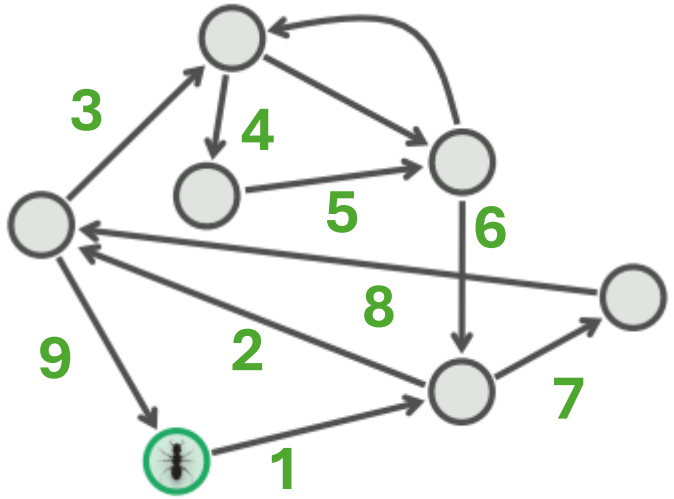
La formica Leo cerca il suo ciclo Euleriano per cercare da mangiare ovunque



Ogni tanto si blocca..



..e si blocca

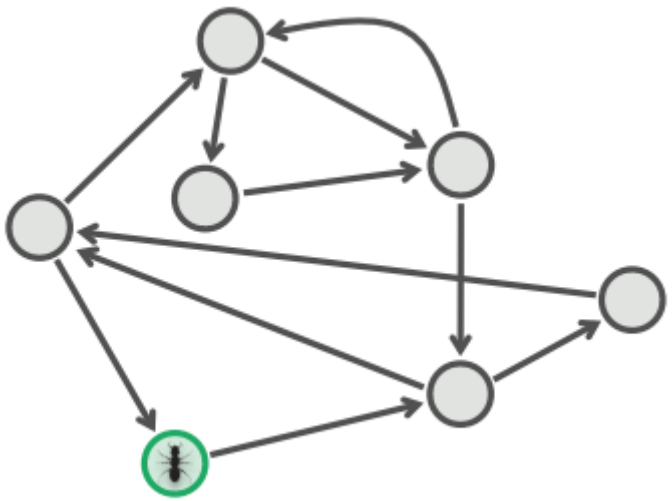


..e si riblocca...

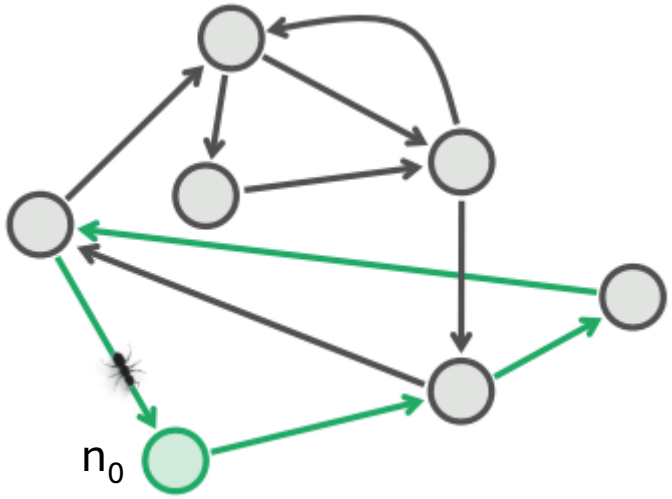
...tornando sempre al nodo di partenza!

Questo ci suggerisce una strategia/algoritmo per aiutare Leo!!

Una strategia per aiutare Leo

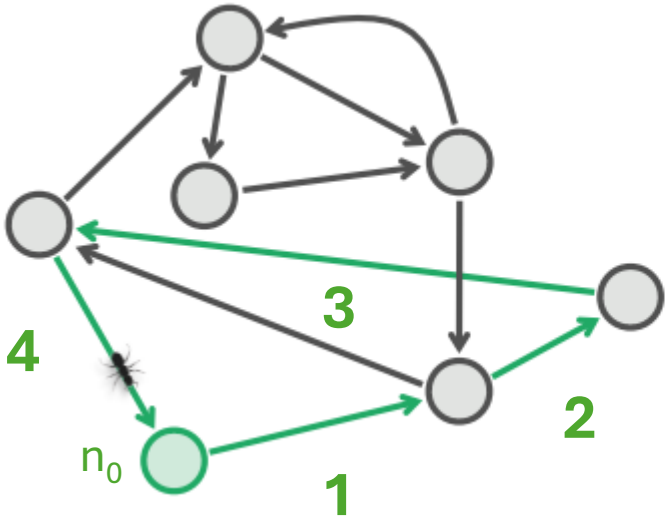


Una strategia per aiutare Leo

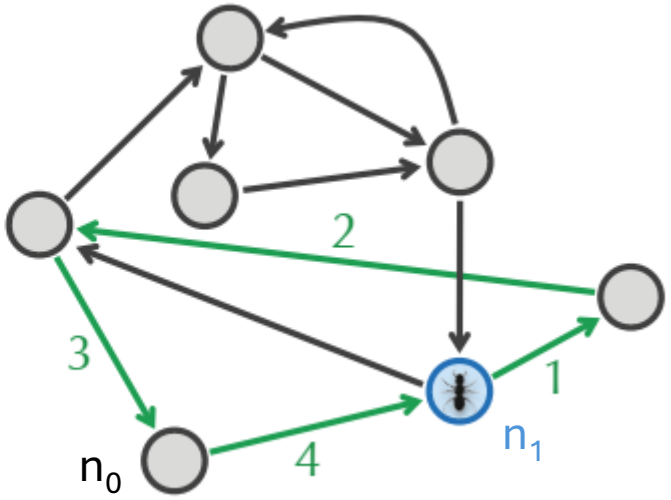


Partendo dal nodo n_0
Leo prova un primo percorso ($Ciclo_0$)

Una strategia per aiutare Leo



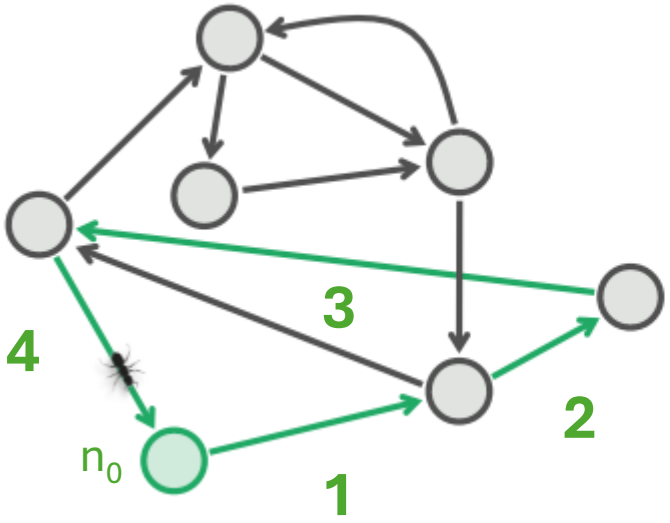
Partendo dal nodo n_0
Leo prova un primo percorso ($Ciclo_0$)



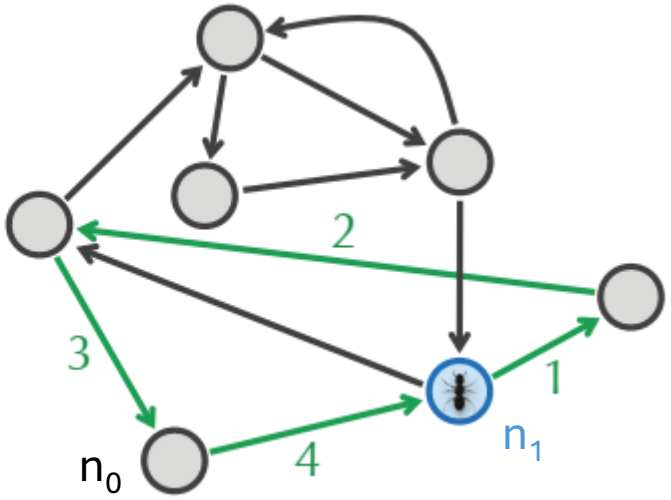
Ritorna poi sui suoi passi, fino al primo nodo n_1 da cui poteva prendere piu' di una strada

Una strategia per aiutare Leo

Notate: Ora riparte da n_1 (non piu' da n_0), quindi il tratto 2 del percorso precedente diventa il tratto 1



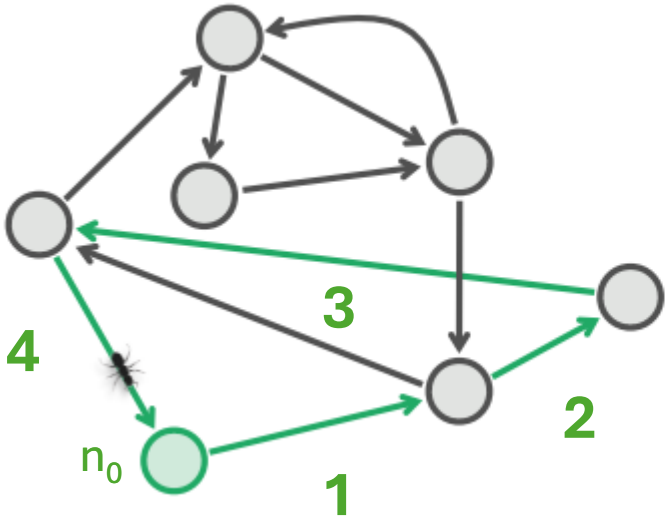
Partendo dal nodo n_0 Leo prova un primo percorso ($Ciclo_0$)



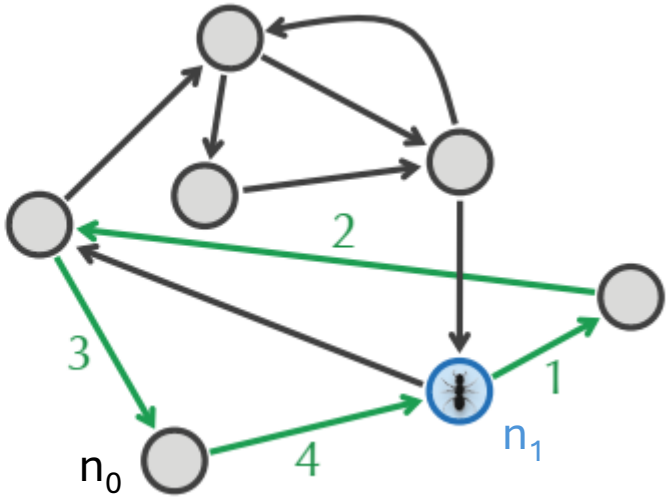
Ritorna poi sui suoi passi, fino al primo nodo n_1 da cui poteva prendere piu' di una strada

Una strategia per aiutare Leo

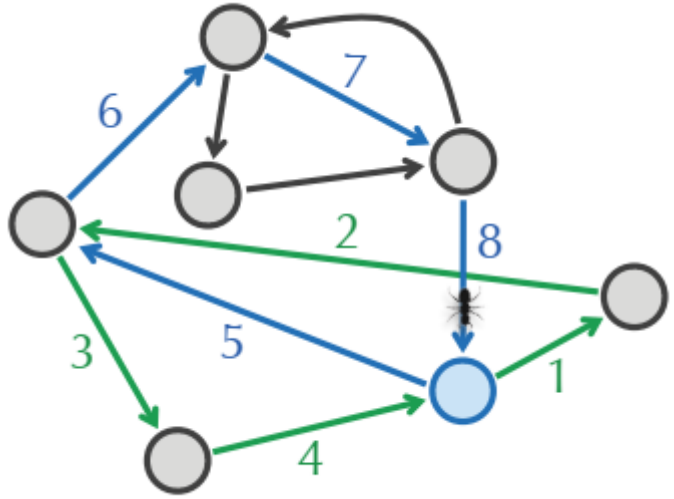
Notate: Tornato ad n_1 puo' ora proseguire per la strada non percorsa precedentemente (in blu)



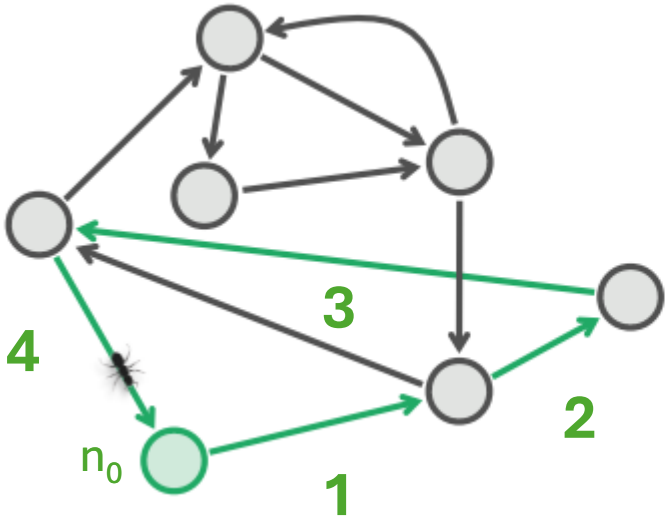
Partendo dal nodo n_0 Leo prova un primo percorso ($Ciclo_0$)



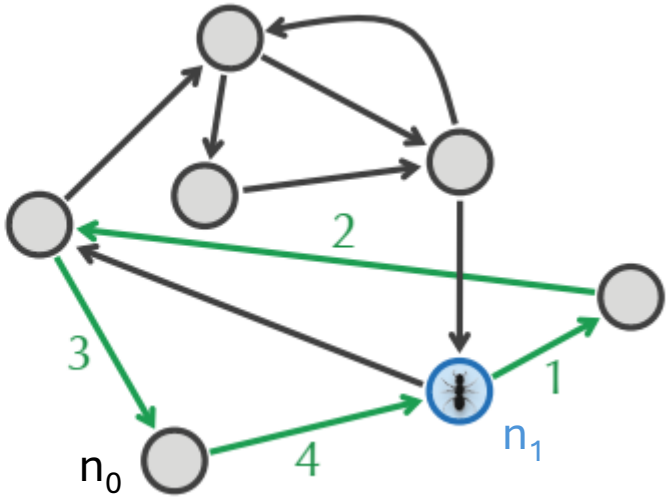
Ritorna poi sui suoi passi, fino al primo nodo n_1 da cui poteva prendere piu' di una strada



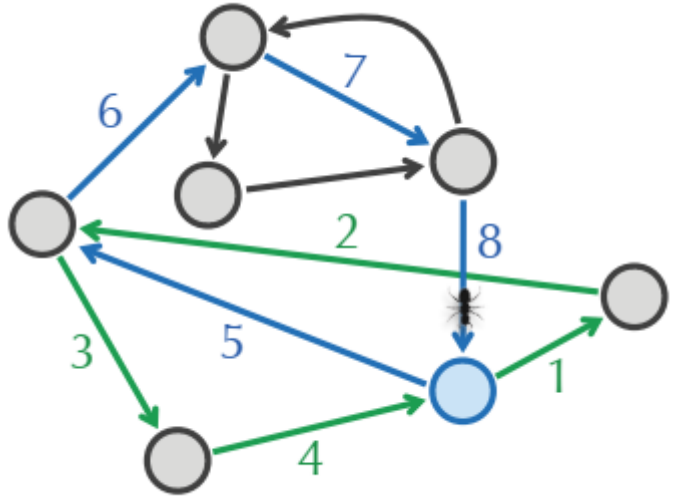
Una strategia per aiutare Leo



Partendo dal nodo n_0 Leo prova un primo percorso ($Ciclo_0$)

