Analisi di algoritmi per il Motif Finding

Tommaso Papini
tommaso.papini1@stud.unifi.it

Gabriele Bani
gabriele.bani@stud.unifi.it

UNIVERSITÀ
DEGLI STUDI
FIRENZE

11 Dicembre 2015

Un po' di background

DNA:

- sequenza di **nucleotidi**
- 4 tipi di nucleotide: A, T, C, G
- ***l*-mer**: sottosequenza di DNA di lunghezza *l*

Un po' di background

DNA:

- sequenza di **nucleotidi**
- 4 tipi di nucleotide: A, T, C, G
- ***l-mer***: sottosequenza di DNA di lunghezza *l*

Motifs

In biologia può essere necessario ricavare certe sequenze di DNA "nascoste"

- ✓ pattern di nucleotidi ripetuti (l-mer)
- ✓ utili a capire determinati comportamenti biologici
  - sequenze di attivazione di geni specifici

Il problema del Motif Finding

Il problema del **Motif Finding** consiste nel ricavare un set di t **l-mer** simili tra loro da un insieme di t sequenze di DNA.

Il problema del Motif Finding

Il problema del **Motif Finding** consiste nel ricavare un set di t **l-mer** simili tra loro da un insieme di t sequenze di DNA.

Input

- *DNA*: matrice di nucleotidi $t \times n$
  - t sequenze di DNA
  - ognuna di lunghezza n
- l: lunghezza del motif cercato

Il problema del Motif Finding

Il problema del **Motif Finding** consiste nel ricavare un set di t **l-mer** simili tra loro da un insieme di t sequenze di DNA.

Input

- *DNA*: matrice di nucleotidi $t \times n$
  - t sequenze di DNA
  - ognuna di lunghezza n
- l: lunghezza del motif cercato

Output

✓ $s = (s_1, s_2, \ldots, s_t)$: lista di t posizioni iniziali di l-mer il più simili tra loro

Un primo esempio

CGGGGCTATGCAACTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAACTTTCAAC
TACATGATCTTTTGATGCAACTTGGATGAGGGAATGATGC

Un primo esempio

CGGGGCTATGCAACTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAACTTTCAAC
TACATGATCTTTTGATGCAACTTGGATGAGGGAATGATGC

Mutazioni random

CGGGGCTATcCAgCTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAggGCAACTCCAAAGCGGACAAA
GGATGgAtCTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGAaGCAACcCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCtTGgAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCcAtTTTCAAC
TACATGATCTTTTGATGgcACTTGGATGAGGGAATGATGC

Come trovare l'l-mer più simile tra tutti?

Allineamento

CGGGGCT ***ATcCAgCT*** *GGGTCGTCACATTCCCCTTTCGATA*

TTTGAGGGTGCCCAATAA ***ggGCAACT*** *CCAAAGCGGACAAA*

GG ***ATGgAtCT*** *GATGCCGTTTGACGACCTAAATCAACGGCC*

AAGG ***AaGCAACc*** *CCAGGAGCGCCTTTGCTGGTTCTACCTG*

AATTTTCTAAAAAGATTATAATGTCGGTCC ***tTGgAACT*** *TC*

CTGCTGTACAACTGAGATCATGCTGC ***ATGCcAtT*** *TTCAAC*

TACATGATCTTTTG ***ATGgcACT*** *TGGATGAGGGAATGATGC*

Profilo e Consenso

Allineamento

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

Profilo e Consenso

Allineamento		A	T	C	C	A	G	C	T
		G	G	G	C	A	A	C	T
		A	T	G	G	A	T	C	T
		A	A	G	C	A	A	C	C
		T	T	G	G	A	A	C	T
		A	T	G	C	C	A	T	T
		A	T	G	G	C	A	C	T
Profilo	**A**	5	1	0	0	5	5	0	0
	T	1	5	0	0	0	1	1	6
	G	1	1	6	3	0	1	0	0
	C	0	0	1	4	2	0	6	1

Profilo e Consenso

Allineamento		A	T	C	C	A	G	C	T
		G	G	G	C	A	A	C	T
		A	T	G	G	A	T	C	T
		A	A	G	C	A	A	C	C
		T	T	G	G	A	A	C	T
		A	T	G	C	C	A	T	T
		A	T	G	G	C	A	C	T
Profilo	**A**	5	1	0	0	5	5	0	0
	T	1	5	0	0	0	1	1	6
	G	1	1	6	3	0	1	0	0
	C	0	0	1	4	2	0	6	1
Consenso		A	T	G	C	A	A	C	T

Score

Come definire la "bontà" di un set di l-mer?

