



Università degli
studi di Salerno

ASSEMBLAGGIO DI GENOMI TRAMITE DE BRUIJN GRAPH:

Studio del sequenziamento,
Bloom Filter e analisi
dell'assemblatore Minia

Studenti
Vittorio Guida
Costantino Paciello

Professori
Rosalba Zizza
Rocco Zaccagnino
Clelia De Felice



Memoria Limitata



→ PROBLEMA

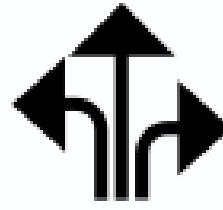
- **Genoma umano** = 3 miliardi bp → con un assembler tradizionale servono 300+ GB → Impartibile!

→ COME RAPPRESENTARE IL GRAFO IMPLICITAMENTE MANTENENDO CORRETTEZZA?

- **Soluzione Minia** = 5.7 GB (50x riduzione!)

Architettura Minia

- BLOOM FILTER
11 bit/k-mer - Rappresenta nodi k-mer
- STRUTTURA CFP
1.86 bit/k-mer - Corregge falsi positivi
- MARKING STRUCTURE
4.42 bit/k-mer - Traccia nodi visitati



Totale: 13.2 bit/k-mer
(genoma umano)

Self-information vs

Minia

Lower Bound:

- Teoria dell'informazione: servono 24 bit / k-mer per codificare esattamente un insieme di k-mer (genoma umano).

Il paradosso di Minia:

- Minia utilizza solo 13.2 bit/k-mer.
- **Come è possibile?** Non memorizza l'insieme esatto dei k-mer in memoria