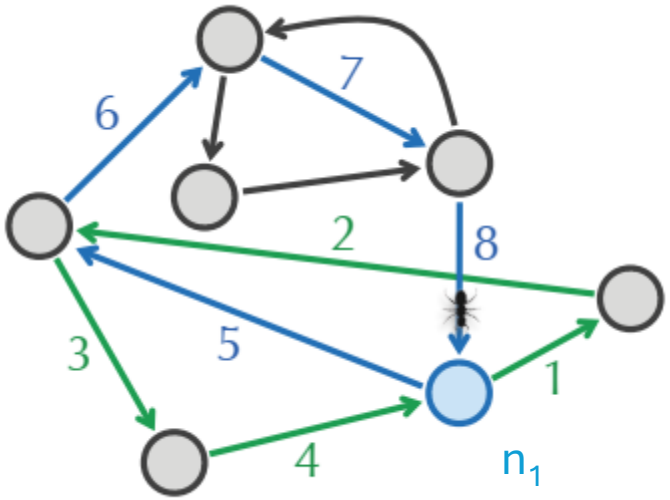


Ritorna poi sui suoi passi, fino al primo nodo n_1 da cui poteva prendere piu' di una strada



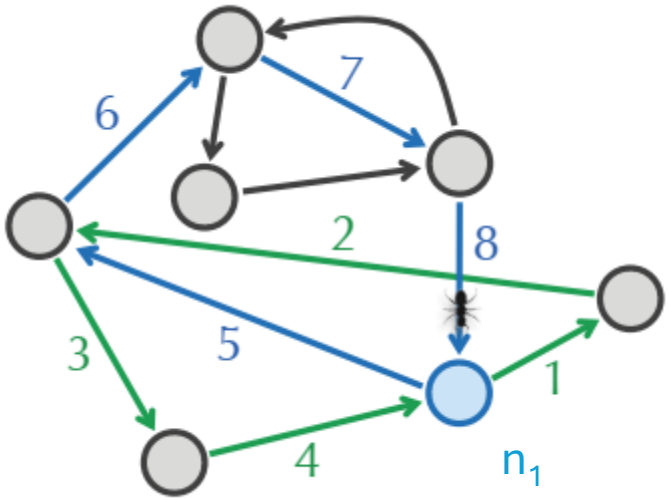
Da n_1 percorre un nuovo ciclo $Ciclo_1$ piu' largo, composto dal vecchio $Ciclo_0$ (ma partendo invece da n_1) piu' il nuovo percorso (blu)

Una strategia per aiutare Leo

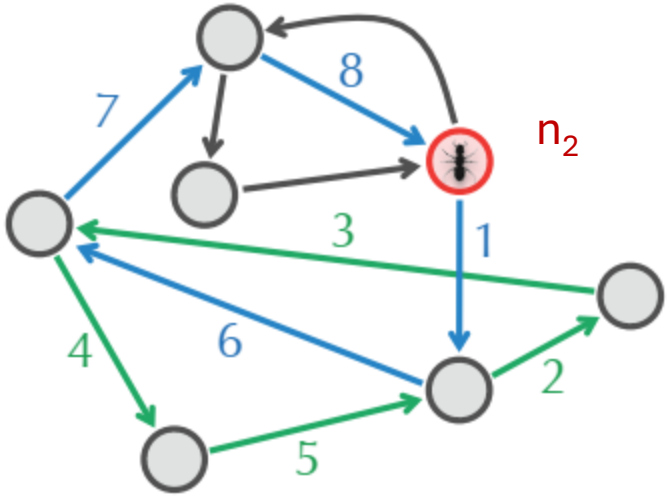


Leo è nuovamente bloccato (questa volta ad n_1).

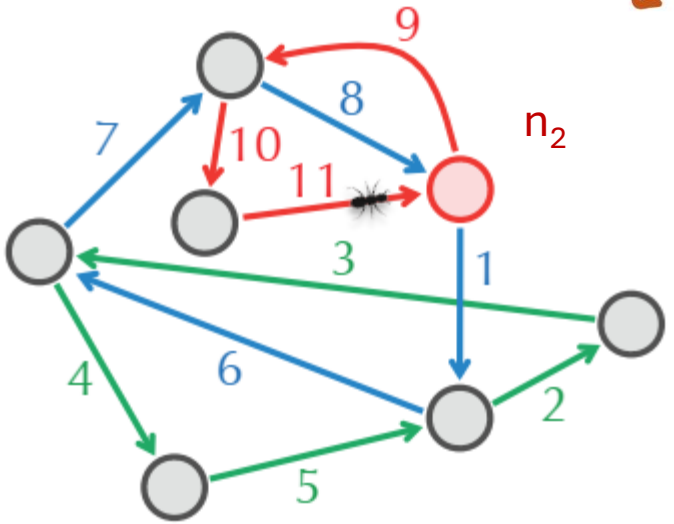
Una strategia per aiutare Leo



Leo è nuovamente bloccato (questa volta ad n_1).

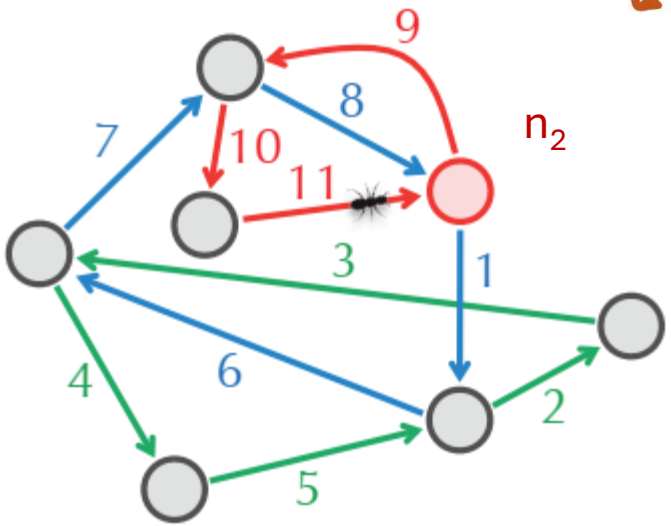


Ripete l'operazione, ripartendo dal nodo successivo con biforcazione (n_2). Rinizia il nuovo ciclo $Ciclo_2$ dalla parte «rimanente» di $Ciclo_1$ (quindi l'8 di $Ciclo_1$ diventa ora 1 di $Ciclo_2$).



Leo ha finalmente percorso tutti i nodi con $Ciclo_2$, che quindi è il ciclo Euleriano!

Quanto è «complesso» questo algoritmo?

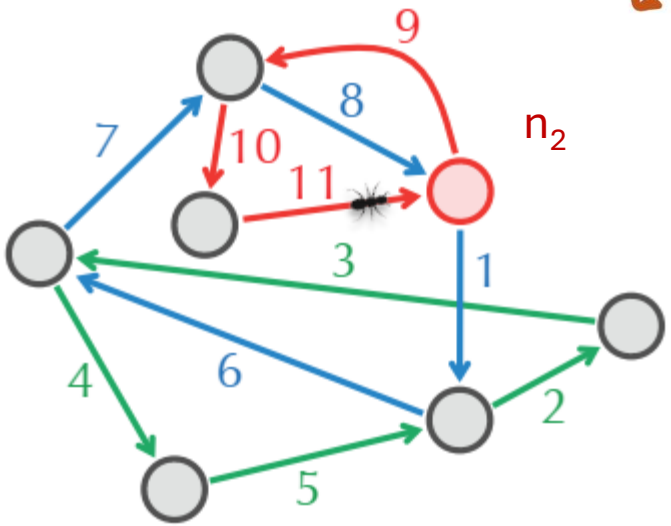


Quanto è «complesso» questo algoritmo?

Abbiamo esplorato 3 potenziali nodi di partenza (che sono al max il numero di nodi totali).

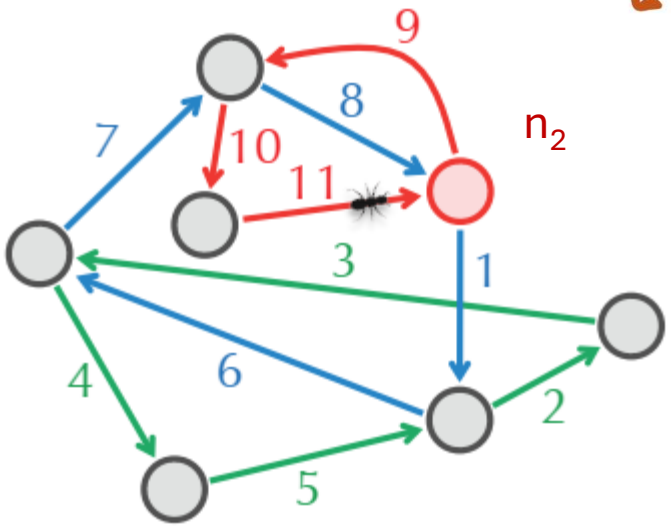
Quindi al crescere dei dati (del grafo) il tempo impiegato cresce ***SOLO* polinomialmente***

*a differenza della ricerca del percorso (Hamiltoniano) nell'overlap-graph che cresceva esponenzialmente!



Quanto è «complesso» questo algoritmo?

Abbiamo esplorato 3 potenziali nodi di partenza (che sono al max il numero di nodi totali).

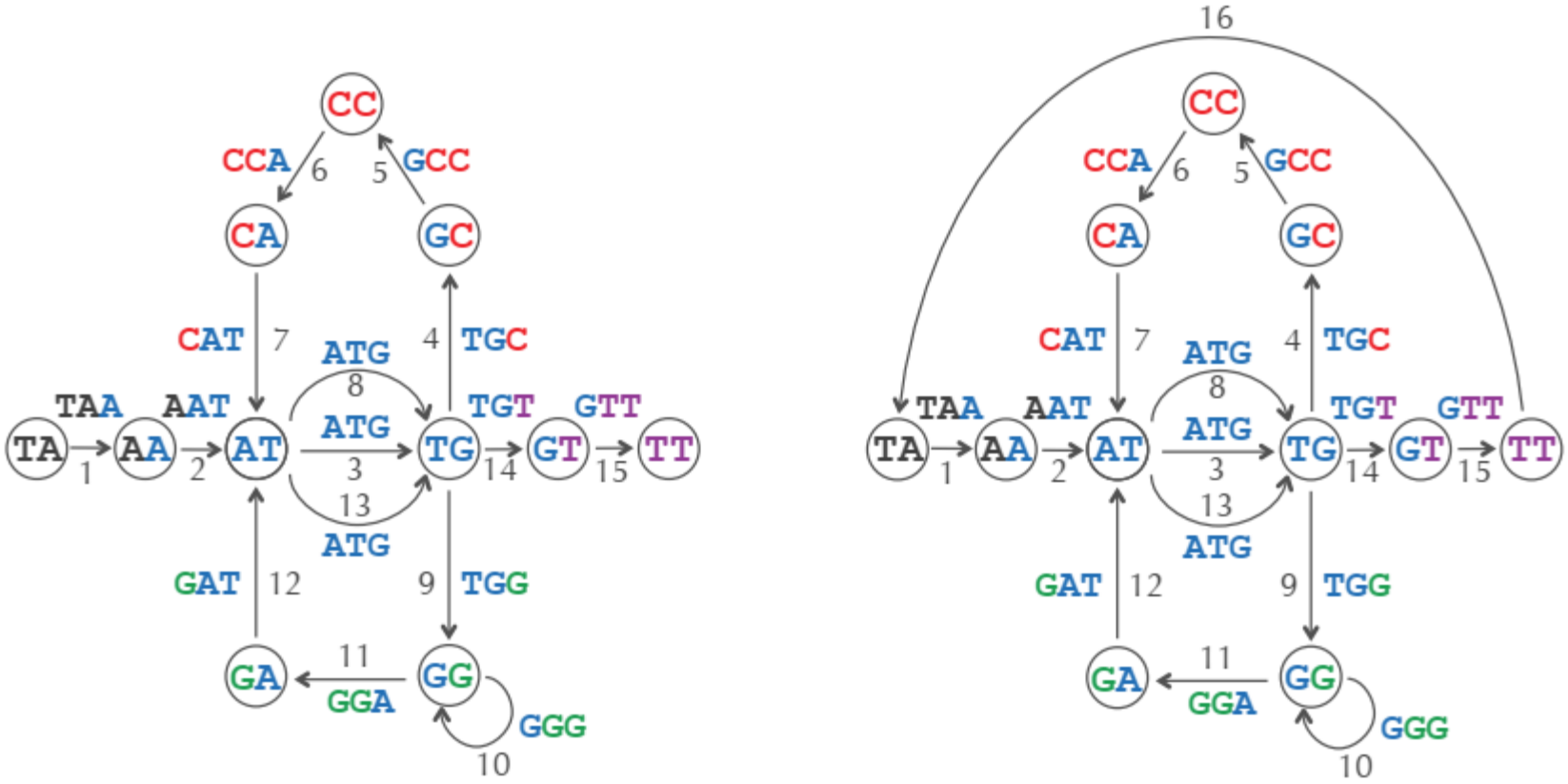


Quindi al crescere dei dati (del grafo) il tempo impiegato cresce ***SOLO* polinomialmente***

*a differenza della ricerca del percorso (Hamiltoniano) nell'overlap-graph che cresceva esponenzialmente!

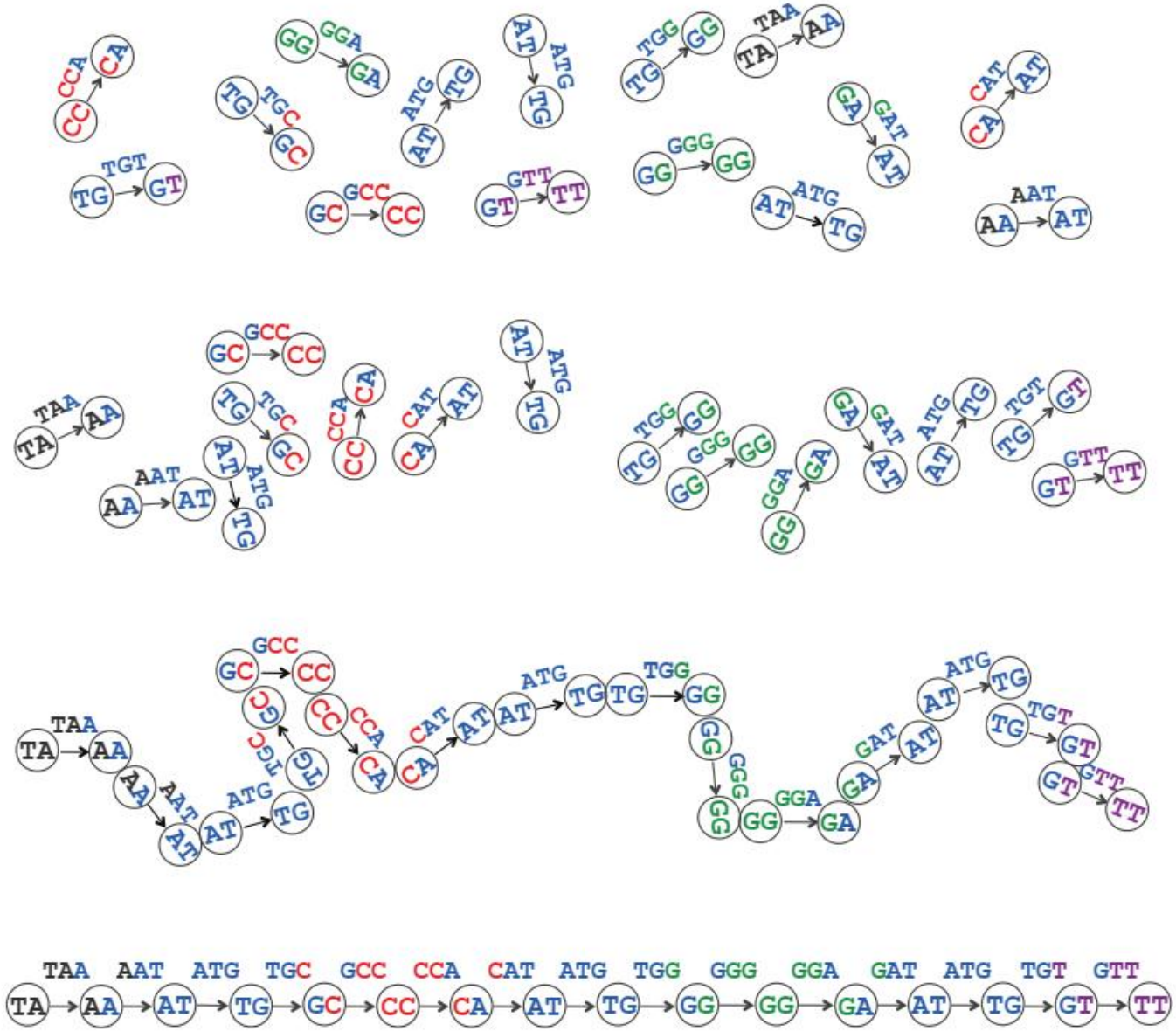
Ottimo, abbiamo un algoritmo per risolvere il problema di Leo!!! Ma il nostro? Era proprio lo stesso problema?