Score

Come definire la "bontà" di un set di l-mer?

Funzione score

Si definisce una funzione **score** sul vettore $s = (s_1, s_2, \ldots, s_t)$ di posizioni iniziali:

$$Score(s, DNA) = \sum_{j=1}^{l} M_{P(s)}(j)$$

dove

- ✓ $P(s)$: matrice profilo su s
- ✓ $M_{P(s)}(j)$: elemento massimo nella colonna j-esima di $P(s)$

Score

Come definire la "bontà" di un set di l-mer?

Funzione score

Si definisce una funzione **score** sul vettore $s = (s_1, s_2, \ldots, s_t)$ di posizioni iniziali:

$$Score(s, DNA) = \sum_{j=1}^{l} M_{P(s)}(j)$$

dove

- ✓ $P(s)$: matrice profilo su s
- ✓ $M_{P(s)}(j)$: elemento massimo nella colonna j-esima di $P(s)$

Si cerca il set di posizioni iniziali s che **massimizzi** $Score(s, DNA)$!

Score: l'esempio di prima

Allineamento		A	T	C	C	A	G	C	T
		G	G	G	C	A	A	C	T
		A	T	G	G	A	T	C	T
		A	A	G	C	A	A	C	C
		T	T	G	G	A	A	C	T
		A	T	G	C	C	A	T	T
		A	T	G	G	C	A	C	T
Profilo	**A**	5	1	0	0	5	5	0	0
	T	1	5	0	0	0	1	1	6
	G	1	1	6	3	0	1	0	0
	C	0	0	1	4	2	0	6	1
Consenso		A	T	G	C	A	A	C	T

Score: l'esempio di prima

Allineamento		A	T	C	C	A	G	C	T
		G	G	G	C	A	A	C	T
		A	T	G	G	A	T	C	T
		A	A	G	C	A	A	C	C
		T	T	G	G	A	A	C	T
		A	T	G	C	C	A	T	T
		A	T	G	G	C	A	C	T
Profilo	**A**	5	1	0	0	5	5	0	0
	T	1	5	0	0	0	1	1	6
	G	1	1	6	3	0	1	0	0
	C	0	0	1	4	2	0	6	1
Consenso		A	T	G	C	A	A	C	T

$$Score(s, DNA) = 5 + 5 + 6 + 4 + 5 + 5 + 6 + 6 = 42$$

Score

Quanto può valere lo **score**?

Score

Quanto può valere lo **score**?

$$Score(s, DNA) = \begin{cases} t \cdot l, & \text{nel caso migliore} \\ \\ \lceil \frac{t}{4} \rceil \cdot l, & \text{nel caso peggiore} \end{cases}$$

Score

Quanto può valere lo **score**?

$$Score(s, DNA) = \begin{cases} t \cdot l, & \text{nel caso migliore} \\ \\ \lceil \frac{t}{4} \rceil \cdot l, & \text{nel caso peggiore} \end{cases}$$

- ✓ tl corrisponde al caso in cui tutti gli l-mer sono identici
- ✓ $\lceil \frac{t}{4} \rceil l$ corrisponde al caso in cui gli l-mer siano diversi in tutte le posizioni

Algoritmi brute force

Forza bruta

In informatica il metodo "**forza bruta**" (o ricerca esaustiva della soluzione) è un algoritmo di risoluzione di un problema dato che consiste nel verificare tutte le soluzioni teoricamente possibili fino a che si trova quella effettivamente corretta.

Simple motif search

L'idea

Esamina tutte le possibili combinazioni delle posizioni di partenza s e prendi quella con maggior *Score*.