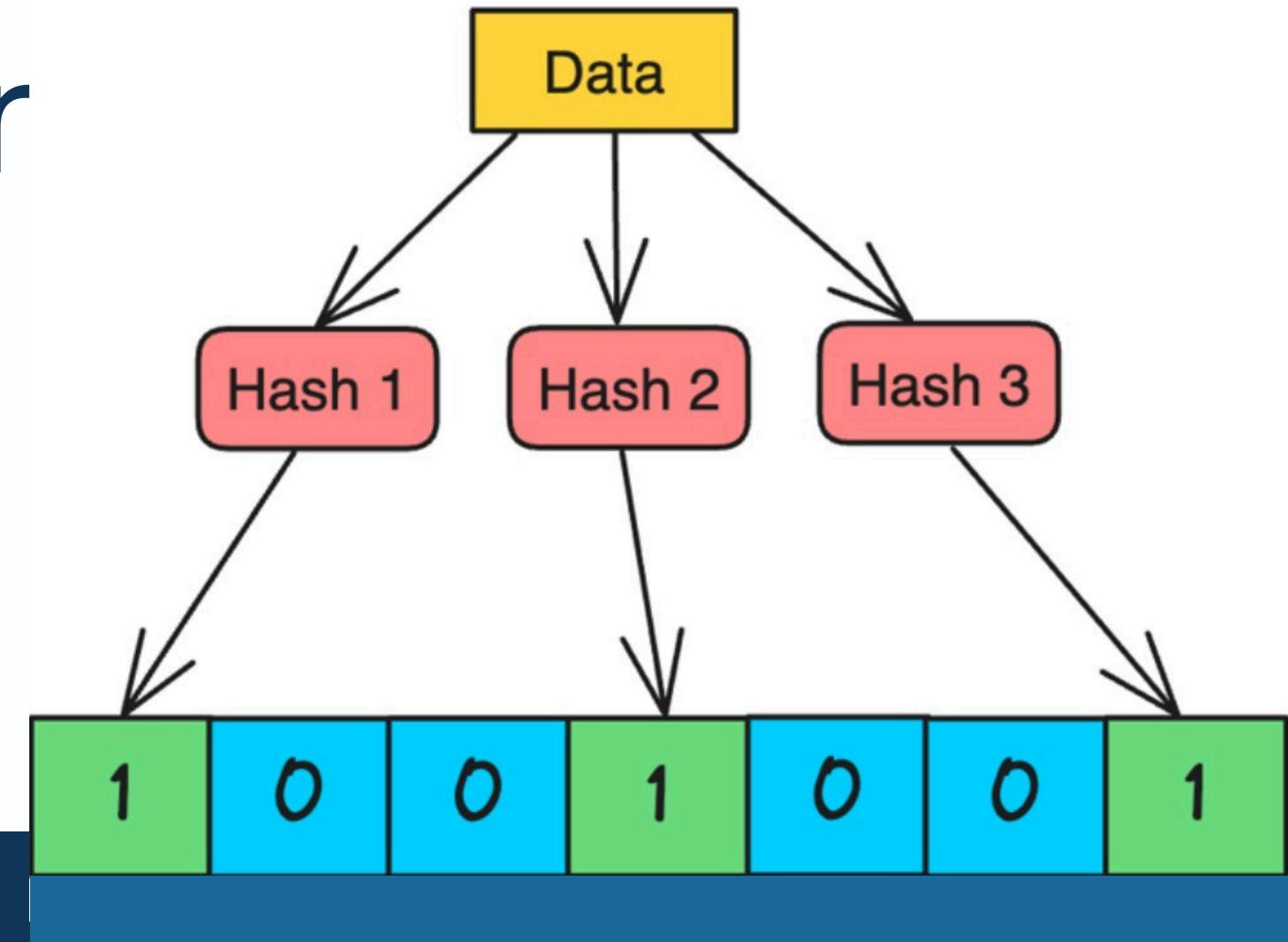


Componente 1: Bloom Filter

Struttura dati probabilistica che risponde a una domanda semplice:
"Questo elemento è nell'insieme?"

Operazioni:

- **Inserimento** di un elemento
- **Query**: verifica se un elemento è "PROBABILMENTE presente" (true) o "SICURAMENTE assente" (false)



→ DE BRUIJN GRAPH PROBABILISTICO

Dato un insieme S di k -mer, il grafo probabilistico $G_p = (V_p, E_p)$ è definito come:

- **Nodi** (V_p): $V_p = \{v \mid \text{BloomFilter.contains}(v) = \text{true}\}$
- **Archi** (E_p): Dedotti dalle estensioni: per ogni nodo v , interroga BF per le 8 possibili estensioni (4 dx, 4 sx)

→ OVER-APPROSSIMAZIONE

Poiché i Bloom Filter producono falsi positivi ma mai falsi negativi, il grafo probabilistico è una sovra-approssimazione del grafo esatto G

- $V \subseteq V_p$ e $E \subseteq E_p$

Il Problema: Falsi Positivi

Bloom Filter introduce ramificazioni spurie (false branching)

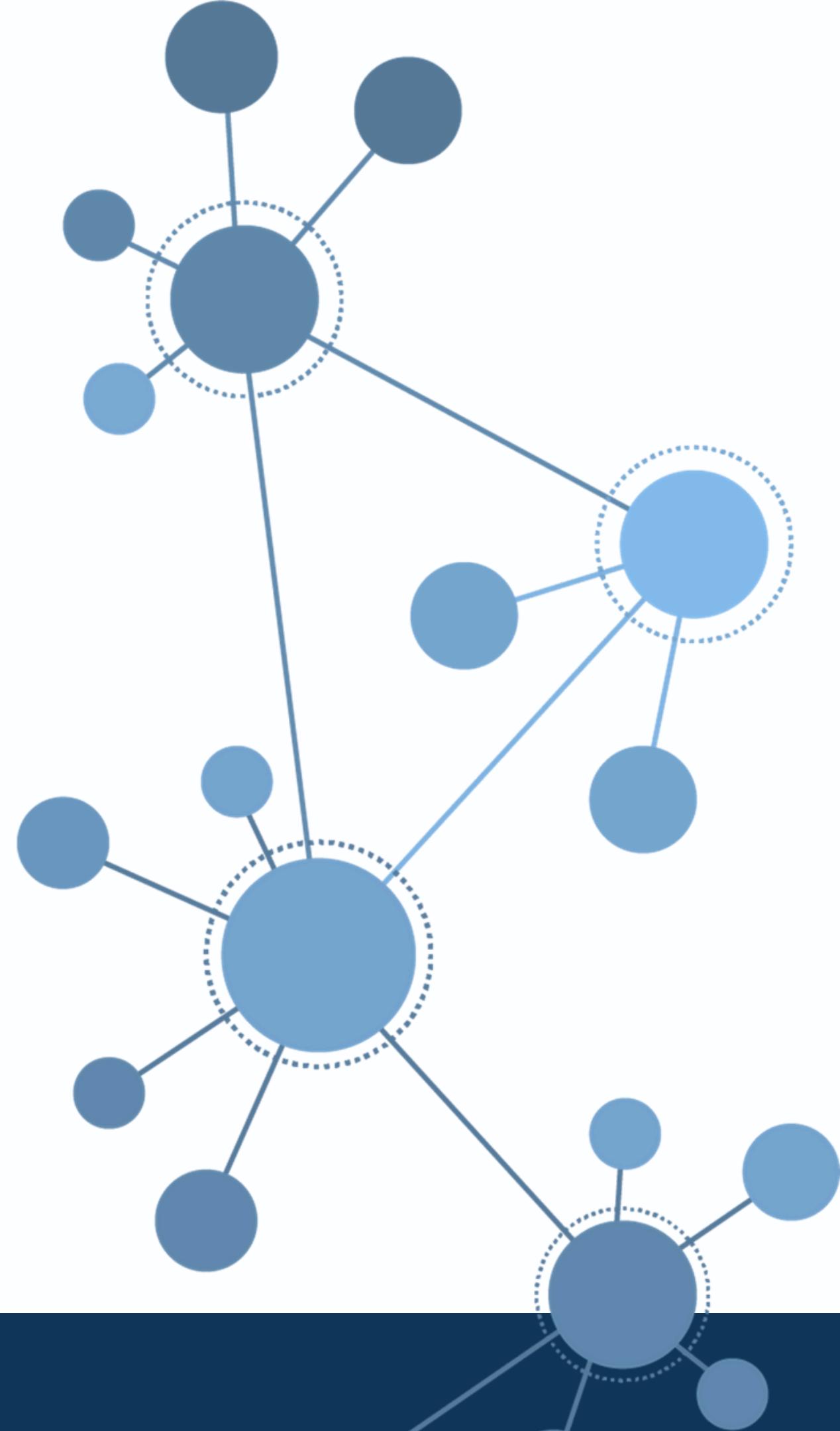
- Esempio:
 - k-mer $v = \text{ACGT}$ ($k = 4$)
 - estensioni (extright(ACGT)):
 - CGTA → true
 - CGTC → true
 - CGTG → false
 - CGTT → false
 - BF riporta 2 (1 FP) → branching artificiale → contigs frammentati