Differenza 1: Nel caso di Leo cercavamo un ciclo Euleriano, ma per genomi lineari non abbiamo un ciclo



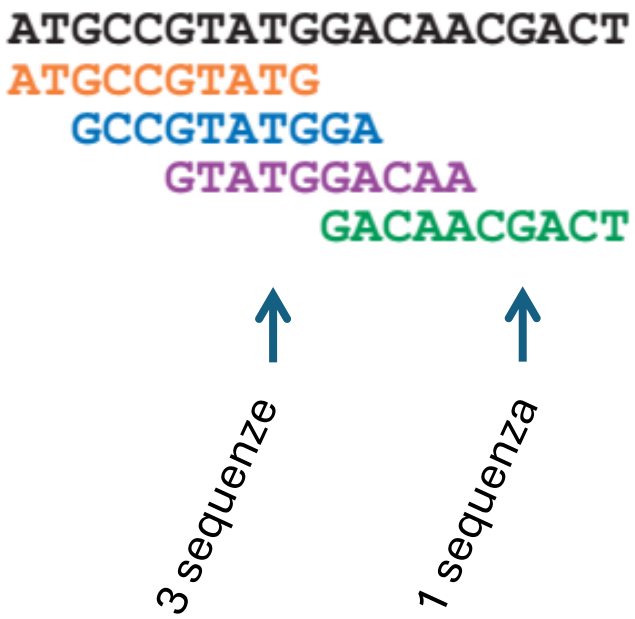
Non importa, basta congiungere inizio e fine della sequenza per avere un «grafo bilanciato»: ora il problema equivale a cercare un ciclo.

Differenza 2: Eravamo partiti da una grafo di De Brujin con sequenza già conosciuta. Ma generalmente non la conosciamo. Come costruirla?

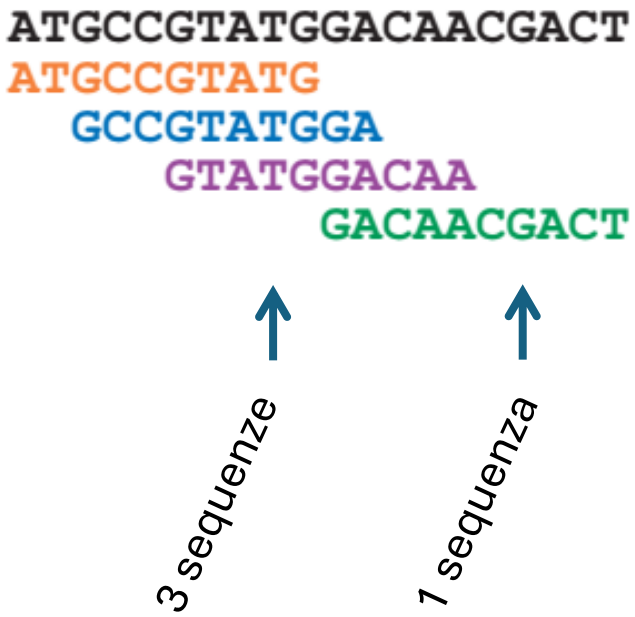


Basta incollare i nodi uguali!!!

Differenza 3: nei dati di sequenziamento la “coverage” non è uniforme



Differenza 3: nei dati di sequenziamento la “coverage” non è uniforme



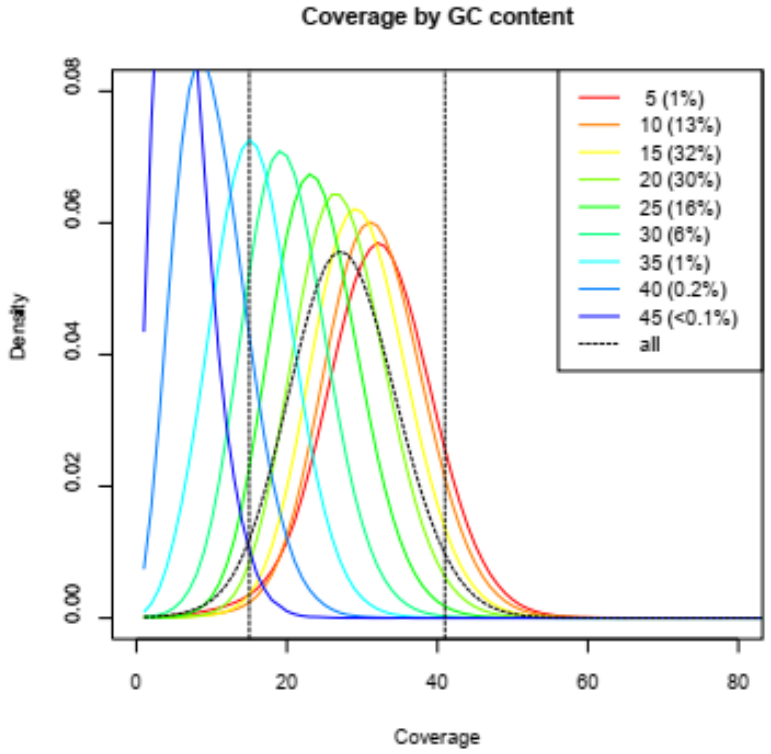
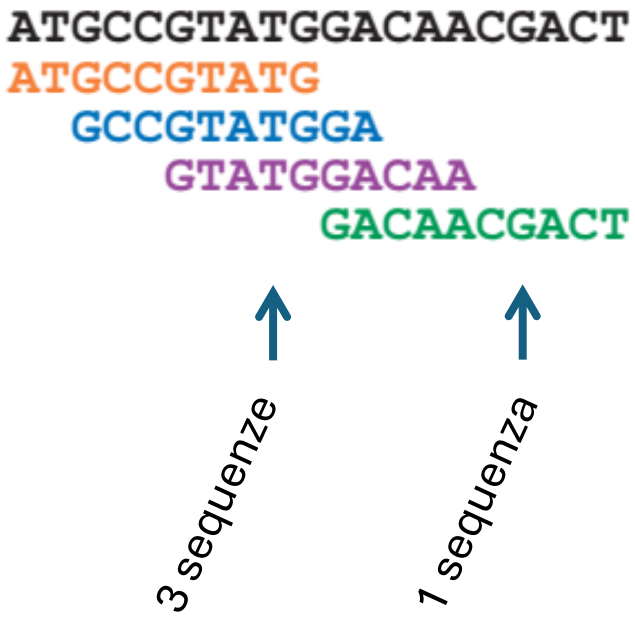


Figure S3.2: Coverage in bins of GC content according to a sliding window of 51bp length along the genome. Dashed lines show the average coverage over all bins and the 95% central part in this distribution. Legend: the number next to the line indicates the GC bin (5 indicates 5-9 G or C bases per 51bp window, 10 indicates 10-14, etc.); Percentages in brackets give the percentage of mappable bases that reside in each bin.

Esempio di coverage per dati reali (un genoma di Neanderthal) da Mafessoni et al., 2020, *PNAS*

Differenza 3: nei dati di sequenziamento la “coverage” non è uniforme, e le sequenze differiscono in lunghezza



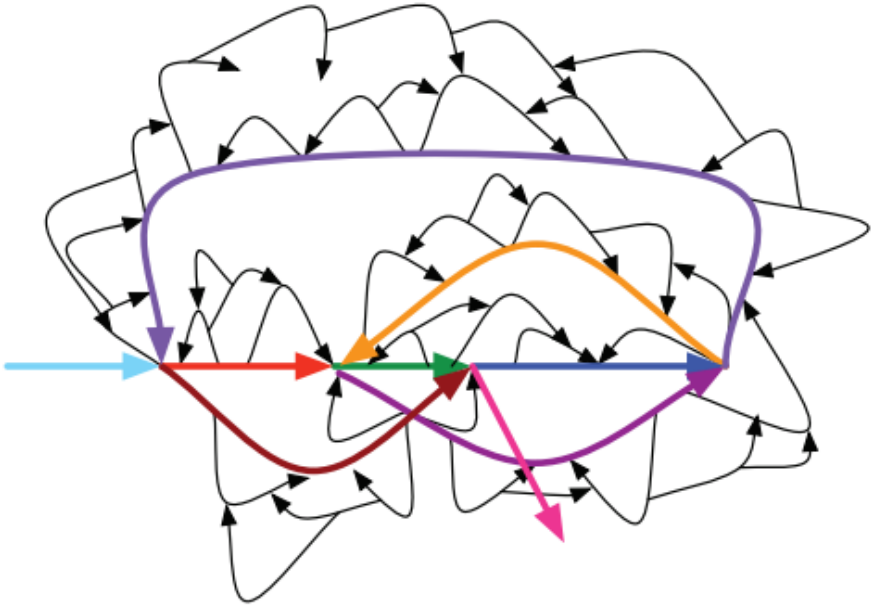
“Rompere” in k-mers le nostre sequenze porta ad una coperatura piu uniforme e a sequenze tutte di lunghezza uguale

Rompere in kmeri piu’ corti risolve (parzialmente)!!

Differenza 4: abbiamo errori nelle sequenze (o bolle)

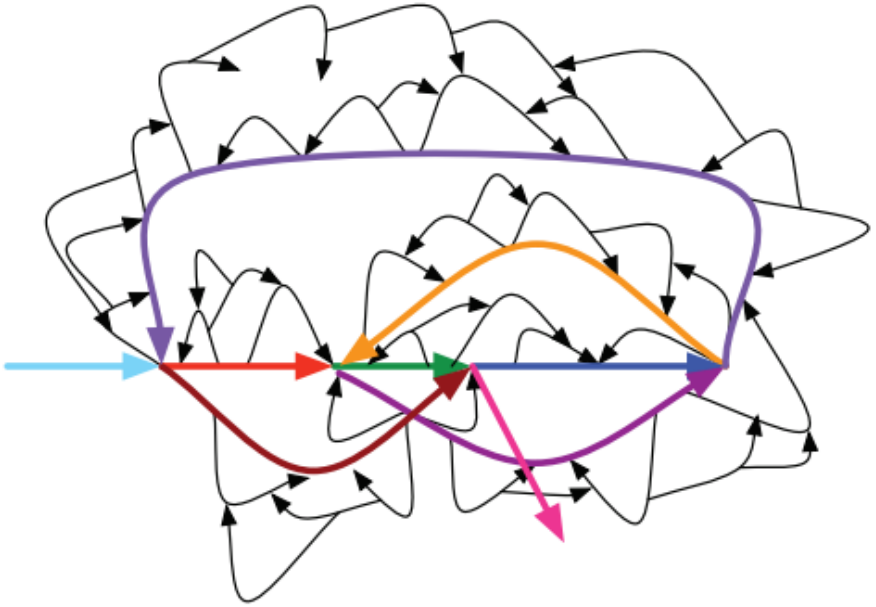
Una “bolla” causata da un errore di sequenziamento (una T letta come C)

Differenza 4: abbiamo errori nelle sequenze (o bolle)



In dati reali ci sono molti errori di sequenziamento (quindi molte bolle)!

Differenza 4: abbiamo errori nelle sequenze (o bolle)



In dati reali ci sono molti errori di sequenziamento (quindi molte bolle)!

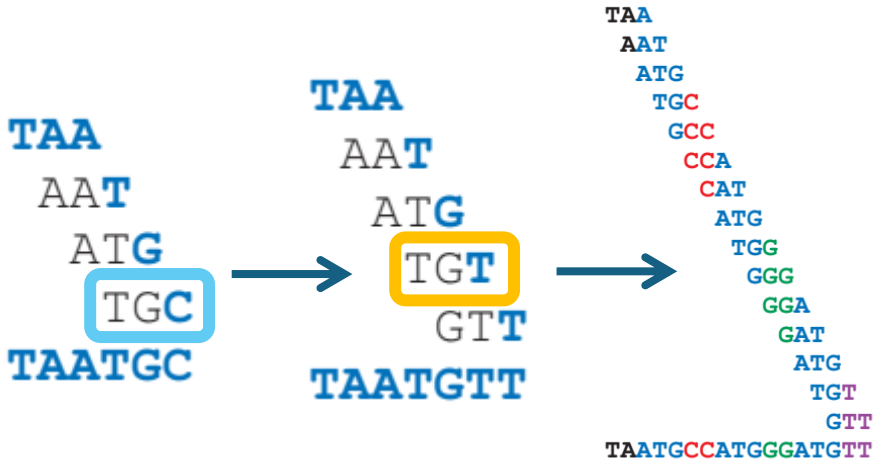
Assemblatori moderni correggono queste bolle sulla base di alcune “euristiche”:

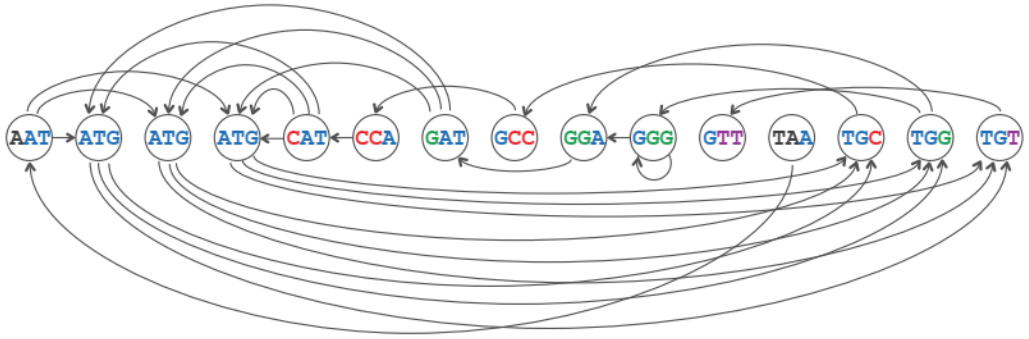
- rimuovere i **kmers rari*** prima della costruzione del grafo
- costruire il grafo e poi rimuovere i percorsi che creano bolle e sono supportati da **kmers rari**

*gli errori di sequenziamento avvengono perlopiu’ casualmente, quindi è abbastanza improbabile che molti errori avvengano nella stessa posizione. Pertanto generalmente gli errori originano kmer unici o molto rari