Simple motif search

L'idea

Esamina tutte le possibili combinazioni delle posizioni di partenza s e prendi quella con maggior *Score*.

- Posizioni iniziali: $s = (1, \ldots, 1)$
- Posizioni finali: $s = ((n - l + 1), \ldots, (n - l + 1))$

Simple motif search

L'idea

Esamina tutte le possibili combinazioni delle posizioni di partenza s e prendi quella con maggior *Score*.

- Posizioni iniziali: $s = (1, \ldots, 1)$
- Posizioni finali: $s = ((n - l + 1), \ldots, (n - l + 1))$

Si utilizza il metodo *NextElement* per passare da un elemento al successivo in ordine alfabetico.

Simple motif search

Pseudocodice

```
1:  procedure SimpleMotifSearch(DNA, t, n, l)
2:      s ← (1, 1, ..., 1)
3:      bestScore ← Score(s, DNA)
4:      while true do
5:          s ← NextElement(s, t, n − l + 1)
6:          if Score(s, DNA) > bestScore then
7:              bestScore ← Score(s, DNA)
8:              bestMotif ← s
9:          end if
10:         if s = (1, 1, ..., 1) then
11:             return bestMotif
12:         end if
13:     end while
14: end procedure
```

Simple motif search

Complessità

Simple motif search

Complessità

Quante **iterazioni** fa l'algoritmo?

Simple motif search

Complessità

Quante **iterazioni** fa l'algoritmo?

- ✓ vengono esaminate tutte le possibili combinazioni
  - $(n - l + 1)$ possibili scelte per ogni posizione iniziale
  - t posizioni iniziali (una per ogni sequenza)
- ✓ $(n - l + 1)^t$ possibili combinazioni di posizioni iniziali

Simple motif search

Complessità

Quante **iterazioni** fa l'algoritmo?

- ✓ vengono esaminate tutte le possibili combinazioni
  - $(n - l + 1)$ possibili scelte per ogni posizione iniziale
  - t posizioni iniziali (una per ogni sequenza)
- ✓ $(n - l + 1)^t$ possibili combinazioni di posizioni iniziali

Quanti **passi** per calcolare *Score*?

Simple motif search

Complessità

Quante **iterazioni** fa l'algoritmo?

- ✓ vengono esaminate tutte le possibili combinazioni
  - $(n - l + 1)$ possibili scelte per ogni posizione iniziale
  - t posizioni iniziali (una per ogni sequenza)
- ✓ $(n - l + 1)^t$ possibili combinazioni di posizioni iniziali

Quanti **passi** per calcolare *Score*?

- ✓ tl per calcolare la matrice Profilo
- ✓ $4l$ per calcolare *Score*

Simple motif search

Complessità

Quante **iterazioni** fa l'algoritmo?

- ✓ vengono esaminate tutte le possibili combinazioni
  - $(n - l + 1)$ possibili scelte per ogni posizione iniziale
  - t posizioni iniziali (una per ogni sequenza)
- ✓ $(n - l + 1)^t$ possibili combinazioni di posizioni iniziali

Quanti **passi** per calcolare *Score*?