Con $F=1\%$ FP: ~192 milioni estensioni spurie

- Assembly di qualità = impossibile

→ **SOLUZIONE**

Identificare e memorizzare i falsi positivi critici



Componente 2: Critical False Positives (cFP)

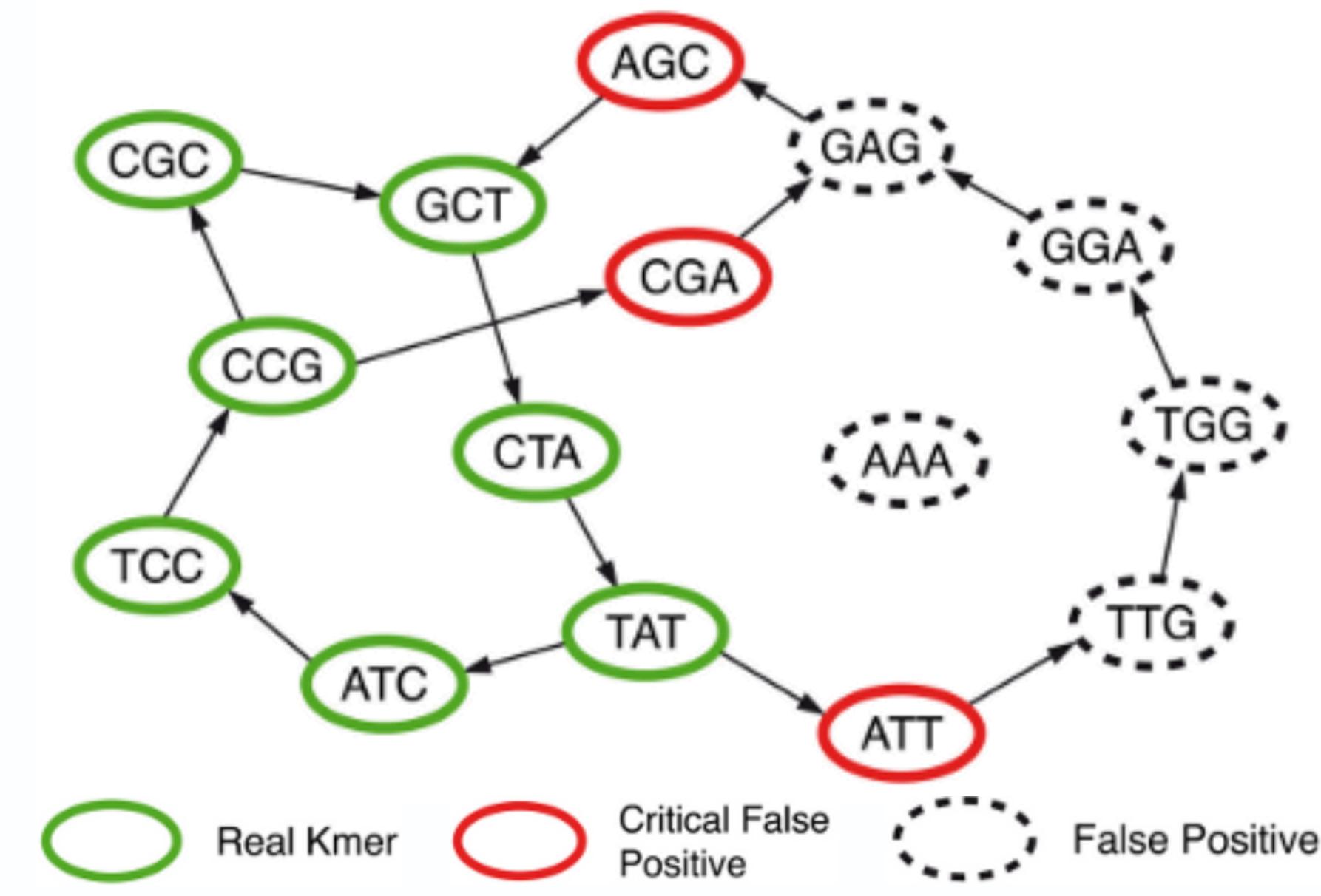
→ Non tutti i falsi positivi danno fastidio.