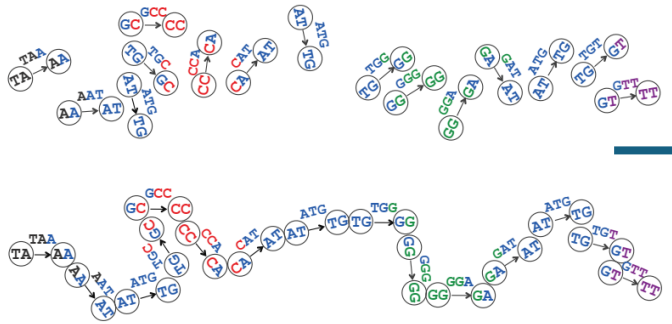
Due approcci per assemblare un genoma

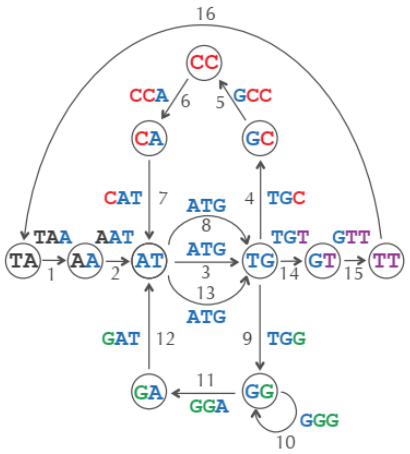
Esplorezione degli **Overlap-graph** per «forza bruta» ed usando al piu delle «euristiche»



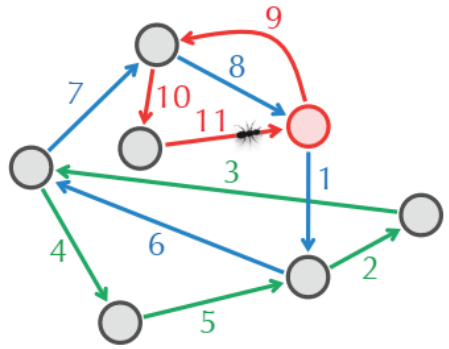


Costruzione del **De Brujin Graph**





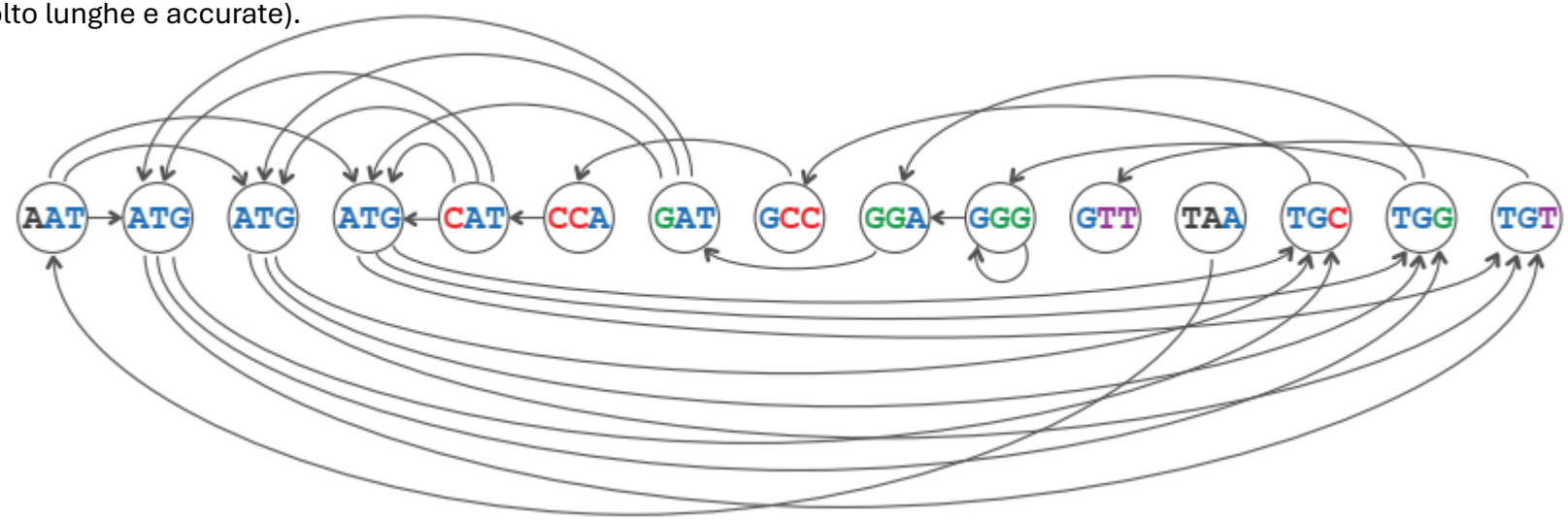
Ricerca del «ciclo Euleriano»*



*con l'algoritmo di Leo la formica

Gli «overlap graphs» sono alla base di vari assemblatori moderni

(specializzati in sequenze molto lunghe e accurate).



ARTICLES
https://doi.org/10.1038/s41592-020-01056-5
nature methods
Check for updates

Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm

Haoyu Cheng[1,2], Gregory T. Concepcion[3], Xiaowen Feng[1,2], Haowen Zhang[4] and Heng Li[1,2]

Method

HiCanu: accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads

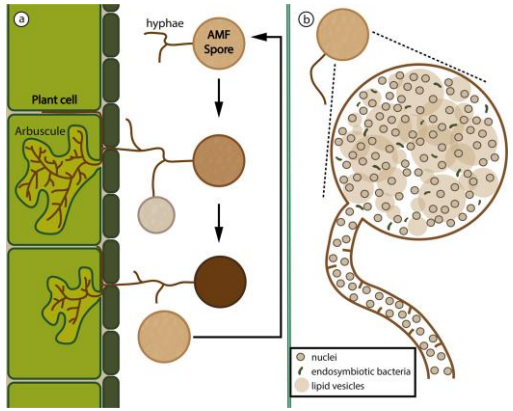
Sergey Nurk,[1,6] Brian P. Walenz,[1,6] Arang Rhie,[1] Mitchell R. Vollger,[2] Glennis A. Logsdon,[2] Robert Grothe,[3] Karen H. Miga,[4] Evan E. Eichler,[2,5] Adam M. Phillippy,[1] and Sergey Koren[1]

Assemblatori basati su De Brujin graphs possono usare anche short reads

SPAdes

Aimed at short reads, small genomes, metagenomes, single-cell data

Human Microbiome Project



Montoliu-Nerin et al.,2020

Verkko2 integrates proximity-ligation data with long-read De Bruijn graphs for efficient telomere-to-telomere genome assembly, phasing, and scaffolding

Dmitry Antipov,[1,4] Mikko Rautiainen,[2,4] Sergey Nurk,[3] Brian P. Walenz,[1] Steven J. Solar,[1] Adam M. Phillippy,[1] and Sergey Koren[1]

High-quality metagenome assembly from long accurate reads with metaMDBG

Nature Biotechnology **42**, 1378–1383 (2024)

Gaëtan Benoit, Sébastien Raguideau, Robert James, Adam M. Phillippy, Rayan Chikhi & Christopher Quince

LJA

Multiplex de Bruijn graphs enable genome assembly from long, high-fidelity reads

Anton Bankevich, Andrey V. Bzikadze, Mikhail Kolmogorov, Dmitry Antipov & Pavel A. Pevzner

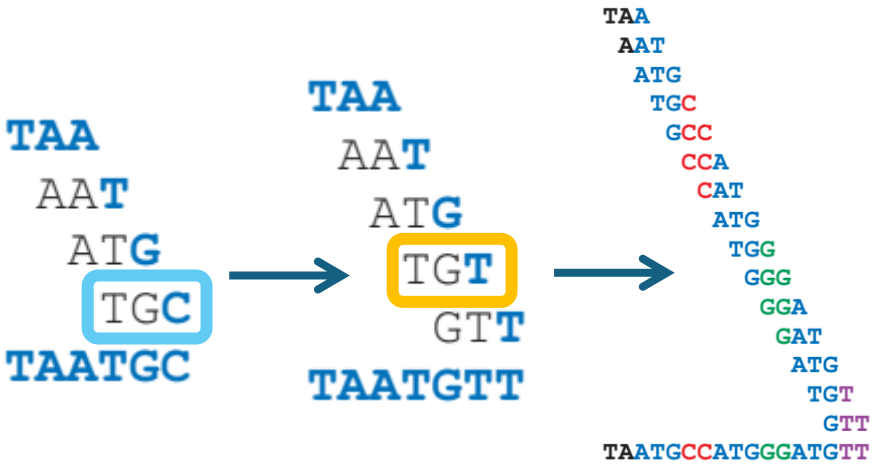
Nature Biotechnology **40**, 1075–1081 (2022) | Cite this article

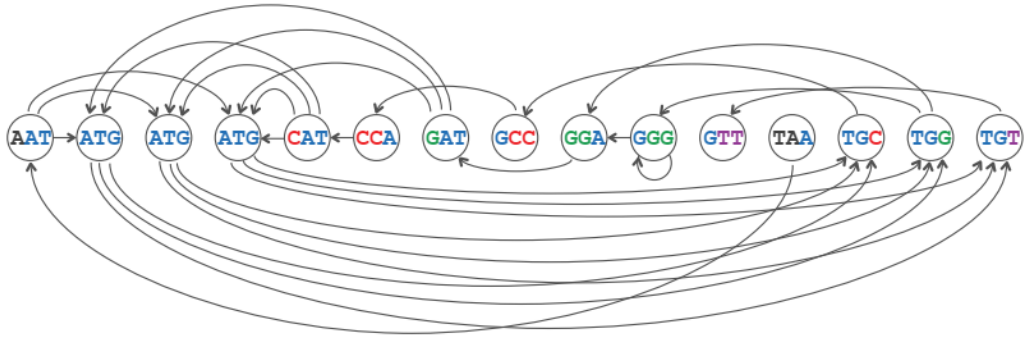
Fast and feasible for large repeated genomes like barley

Due approcci per assemblare un genoma

Applicazioni

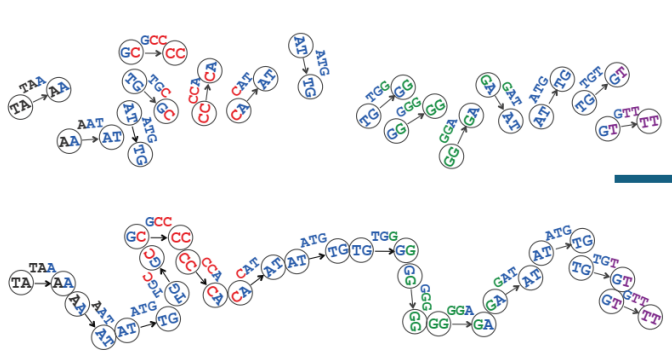
Esplorezione degli **Overlap-graph** per «forza bruta» ed usando al piu delle «euristiche»

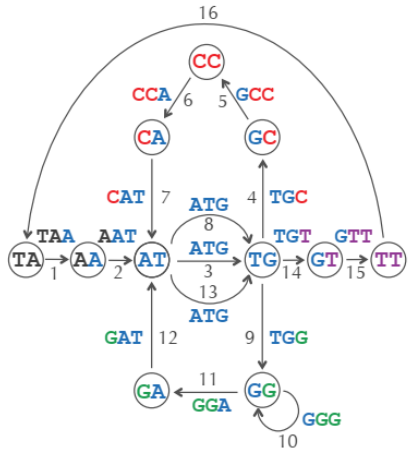




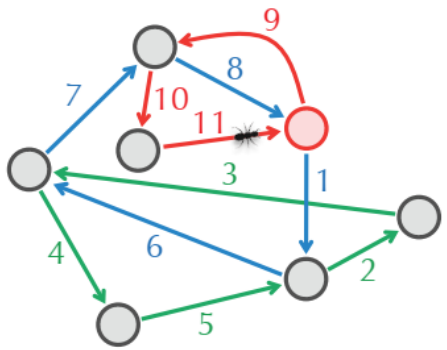
Applicabili solo a sequenze ultralunghe e accurate, ad esempio per genomi ultracompleti (Telomere 2 Telomere).

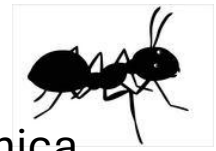
Costruzione del **De Brujin Graph**





Ricerca del «ciclo Euleriano»*





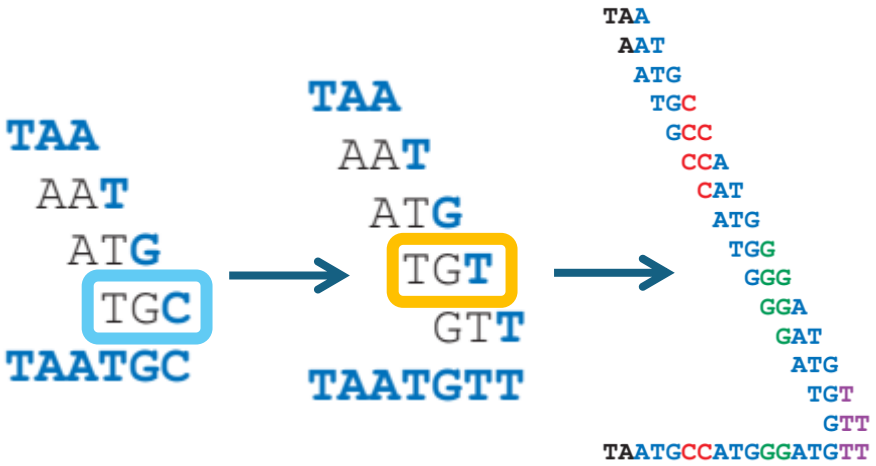
*con l'algoritmo di Leo la formica

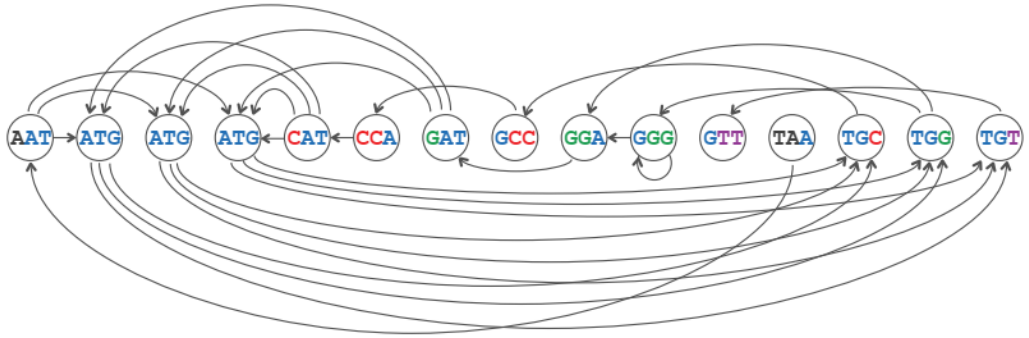
Metagenomi, genomi microbici, dati vari incluse sequenze corte (illumina), ma anche phasing e genomi complessi

Due approcci per assemblare un genoma

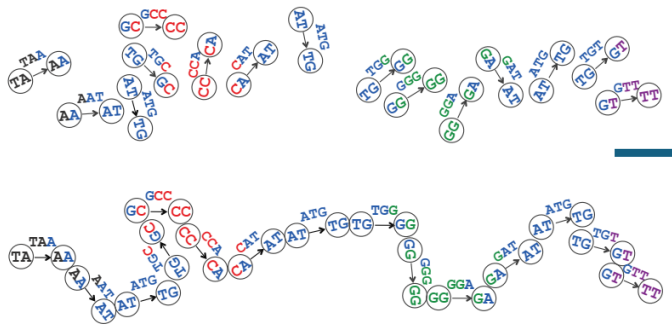
Applicazioni

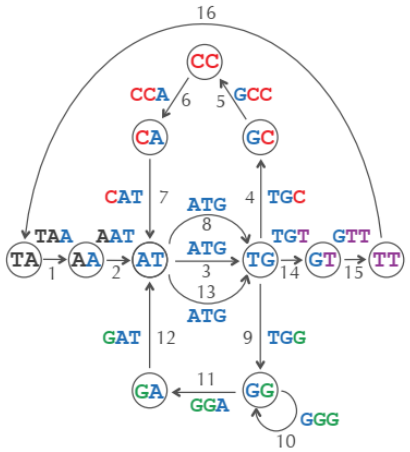
Esplorezione degli **Overlap-graph** per «forza bruta» ed usando al piu delle «euristiche»