- ✓ tl per calcolare la matrice Profilo
- ✓ $4l$ per calcolare *Score*

Costo

$$\mathcal{O}(tln^t)$$

Branch & bound

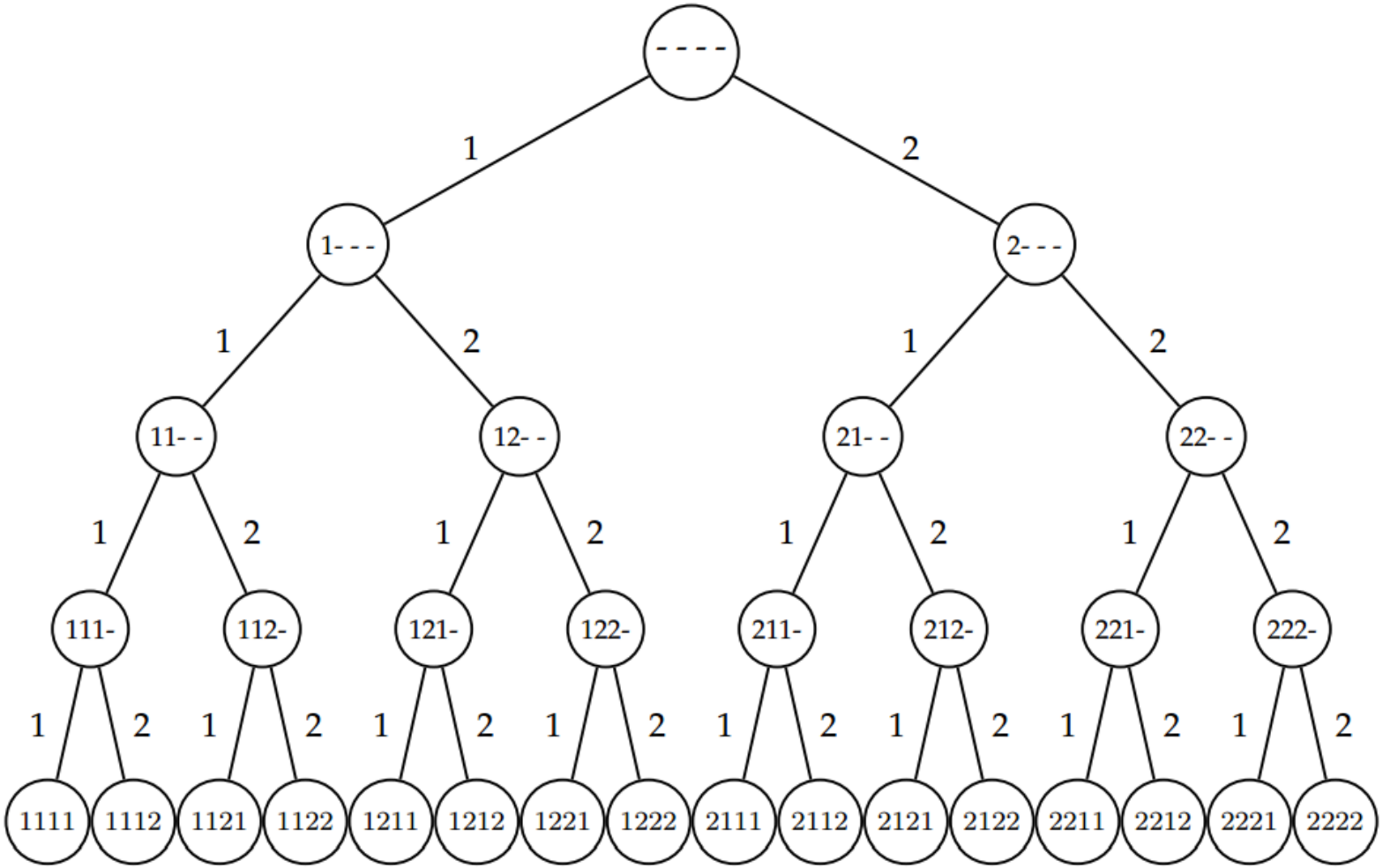
L'idea

Ottimizza *Simple Motif Search* evitando di analizzare sequenze non ottimali

- ✓ enumerazione delle sequenze tramite **alberi**
- ✓ funzione di *Score ottimistico*
  - *Score parziale* su un nodo interno
  - *Score ideale* per le restanti posizioni