Definizione: Falsi positivi che sono estensioni dirette di nodi reali.

- Solo questi vengono incontrati durante l'attraversamento.

Osservazione: Durante il traversal, interrogate SOLO le estensioni di nodi reali $v \in S$

Conseguenza: Possiamo memorizzare esplicitamente SOLO i cFP, non tutti i falsi positivi possibili.



Componente 2: Critical False Positives (cFP)

Formalmente

- $cFP = P \setminus S \rightarrow$ insieme cFP
- $E[|cFP|] = 8 \cdot n \cdot F \rightarrow$ numero i cFP

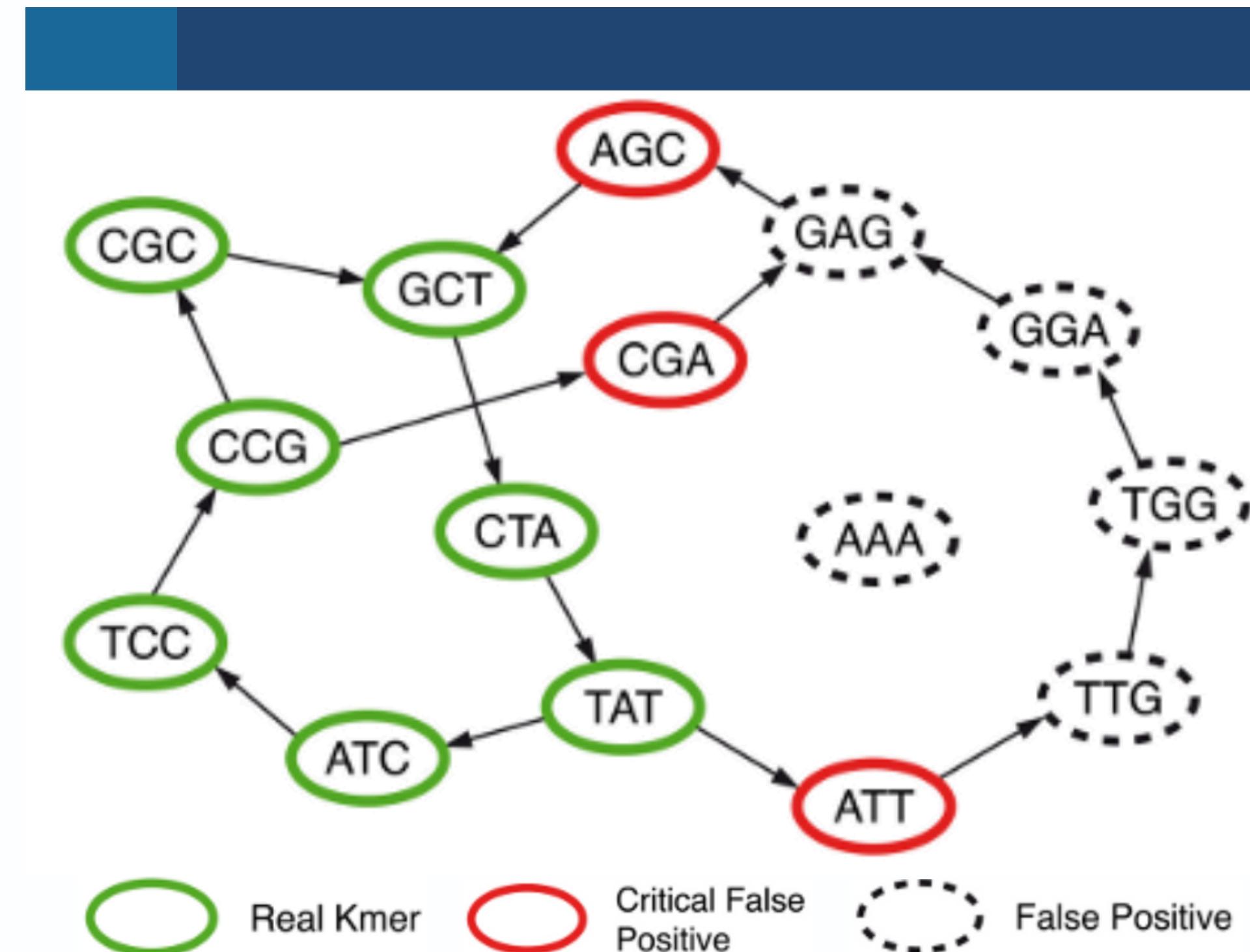
Impatto per il genoma umano:

- $|cFP| = 78.7$ miliardi
- Solo il 2.9% dei k-mer totali è un cFP.
 - 1.86 bit/k-mer

Vengono salvati a parte in una struttura dedicata (Hash Table).

→ MODIFICA DELLE QUERY AL BLOOM FILTER

Topologia = identica a grafo esatto deterministico





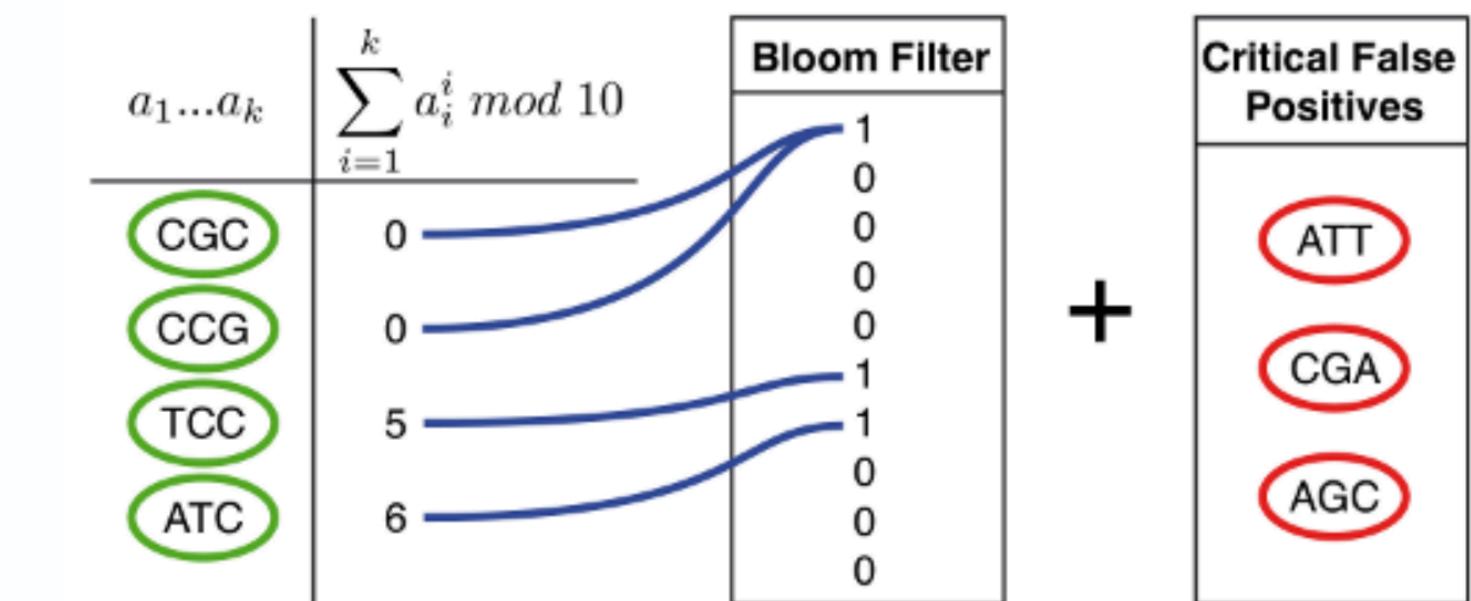
ALGORITMO: PERMETTE DI CALCOLARE I CFP USANDO MEMORIA COSTANTE

Problema:

P e S sono troppo grandi per entrare in RAM.