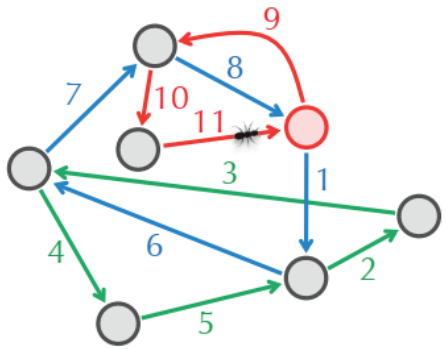


Costruzione del **De Brujin Graph**





Ricerca del «ciclo Euleriano»*

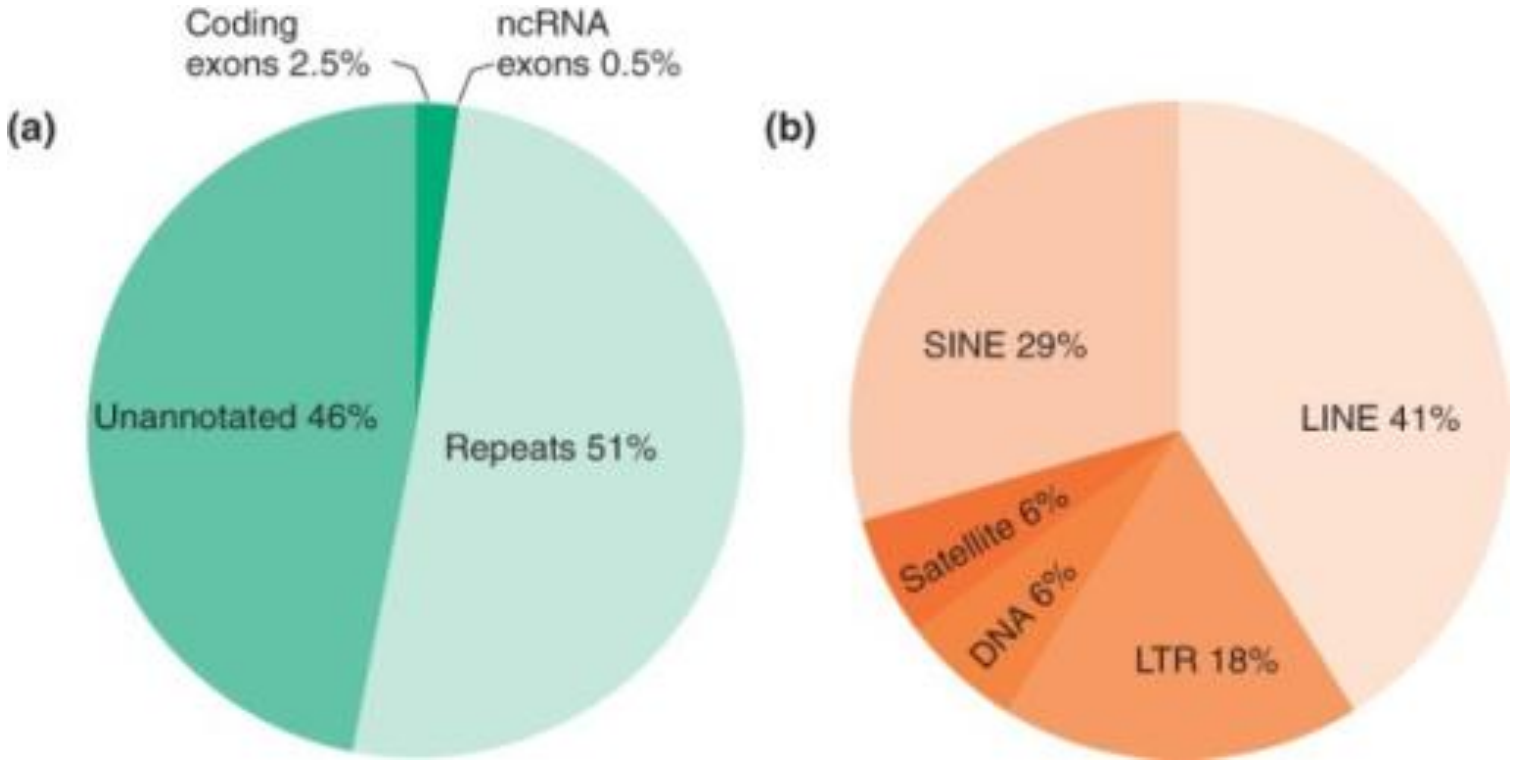


*con l’algoritmo di Leo la formica

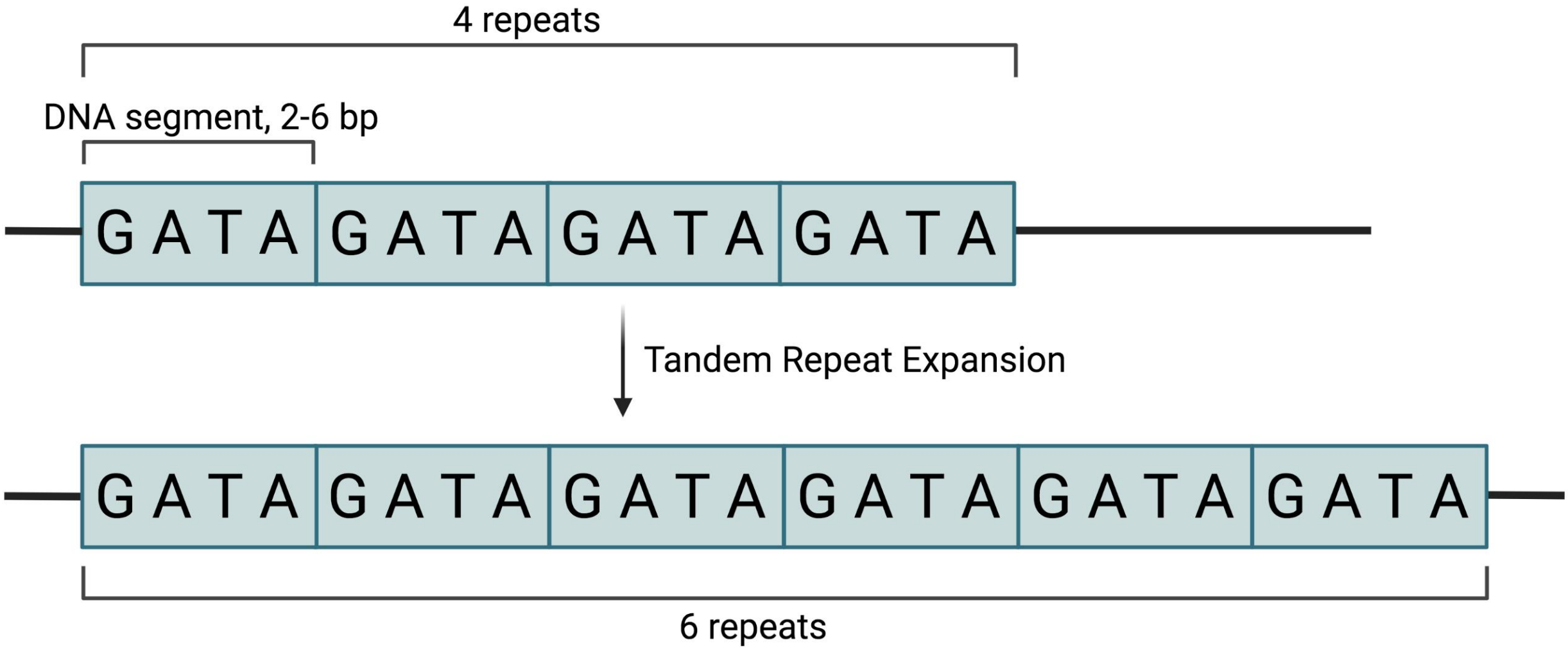
Come mai i De Brujin Graphs non funzionino meglio in ogni circostanza?

- Anche se il problema diventa piu' trattabile dobbiamo necessariamente rompere in k-mers
- Per sequenze ripetute i k-mers possono introdurre molte ambiguità, risolte solamente da tecnologie in grado di dare informazione a lungo raggio o sequenze molto lunghe

Il genoma (umano) è altamente ripetitivo



Short Tandem Repeat (STR) o *microsatelliti*



I problemi introdotti dalle ripetizioni

- Le zone ripetute sono ricostruite erroneamente

CACGTATATATATATATATATA**TTTGGCAT**

ricostruito come

CACGTATATA**TTTGGCAT**

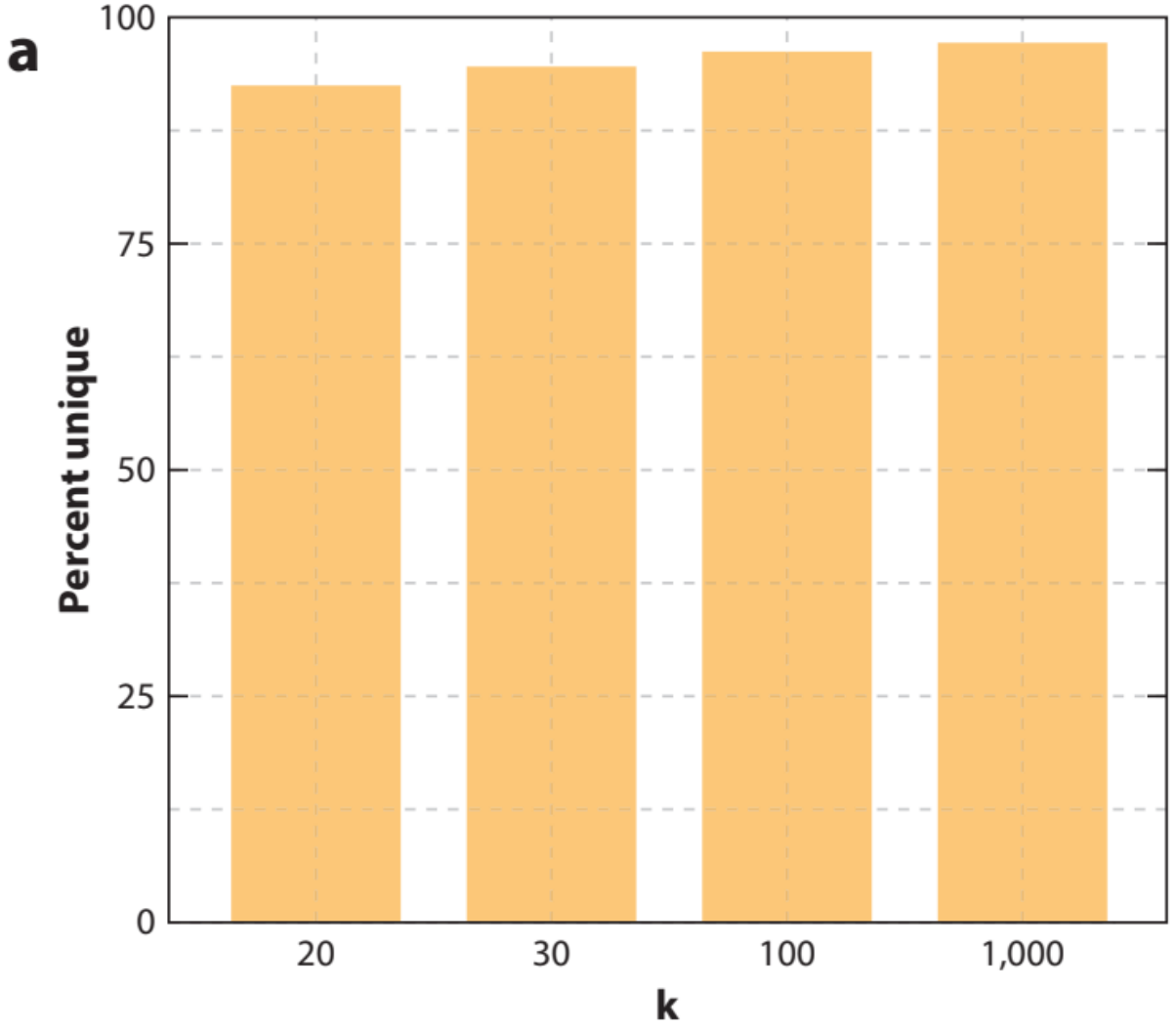
oppure

CACGTATATATATATATATATATATATATATATATA**TTTGGCAT**

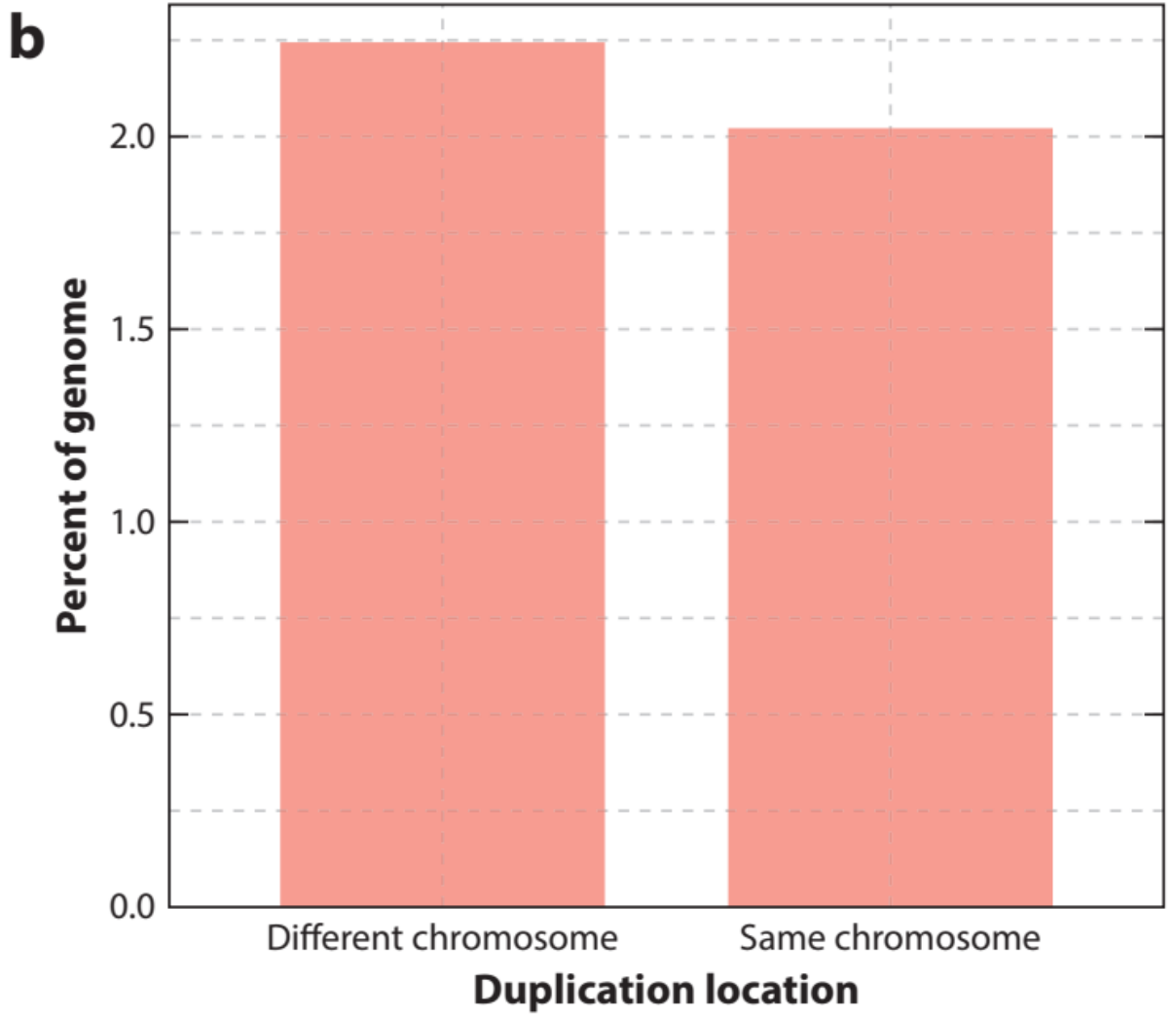
- Piu’ in generale, diventa impossibile ricostruire un genoma in modo ragionevole a causa dei molti «percorsi» errati nell’overlap graph

ricostruito come

Tuttavia alcune regioni ripetute sono >10kb e possono essere risolte solo con sequenze lunghissime e accurate (o con lo scaffolding)