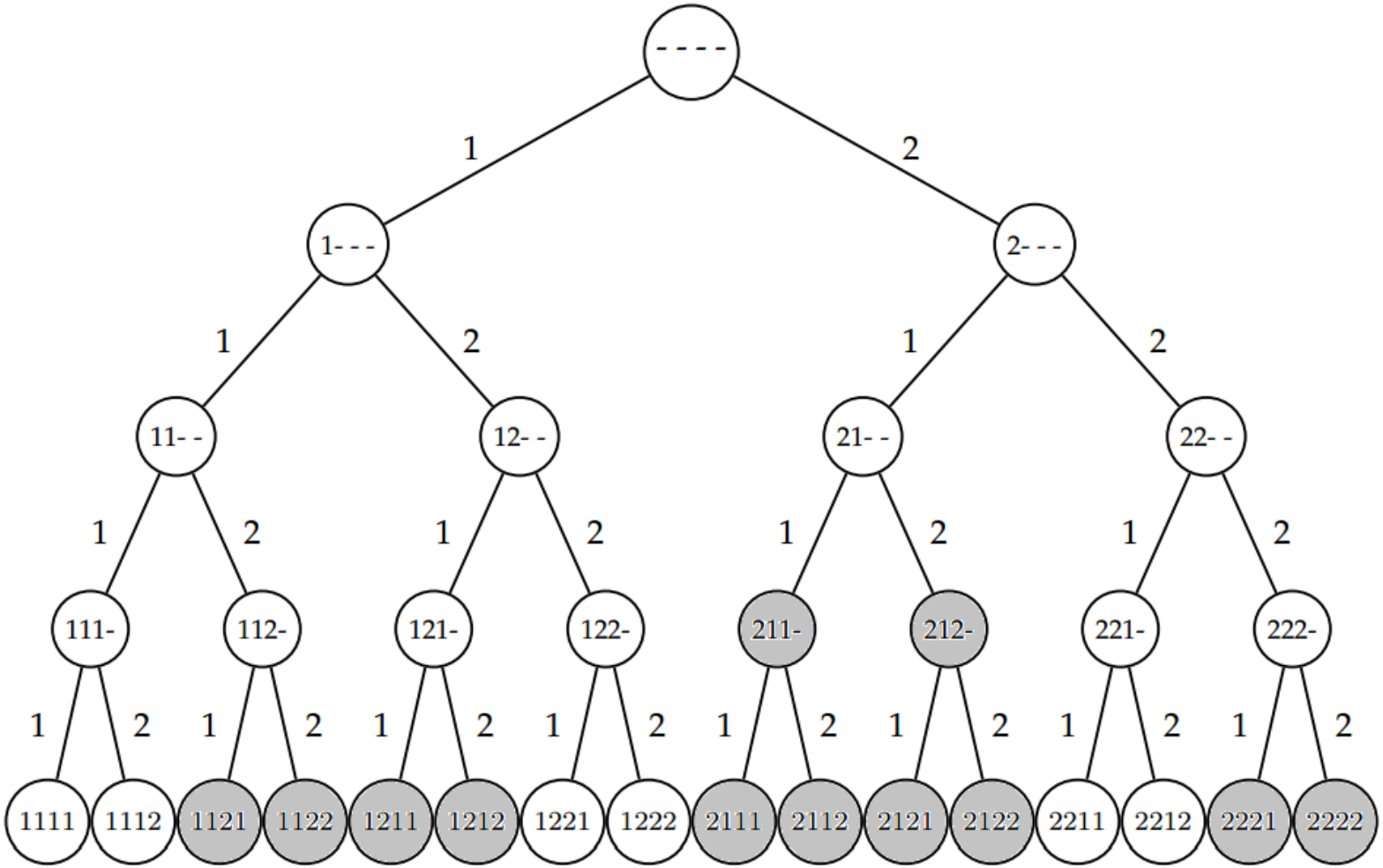
Branch & bound

Un esempio



Branch & bound

Un esempio



Branch & bound

Si esegue una visita in profondità dell'albero

Branch & bound

Si esegue una visita in profondità dell'albero

- ✓ calcola lo *Score ottimistico* per ogni nodo
- ✓ scarta i sottoalberi con *Score ottimistico* sub-ottimo

Branch & bound

Si esegue una visita in profondità dell'albero

- ✓ calcola lo *Score ottimistico* per ogni nodo
- ✓ scarta i sottoalberi con *Score ottimistico* sub-ottimo

Score ottimistico

$$\text{Score ottimistico} = \mathit{Score}(s, i, \mathit{DNA}) + (t - i) \cdot l$$

- ✓ $\mathit{Score}(s, i, \mathit{DNA})$: *Score parziale* relativo alle prime i sequenze di DNA
- ✓ $(t - i) \cdot l$: *Score parziale* delle restanti posizioni (supponendole identiche)