Soluzione: Lavoro su partizioni di S.

1. Salvare P sul disco (tutte le estensioni positive)
2. Liberare la memoria RAM dal Bloom Filter → recupera ~4 GB
3. Processare S in partizioni:
Per ogni partizione P_i di S (caricata in RAM):
 - Scansiona P dal disco sequenzialmente:
 - Se un nodo di P appartiene anche a P_i → true positive (è reale, lo scartiamo)
 - Altrimenti → è un cFP (lo memorizziamo)
 - Output: Restituiamo l'insieme cFP = $P \setminus S$



Complessità:

- Memoria: $O(M)$ — costante, configurabile
- I/O: Sequenziale (veloce anche su disco meccanico)
- Tempo: $O(|S|/M \times |P|)$

Risultato: cFP calcolato senza mai superare il budget RAM

Dimensione ottimale Bloom Filter



TRADE-OFF

- Bloom Filter grande → Pochi cFP (ma spreco RAM nel filtro).
- Bloom Filter piccolo → Tanti cFP (spreco RAM nella struttura cFP).

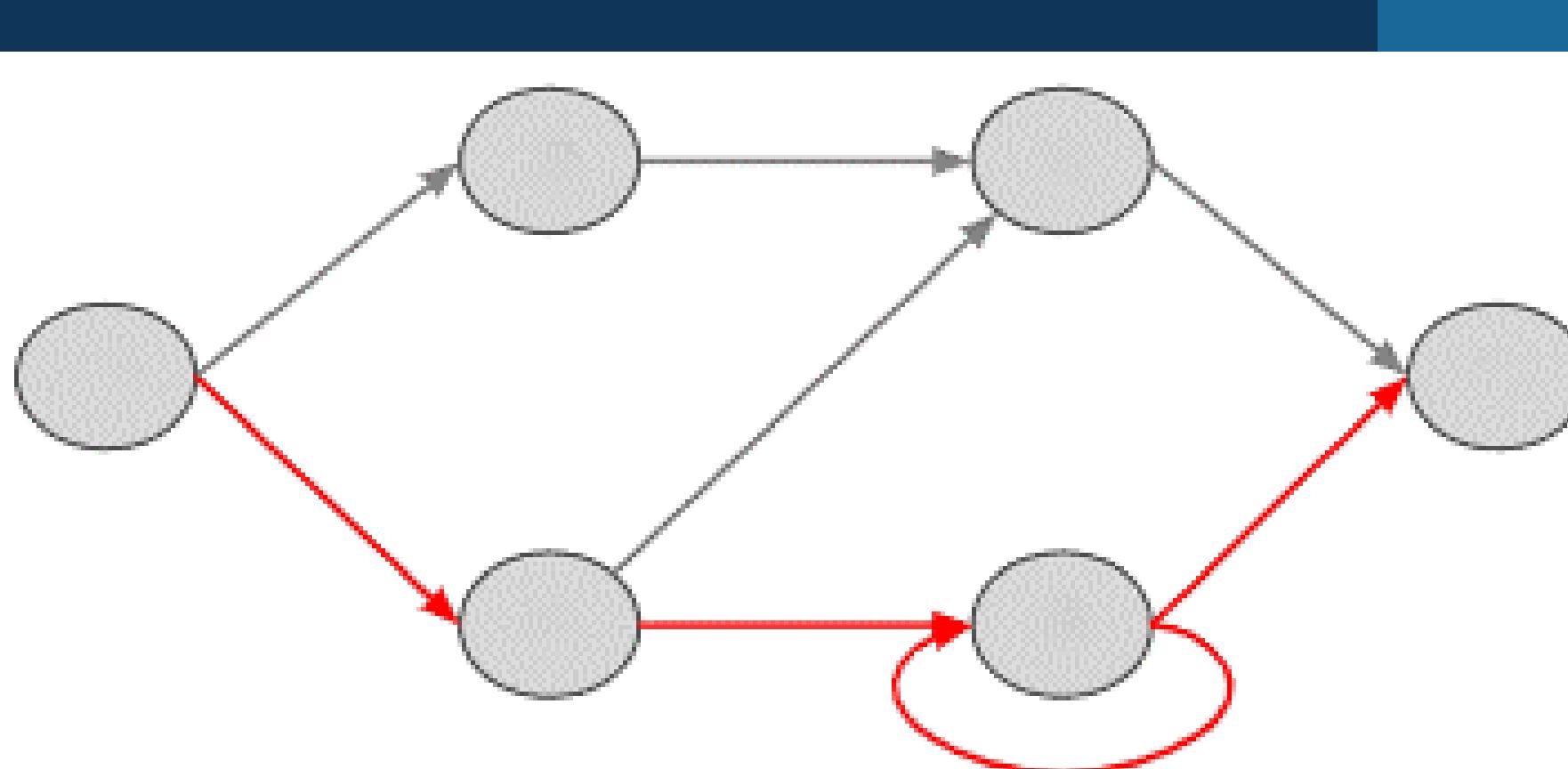
Punto di ottimo:

- Per $k=27$, il minimo si ha a 11.1 bit/k-mer per il Bloom Filter.
- Struttura cFP: 1.86 bit/k-mer.
- Totale: 13.2 bit/k-mer.

Scalabilità: il costo di cFP è costante quando si sceglie F_{opt} , mentre il Bloom Filter cresce solo logaritmicamente rispetto a k .



Componente 3: Marking Structure



→ **OBIETTIVO**

Attraversare il grafo visitando ogni nota solo una volta

→ **LIMITI**

Il Bloom Filter è immutabile (non possiamo scriverci "visitato").

→ **SOLUZIONE NAIVE**

Mantenere un array di bit separato, con un bit per ciascuno degli n k-mer

Componente 3: Marking Structure

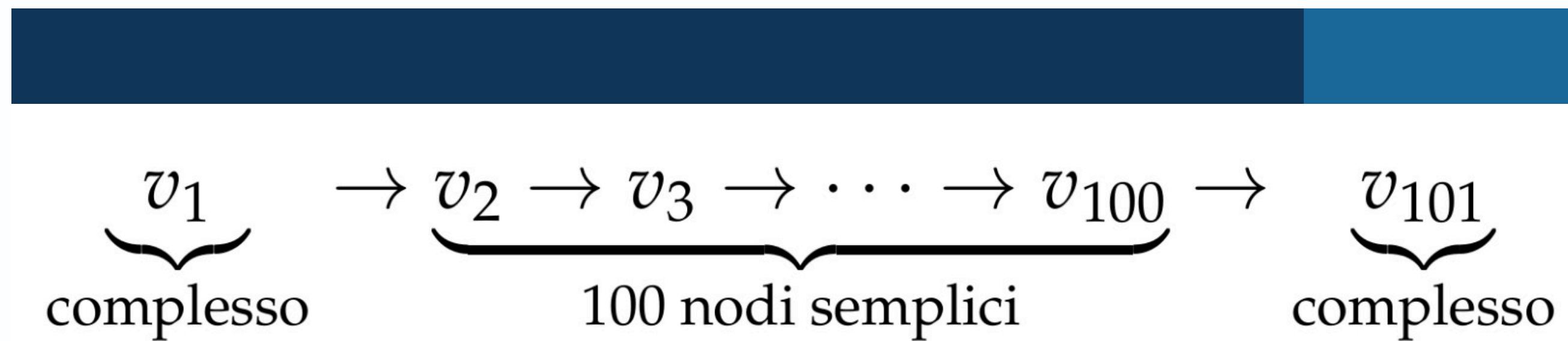
 **SOLUZIONE** Marcatura selettiva

- **Nodi Semplici** (lineari): Non vengono marcati. Condividono lo stato con il cammino.
 - **Nodi Complessi** (biforazioni): Solo questi vengono memorizzati nella `MarkingStructure`.

 RISPARMIO

Solo il 6.1% dei nodi è complesso

- Memoria ridotta a 4.42 bit/k-mer.



→ ALGORITMO: ATTRAVERSAMENTO DEL GRAFO CON MARCatura SELETTIVA

TRAVESEGRAPH

Input: Insieme S di k-mer, Bloom Filter + cFP, MarkingSet (vuota)

Per ogni k-mer v in S:

- SE v è complesso E v è già in MarkingSet:
 - SALTA (già visitato)
- ALTRIMENTI:
 - contig ← NUOVA_SEQUENZA()
 - EXTEND_PATH(v, contig, MarkingSet)
 - OUTPUT(contig)