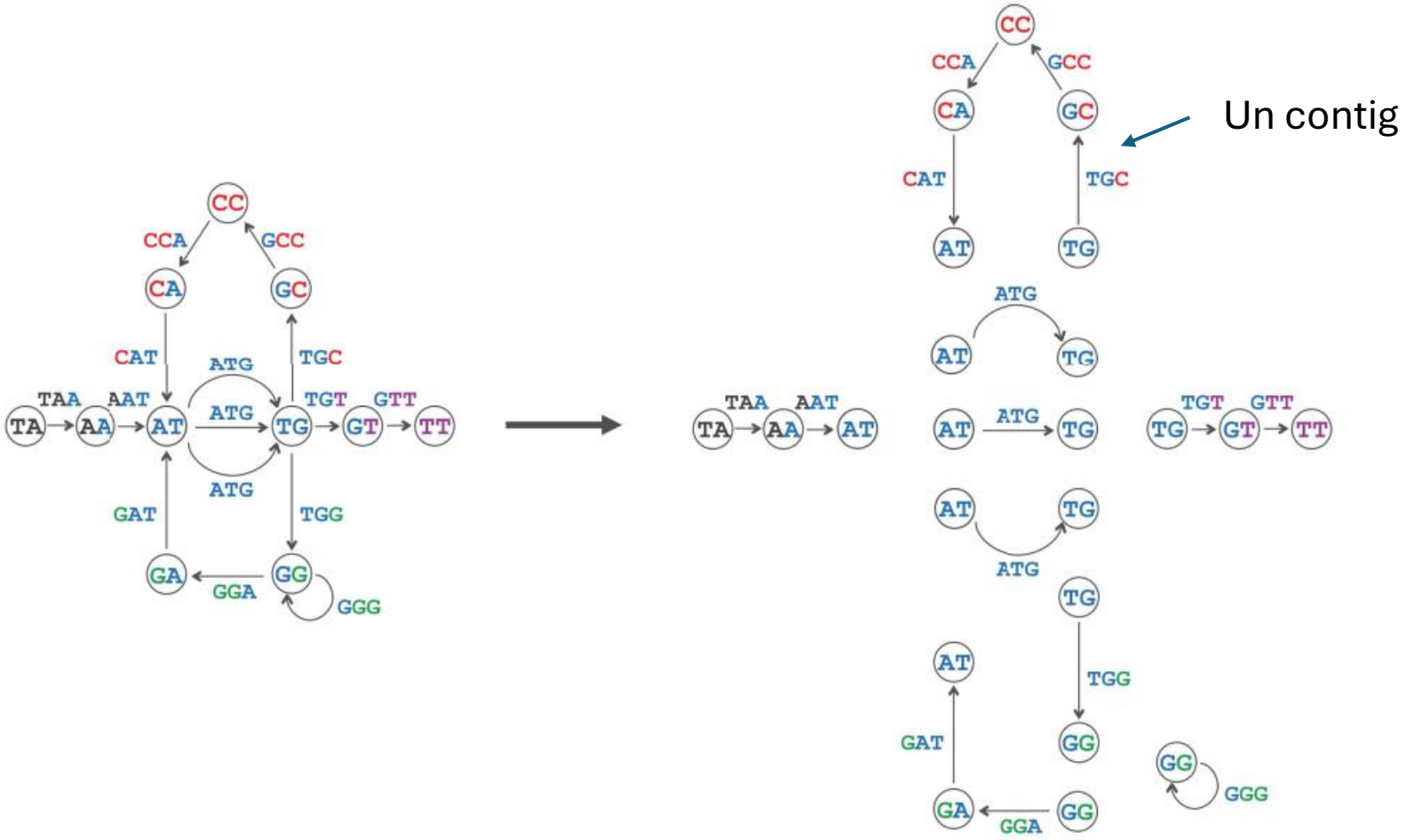


Percentuale di genoma unico per k-mers di varie lunghezze

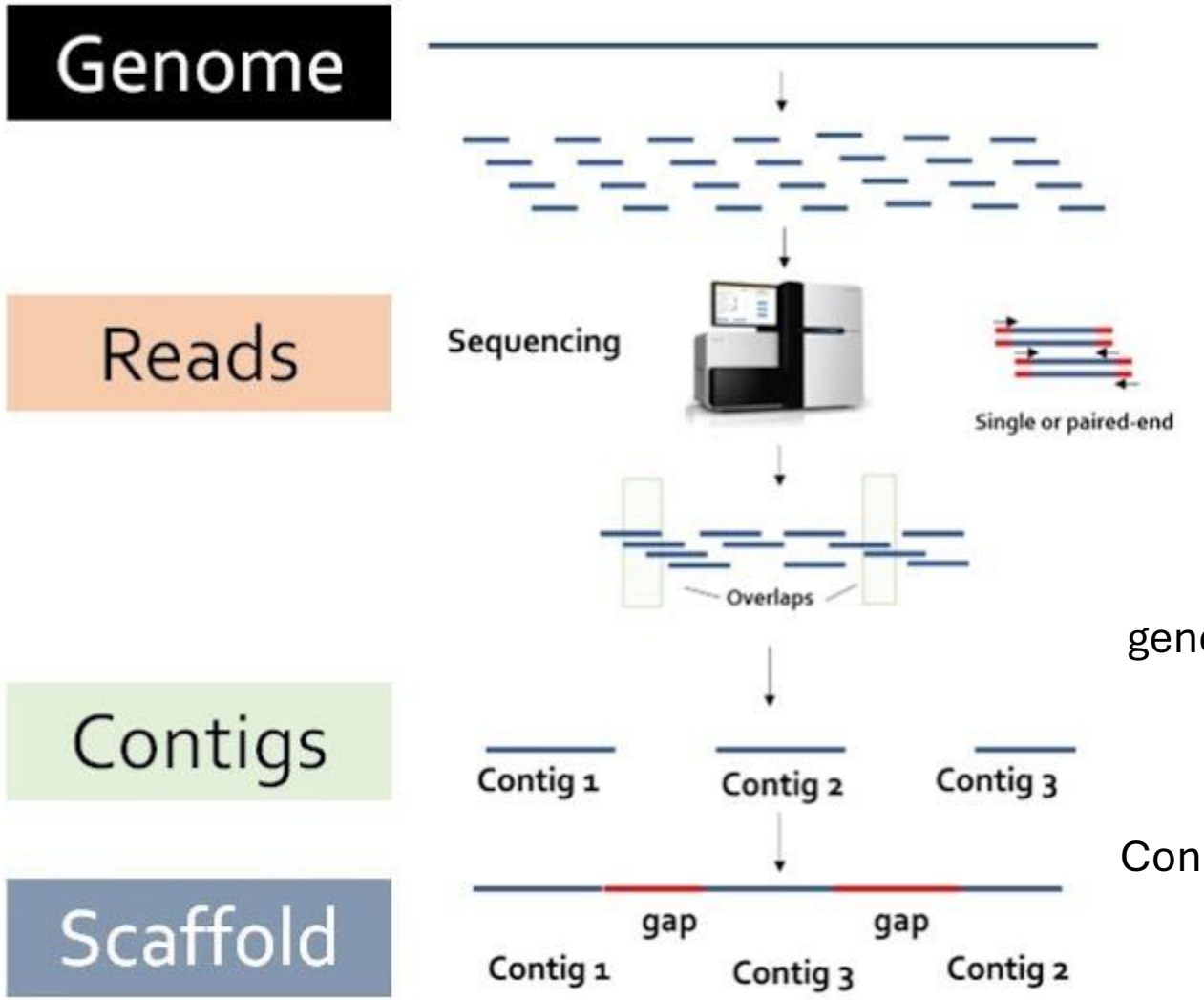


Percentuale di genoma duplicato >5kb con similarità >95%

In pratica, gli assemblaggi vengono ridotti a contigs non ambigui



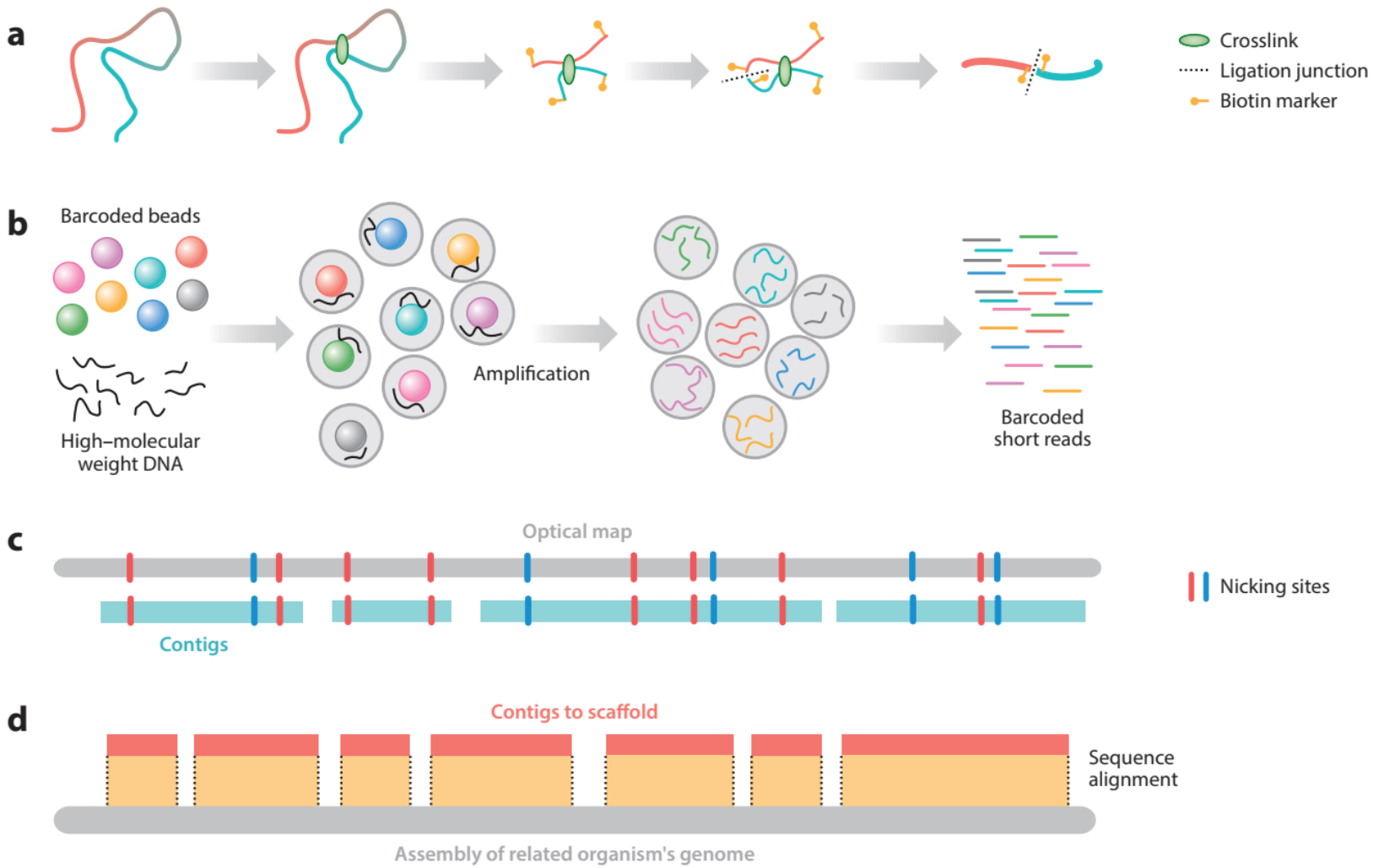
Reads, contigs and scaffolds (in genomic context)



Assemblaggio
generalmente con grafi di de Brujin

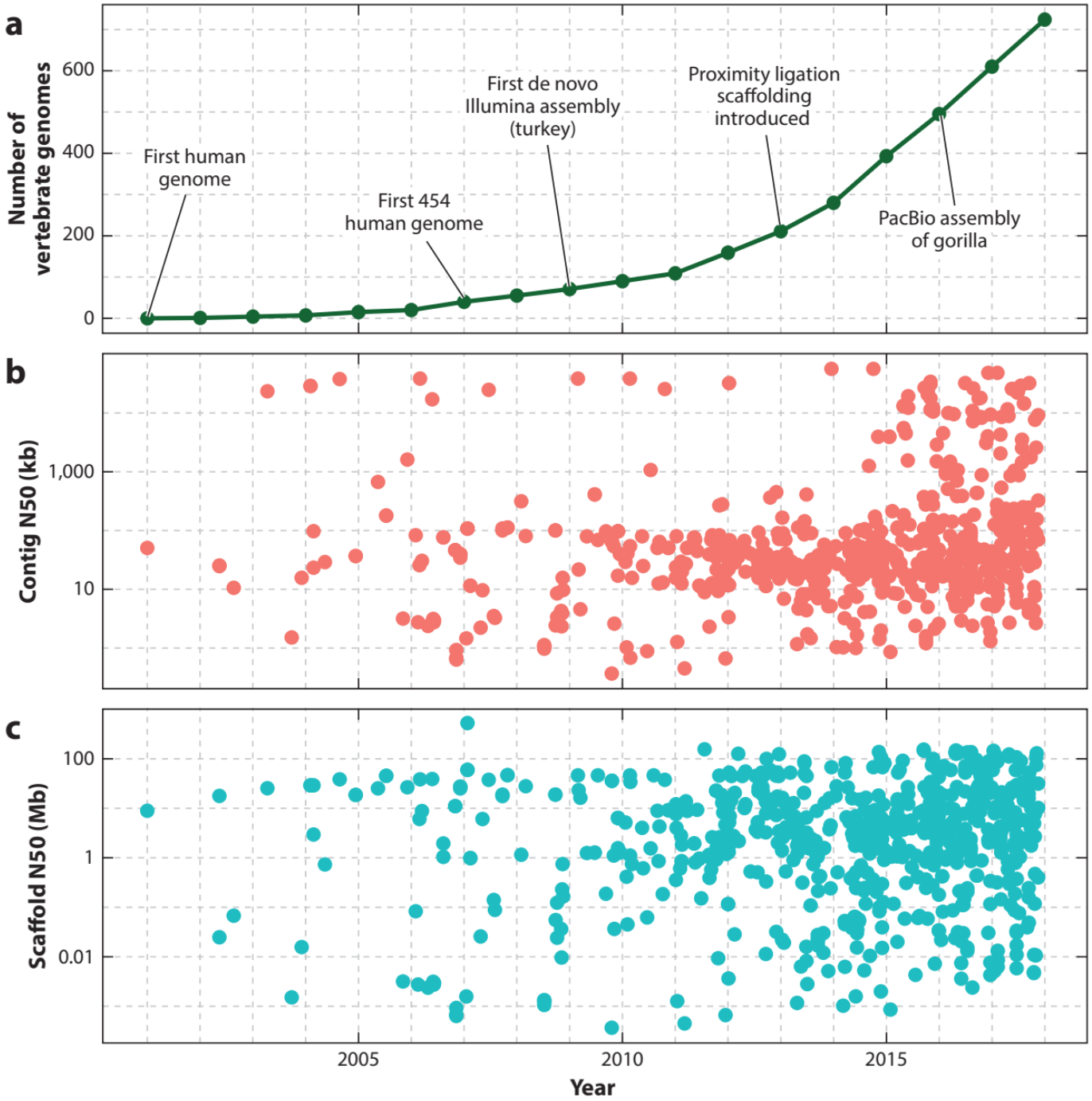
Scaffolding
Con informazione a lungo raggio