Branch & bound

Si esegue una visita in profondità dell'albero

- ✓ calcola lo *Score ottimistico* per ogni nodo
- ✓ scarta i sottoalberi con *Score ottimistico* sub-ottimo

Score ottimistico

$$\text{Score ottimistico} = \mathit{Score}(s, i, \mathit{DNA}) + (t - i) \cdot l$$

- ✓ $\mathit{Score}(s, i, \mathit{DNA})$: *Score parziale* relativo alle prime i sequenze di DNA
- ✓ $(t - i) \cdot l$: *Score parziale* delle restanti posizioni (supponendole identiche)

- ✓ *NextVertex* per passare al prossimo vertice (DFS)
- ✓ *Skip* per passare al prossimo vertice saltando il sottoalbero attuale

Branch & bound

Pseudocodice

```
1:  procedure BranchAndBoundMotifSearch(DNA, t, n, l)
2:      s ← (1, 1, ..., 1)
3:      bestScore ← 0
4:      i ← 1
5:      while i > 0 do
6:          if i < t then
7:              optimisticScore ← Score(s, i, DNA) + (t − i)l
8:              if optimisticScore < bestScore then
9:                  (s, i) ← Skip(s, i, (n − l + 1))
10:             else
11:                 (s, i) ← NextVertex(s, i, t, (n − l + 1))
12:             end if
13:         else
14:             if Score(s, DNA) > bestScore then
15:                 bestScore ← Score(s, DNA)
16:                 bestMotif ← s
17:             end if
18:             (s, i) ← NextVertex(s, i, t, (n − l + 1))
19:         end if
20:     end while
21:     return bestMotif
22: end procedure
```

Branch & bound

Complessità

Quante **iterazioni**?

Branch & bound

Complessità

Quante **iterazioni**?

- ✓ una per ogni nodo interno/foglia (caso pessimo)
  - $N = \frac{(n-l+1)^{t}-1}{(n-l+1)-1}$ nodi interni
  - $L = (n-l+1)^{t}$ foglie
- ✓ $N + L$ passi totali

Branch & bound

Complessità

Quante **iterazioni**?

- ✓ una per ogni nodo interno/foglia (caso pessimo)
  - $N = \frac{(n-l+1)^{t}-1}{(n-l+1)-1}$ nodi interni
  - $L = (n-l+1)^{t}$ foglie
- ✓ $N + L$ passi totali

Quanto **passi** per calcolare *Score*?

Branch & bound

Complessità

Quante **iterazioni**?

- ✓ una per ogni nodo interno/foglia (caso pessimo)
  - $N = \frac{(n-l+1)^t-1}{(n-l+1)-1}$ nodi interni
  - $L = (n-l+1)^t$ foglie
- ✓ $N + L$ passi totali

Quanto **passi** per calcolare *Score*?

- ✓ come prima!

Branch & bound

Complessità

Quante **iterazioni**?

- ✓ una per ogni nodo interno/foglia (caso pessimo)
  - $N = \frac{(n-l+1)^t-1}{(n-l+1)-1}$ nodi interni
  - $L = (n-l+1)^t$ foglie
- ✓ $N + L$ passi totali

Quanto **passi** per calcolare *Score*?

- ✓ come prima!

Costo

$$\mathcal{O}(tln^t)$$

Branch & bound

Complessità

Quante **iterazioni**?

- ✓ una per ogni nodo interno/foglia (caso pessimo)
  - $N = \frac{(n-l+1)^t-1}{(n-l+1)-1}$ nodi interni
  - $L = (n-l+1)^t$ foglie
- ✓ $N + L$ passi totali

Quanto **passi** per calcolare *Score*?

- ✓ come prima!

Costo

$$\mathcal{O}(tln^t)$$

Come prima!