EXTEND_PATH

Input: nodo, contig, marking

- SE nodo è complesso:
 - AGGIUNGI nodo a marking ← Segna solo biforcati
- AGGIUNGI nodo a contig
- vicini ← TROVA_VICINI(nodo) ← Usa BF + cFP
- SE vicini == 1:
 - EXTEND_PATH(vicino, contig, marking) ← Continua
- ALTRIMENTI:
 - RETURN ← Nessun vicino o biforcazione: termina

Complessità:

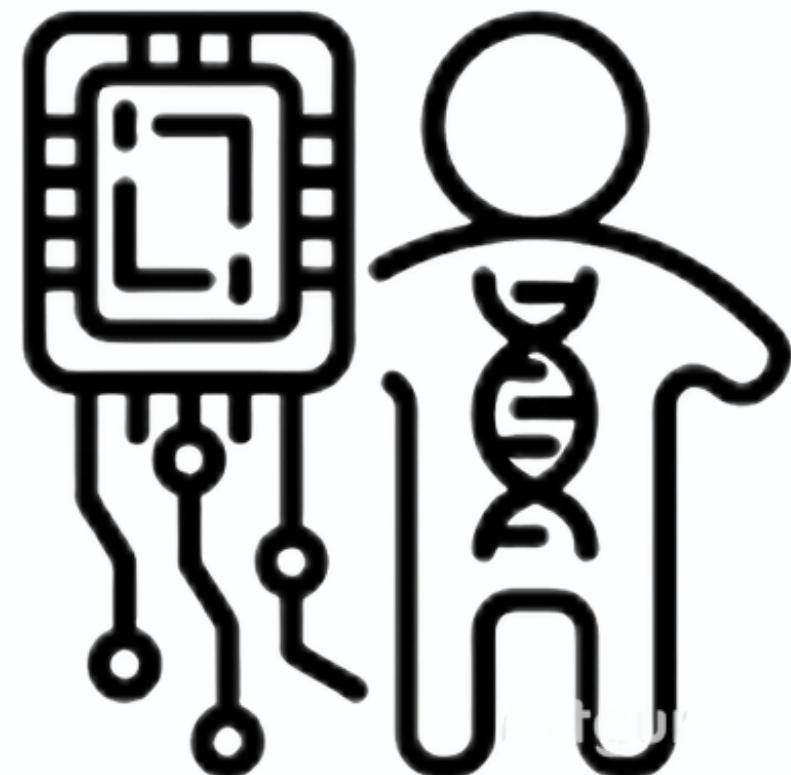
- **Tempo:** $O(|S|)$ → ogni k-mer visitato esattamente 1 volta
- **Spazio:** $O(\text{nodi_complessi})$ → solo 6% marcati



GENOMA UMANO

Componente	Dimensione	bit/k-mer	% totale
Bloom Filter	3.75 GB	11.1	65.8%
cFP	0.53 GB	1.86	9.3%
Marking structure	1.29 GB	4.42	22.6%
Overhead sistema	0.13 GB	0.48	2.3%
Totale	5.7 GB	17.4	100%

Memoria totale



Validazione Sperimentale

→ RIDUZIONE DRASTICA DELLA MEMORIA

5.7 GB per il genoma umano, 59x meno di ABYSS

→ ACCURATEZZA PRESERVATA

94.6% dei contigs corretti, comparabile agli assemblatori tradizionali

→ CORRETTEZZA GARANTITA

La struttura cFP elimina gli errori introdotti dai falsi positivi del Bloom Filter

→ SCALABILITÀ

Primo assemblatore in grado di processare un genoma umano completo su un desktop consumer



GENOMA UMANO

Metrica	Minia	C.&B.	ABYSS	SOAPdenovo
Parametro k	27	27	27	25
N50 (bp)	1,156	250	870	886
Basi assemblate (Gbp)	2.09	1.72	2.10	2.08
Cores utilizzati	1	8	21,168	116
Tempo (ore)	23	50	15	33
Memoria (GB)	5.7	32	336	140

Innovazione & Impatto



SUCCESSI

Minia dimostra che l'approccio probabilistico (Bloom Filter) è utilizzabile per l'assemblaggio esatto.

Elementi chiave del successo:

1. Struttura CFP (corregge gli errori con poca memoria).
2. Marcatura selettiva (marca solo i nodi complessi).





Università degli
studi di Salerno

GRAZIE PER L'ATTENZIONE



www.github.com/minia-analysis



v.guid16@studenti.unisa.it

c.paciello7@studenti.unisa.it



www.unisa.it

Studenti

Vittorio Guida
Costantino Paciello

Professori

Rosalba Zizza
Rocco Zaccagnino
Clelia De Felice