Lo scaffolding permette di «unire» piu' contigs



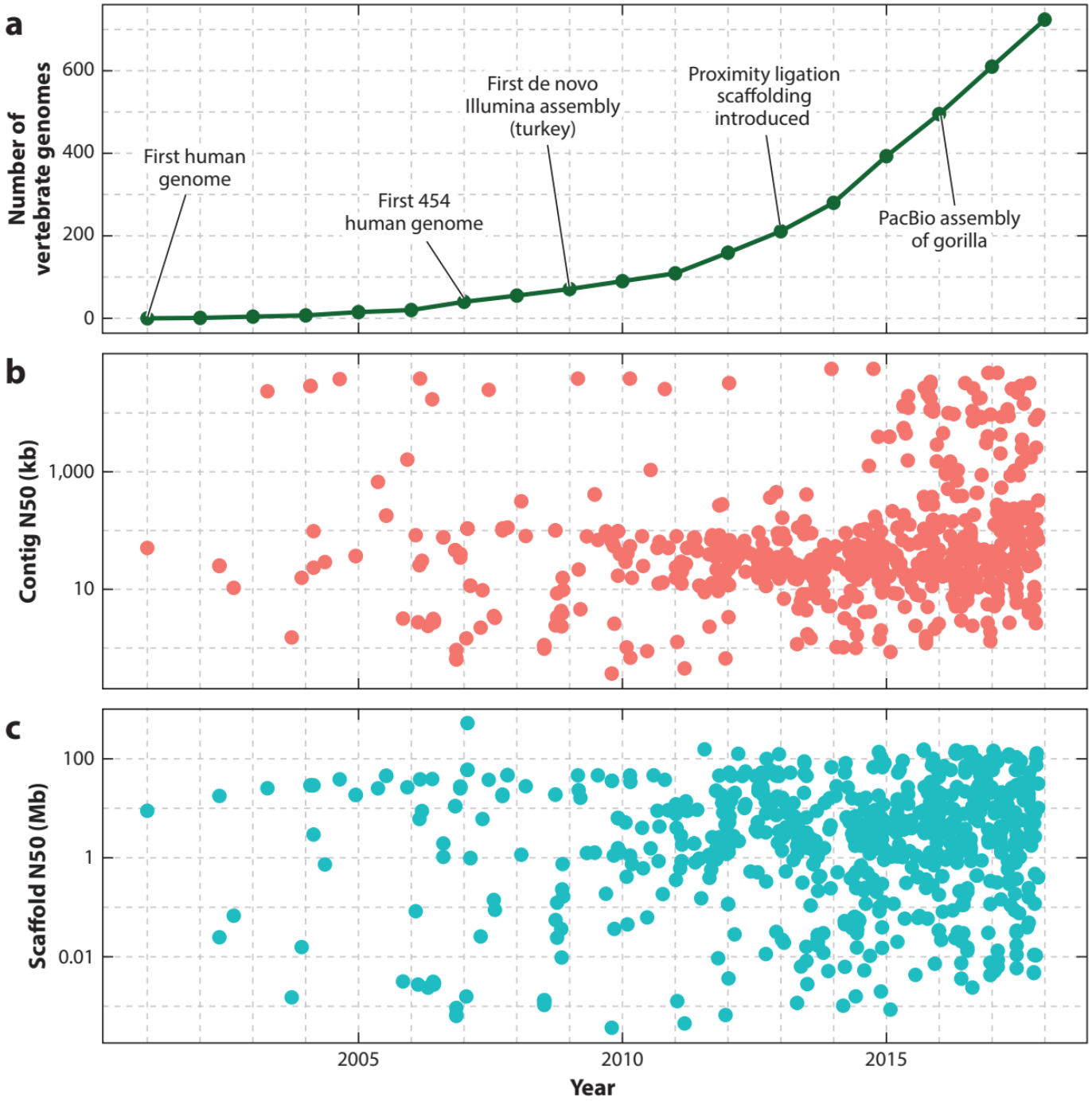
- Proximity ligation (Hi-C)
- Linked-read sequencing (10x), amplificazione delle sequenze dentro droplets
- Enzimi che attaccano marcatori fluorescent (BioNano)
- Approcci basati su sintenia che usano altri genomi come referenza

Approfondirete tutte queste tecniche in Genomica Applicata col Prof. Marco Gerdol



Contiguità dei genomi

L’arrivo di alcune tecnologie (Hi-C) ha portato a miglioramenti nella contiguità, ma il grosso dei genomi è stato finora limitato (dalle lunghe sequenze ripetute).



Contiguità dei genomi

L’arrivo di alcune tecnologie (Hi-C) ha portato a miglioramenti nella contiguità, ma il grosso dei genomi è stato finora limitato (dalle lunghe sequenze ripetute).

Solo dal 2019 circa le sequenze accurate ultralunghe hanno permesso i primi genomi T2T con overlap-graphs

Misurare la contiguità di un assemblaggio: N50

Definizione: la lunghezza del contig piu piccolo nel piu' piccolo insieme di contig che contiene il 50% del genoma

O

Cercando di selezionare solo i contigs piu' lunghi, il valore del contig piu' piccolo con il quale riesco ancora a rappresentare almeno il 50% del genoma. Il numero di quei contigs viene definito **L50**

Esempio con 1Mb:

N50 size = 30 kbp
(300k+100k+45k+45k+30k >= 500kbp)

Alto N50, basso N50: pochi contigs molto lunghi

Basso N50, Alto L50: molti contigs corti

Misurare la contiguità di un assemblaggio: N50

Definizione: la lunghezza del contig piu piccolo nel piu' piccolo insieme di contig che contiene il 50% del genoma

O

Cercando di selezionare solo i contigs piu' lunghi, il valore del contig piu' piccolo con il quale riesco ancora a rappresentare almeno il 50% del genoma. Il numero di quei contigs viene definito **L50**

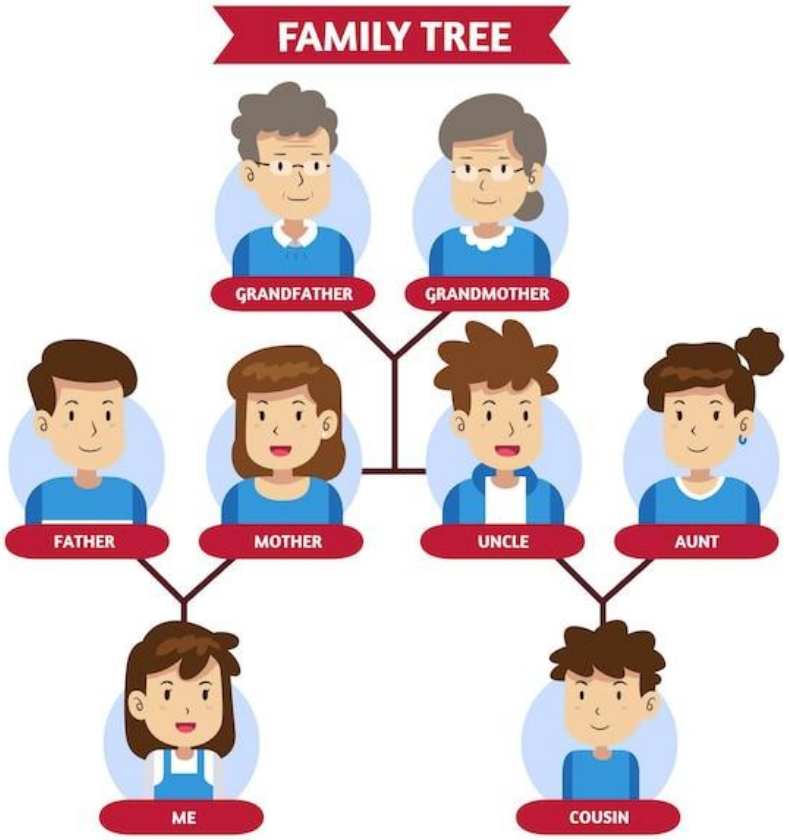
Esempio con 1Mb:

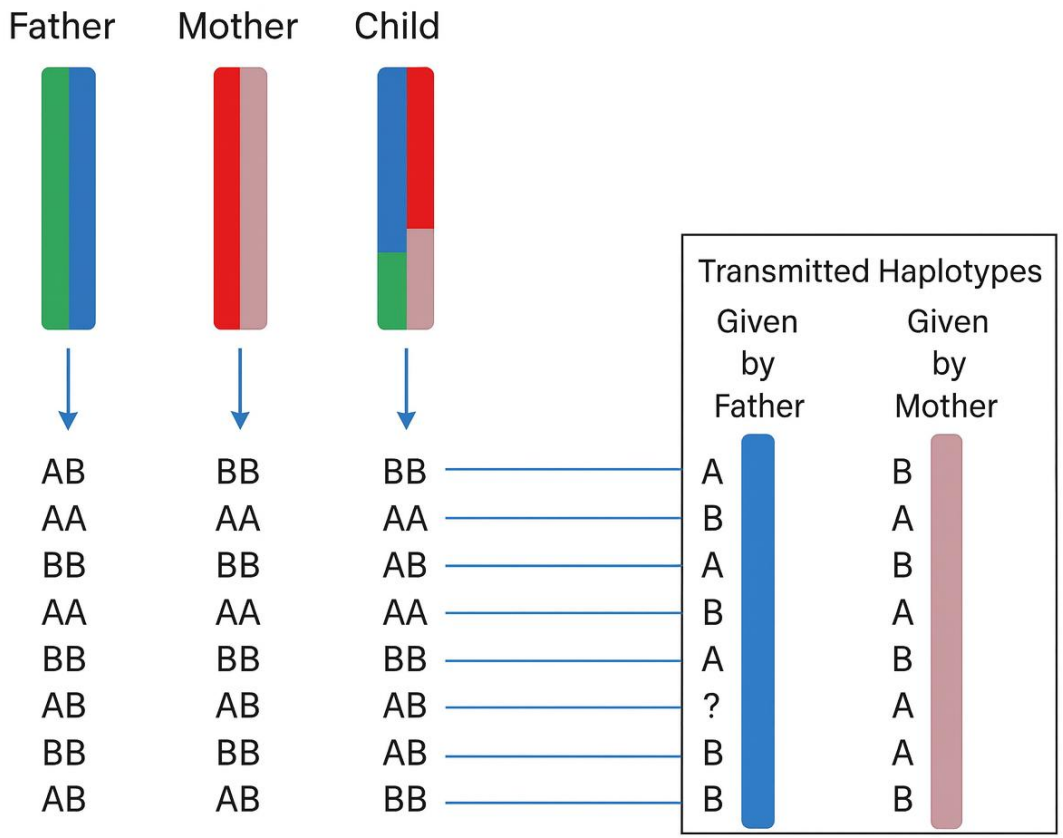
Alto N50, basso N50: pochi contigs molto lunghi

Basso N50, Alto L50: molti contigs corti

Un tipo di ripetizioni «quasi» inevitabile: i genomi diploidi!!

- Il nostro intero genoma è ripetuto almeno una volta





Un tipo di ripetizioni «quasi» inevitabile: i genomi diploidi!!

- Una prima semplice soluzione sperimentale: non usare diploidi! Usare individui «inbred» od omozigoti

Generazione 1

Generazione 2

Generazione 3

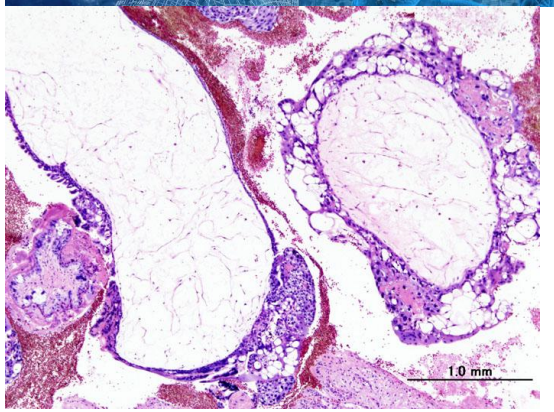
Un tipo di ripetizioni «quasi» inevitabile: i genomi diploidi!!

- Una prima semplice soluzione sperimentale: non usare diploidi! Usare individui «inbred» od omozigoti

Generazione 1

Generazione 2

Generazione 3

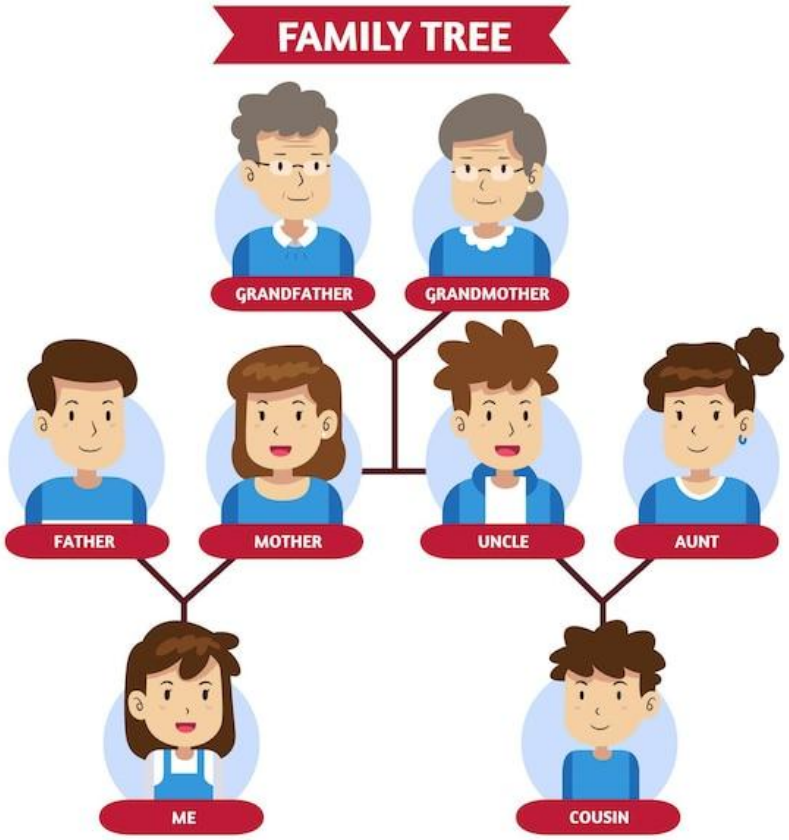


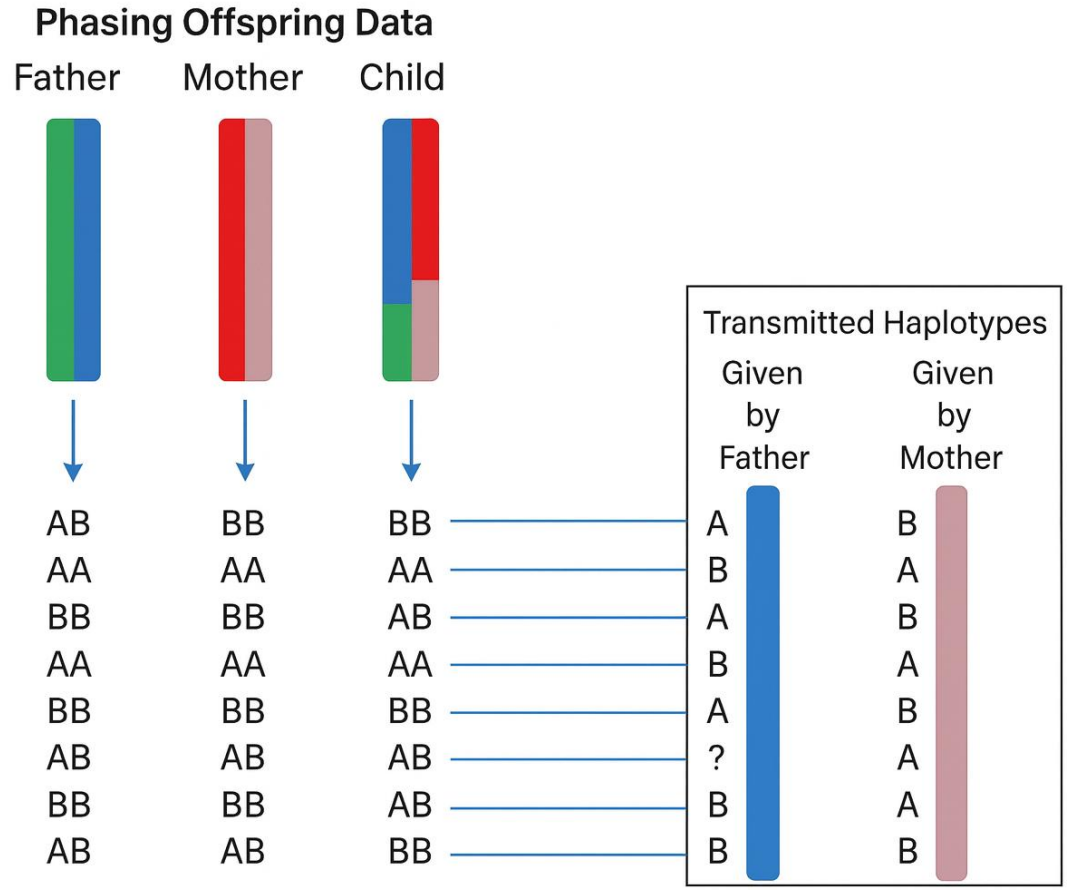
Ottenuto da una **mole idatiforme* interamente omozigote e con cariotipo 46XX.**

*un embrione anomalo che cresce come massa nell'utero in cui spesso uno dei due «genomi» parentali viene espulso, e che non sviluppa in un feto

Un tipo di ripetizioni «quasi» inevitabile: i genomi diploidi!!

- Una seconda strategia: utilizzo di alberi genealogici (o trio). Sequenziare anche i genitori per inferire la “fase” dei cromosomi!





Un tipo di ripetizioni «quasi» inevitabile: i genomi diploidi!!

- Un’altra strategia molto usata: ignorare il problema.

Le recenti tecnologie di sequenziamento ultra-lungo permettono oggi di affrontare il problema (siamo agli inizi)

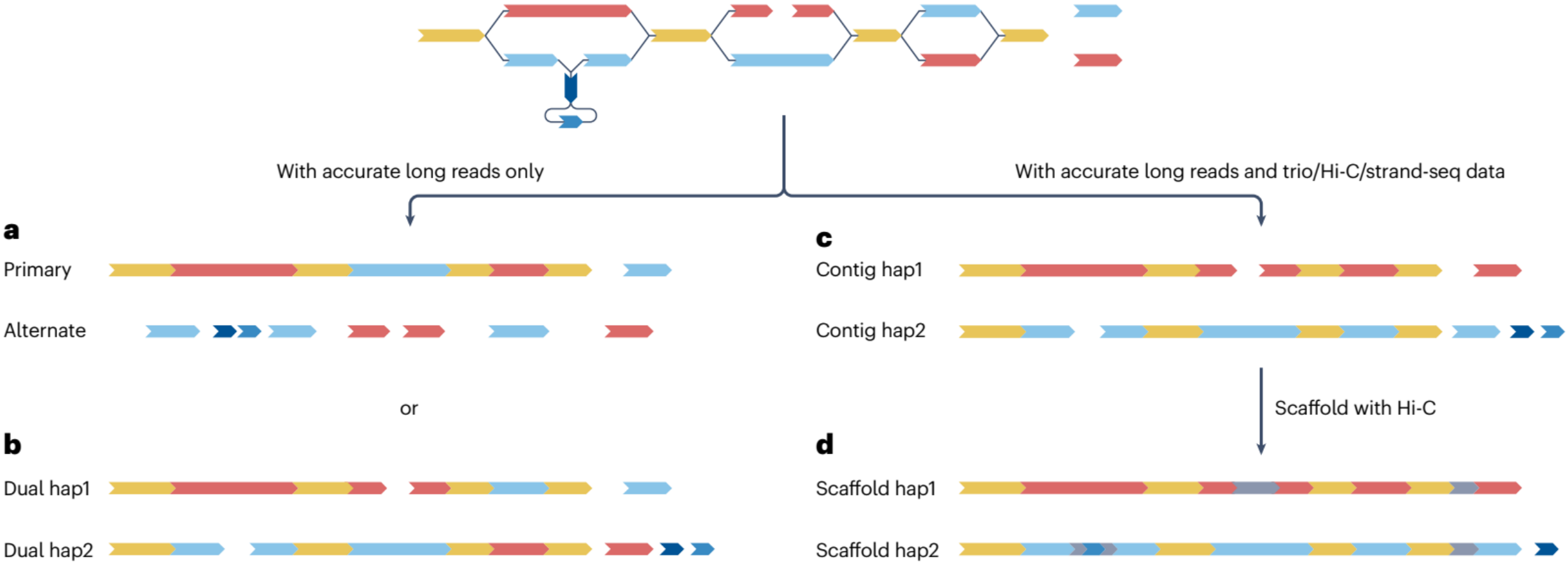


Fig. 2 | Types of phased assembly of diploid samples. The initial assembly graph can be further processed into different types of assemblies. If only accurate long-read sequence data are available, a primary–alternate assembly pair or a dual assembly pair can be generated. If long-range data (such as trio, Hi-C or strand-seq data) are available, chromosome-phased assemblies can be generated. **a**, In a primary–alternate assembly pair, the primary assembly represents a complete haploid genome with occasional phase switches whereas the alternate assembly is fragmented and contains no homozygous regions longer than the read length. **b**, In a pair of dual assemblies, each assembly is similar to a primary assembly. **c**, In a pair of chromosome-phased assemblies, long-range data are used to partition contigs from the same haploid chromosome to the same assembly. **d**, In a pair of chromosome-phased assemblies with scaffolding, Hi-C data are used to join contigs into chromosomes across assembly gaps (marked by thick grey arrows). hap, haplotype; trio data, sequencing data on an individual and both its parents.

Li e Durbin, 2024

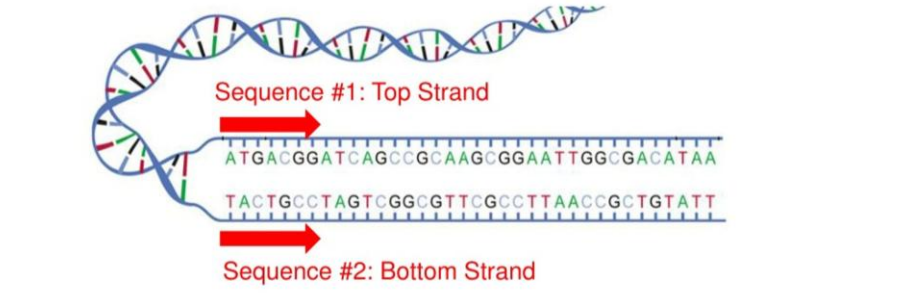
Un tipo di «ripetizione» totalmente inevitabile

Il DNA è a doppio filamento!!!

Di conseguenza per ogni filamento GATGTCATA avremo anche TATGACATC

Per risolvere questo problema, vengono usate euristiche:

1) Creazione per ogni sequenza del suo reverse complement
2) Risoluzione del «doppio grafo» a posteriori

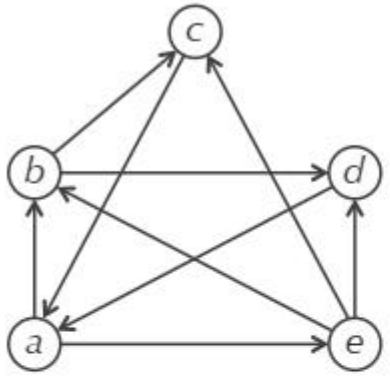


Sequence #1: Top ("F") 5′ – A T G A C G G A T C A G C – 3′

Sequence #2: Bottom ("R") 3′ – T A C T G C C T A G T C G – 5′

I grafi si possono rappresentare come matrici o liste

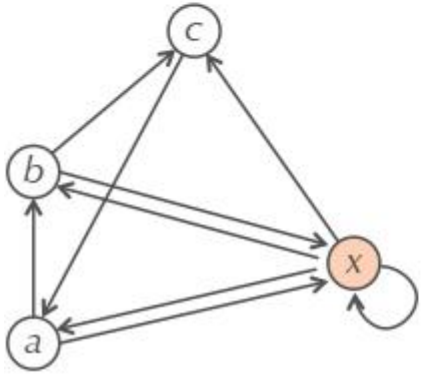
Graph



Adjacency List

a is adjacent to *b* and *e*
b is adjacent to *c* and *d*
c is adjacent to *a*
d is adjacent to *a*
e is adjacent to *b*, *c*, and *d*

Graph

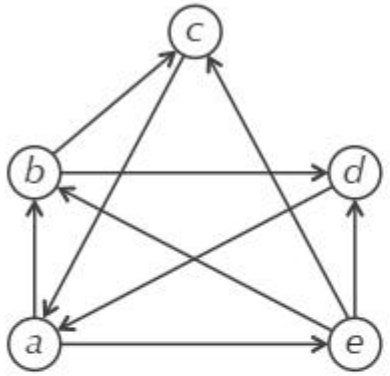


Adjacency List

a is adjacent to *b* and *x*
b is adjacent to *c* and *x*
c is adjacent to *a*
x is adjacent to *a*, *b*, *c*, and *x*

I grafi si possono rappresentare come matrici o liste

Graph



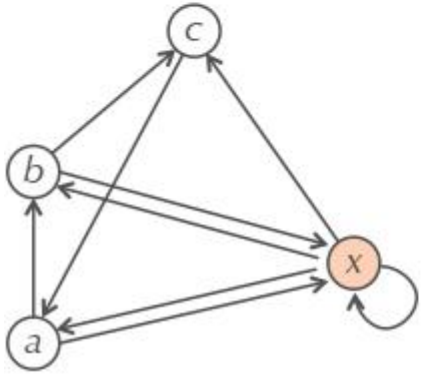
Adjacency Matrix

	a	*b*	*c*	*d*	*e*
a	0	1	0	0	1
b	0	0	1	1	0
c	1	0	0	0	0
d	1	0	0	0	0
e	0	1	1	1	0

Adjacency List

a is adjacent to *b* and *e*
b is adjacent to *c* and *d*
c is adjacent to *a*
d is adjacent to *a*
e is adjacent to *b*, *c*, and *d*

Graph



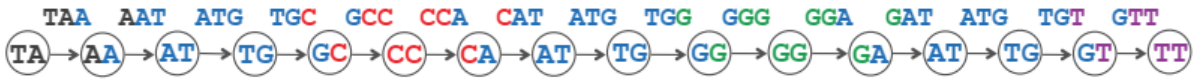
Adjacency Matrix

	a	*b*	*c*	*x*
a	0	1	0	1
b	0	0	1	1
c	1	0	0	0
x	1	1	1	1

Adjacency List

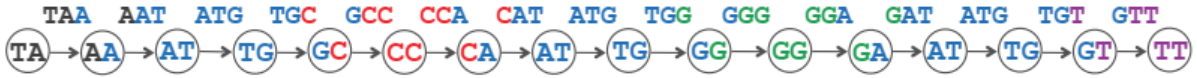
a is adjacent to *b* and *x*
b is adjacent to *c* and *x*
c is adjacent to *a*
x is adjacent to *a*, *b*, *c*, and *x*

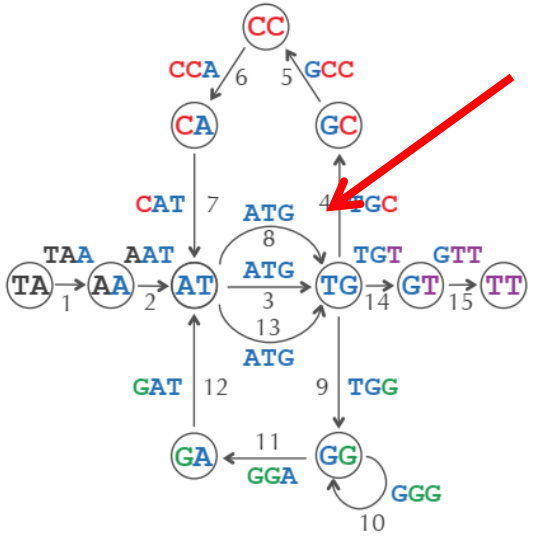
Le rappresentazioni matriciali possono anche rappresentare operazioni come «l'incollamento» dei grafi di De Brujin



	TA	AA	AT_1	TG_1	GC	CC	CA	AT_2	TG_2	GG_1	GG_2	GA	AT_3	TG_3	GT	TT
TA	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AA	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
AT_1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
TG_1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
GC	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
CC	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
CA	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
AT_2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
TG_2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
GG_1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
GG_2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
GA	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
AT_3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
TG_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
GT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
TT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Le rappresentazioni matriciali possono anche rappresentare operazioni come «l'incollamento» dei grafi di De Bruijn





	TA	AA	AT_1	TG_1	GC	CC	CA	AT_2	TG_2	GG_1	GG_2	GA	AT_3	TG_3	GT	TT
TA	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AA	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
AT_1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
TG_1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
GC	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
CC	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
CA	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
AT_2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
TG_2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
GG_1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
GG_2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
GA	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
AT_3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
TG_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
GT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
TT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	TA	AA	AT	TG	GC	CC	CA	GG	GA	GT	TT
TA	0	1	0	0	0	0	0	0	0	0	0
AA	0	0	1	0	0	0	0	0	0	0	0
AT	0	0	0	3	0	0	0	0	0	0	0
TG	0	0	0	0	1	0	0	1	0	1	0
GC	0	0	0	0	0	1	0	0	0	0	0
CC	0	0	0	0	0	0	1	0	0	0	0
CA	0	0	1	0	0	0	0	0	0	0	0
GG	0	0	0	0	0	0	0	1	1	0	0
GA	0	0	1	0	0	0	0	0	0	0	0
GT	0	0	0	0	0	0	0	0	0	0	1
TT	0	0	0	0	0	0	0	0	0	0	0

Cosa abbiamo visto oggi

- Perché è difficile assemblare
- Algoritmi per assemblare:
  - Overlap-graphs
  - De Brujin graphs
- Concetti generali delle scienze computazionali (“Teoria dei grafi”, algoritmi polinomiali ed intrattabili)
- Applicazioni specifiche a tipologie di dati di sequenziamento e problemi dei “genomi veri”
  - Il problema delle ripetizioni
  - Scaffolding
  - Assemblare con sequenze lunghe e corte

La prossima lezione

- Pratica di assemblaggio di gneomi con strumenti bioinformatici