- ✓ più costoso nel caso pessimo
- ✓ conveniente se esegue tanti *Skip*

Algoritmi greedy

Greedy

Un algoritmo **greedy** è un algoritmo che cerca di ottenere una soluzione ottima da un punto di vista globale attraverso la scelta della soluzione più golosa ad ogni passo locale

Greedy motif search

L'idea

Scansiona ogni sequenza di DNA una sola volta e prendi l'l-mer che massimizza lo *Score parziale*

Greedy motif search

L'idea

Scansiona ogni sequenza di DNA una sola volta e prendi l'l-mer che massimizza lo *Score parziale*

- ✓ ciclo di inizializzazione per calcolare i primi due l-mer
  - brute force
- ✓ i restanti l-mer scelti secondo lo *Score parziale*

Greedy motif search

Pseudocodice

```
1:  procedure GreedyMotifSearch(DNA, t, n, l)
2:      bestMotif ← (1, 1, ..., 1)
3:      s ← (1, 1, ..., 1)
4:      for s[1] ← 1 to n − l + 1 do
5:          for s[2] ← 1 to n − l + 1 do
6:              if Score(S, 2, DNA) > Score(bestMotif, 2, DNA) then
7:                  bestMotif[1] ← s[1]
8:                  bestMotif[2] ← s[2]
9:              end if
10:         end for
11:     end for
12:     s ← bestMotif
13:     for i ← 3 to t do
14:         for s[i] ← 1 to n − l + 1 do
15:             if Score(S, i, DNA) > Score(bestMotif, i, DNA) then
16:                 bestMotif[i] ← s[i]
17:             end if
18:         end for
19:         s[i] ← bestMotif[i]
20:     end for
21:     return bestMotif
22: end procedure
```

Greedy motif search

Complessità

Quanti **passi** fanno i primi due cicli?

Greedy motif search

Complessità

Quanti **passi** fanno i primi due cicli?

- ✓ $(n - l + 1)$ cicli ognuno
- ✓ in ogni ciclo si invoca *Score* due volte
  - $2(tl + 4l)$ passi

Greedy motif search

Complessità

Quanti **passi** fanno i primi due cicli?

- ✓ $(n - l + 1)$ cicli ognuno
- ✓ in ogni ciclo si invoca *Score* due volte
  - $2(tl + 4l)$ passi

Quanti **passi** fanno i secondi due cicli?

Greedy motif search

Complessità

Quanti **passi** fanno i primi due cicli?

- ✓ $(n - l + 1)$ cicli ognuno
- ✓ in ogni ciclo si invoca *Score* due volte
  - $2(tl + 4l)$ passi

Quanti **passi** fanno i secondi due cicli?

- ✓ $(t - 3)$ passi il ciclo esterno
- ✓ $(n - l + 1)$ passi il ciclo interno
- ✓ in ogni ciclo si invoca *Score* due volte
  - come prima!

Greedy motif search

Complessità

Quanti **passi** fanno i primi due cicli?

- ✓ $(n - l + 1)$ cicli ognuno
- ✓ in ogni ciclo si invoca *Score* due volte
  - $2(tl + 4l)$ passi

Quanti **passi** fanno i secondi due cicli?

- ✓ $(t - 3)$ passi il ciclo esterno
- ✓ $(n - l + 1)$ passi il ciclo interno
- ✓ in ogni ciclo si invoca *Score* due volte
  - come prima!

Costo

$$\mathcal{O}(tln^2 + t^2ln) =$$
$$= \mathcal{O}(tln(n + t))$$

Greedy motif search

Esattezza e approssimazione

GreedyMotifSearch potrebbe non calcolare la soluzione ottima!

- ✓ algoritmo greedy approssimato (non esatto)

Greedy motif search

Esattezza e approssimazione

GreedyMotifSearch potrebbe non calcolare la soluzione ottima!

- ✓ algoritmo greedy approssimato (non esatto)

Di quanto viene approssimata la soluzione trovata?

- ✓ fattore di approssimazione sconosciuto!

Greedy motif search

Esattezza e approssimazione

GreedyMotifSearch potrebbe non calcolare la soluzione ottima!

- ✓ algoritmo greedy approssimato (non esatto)

Di quanto viene approssimata la soluzione trovata?

- ✓ fattore di approssimazione sconosciuto!

CONSENSUS

Esiste un'implementazione (CONSENSUS) di *GreedyMotifSearch*

- ✓ risultati spesso vicini all'ottimo
- ✓ complessità molto bassa

Algoritmi randomizzati

Random

Gli algoritmi **randomizzati** sono algoritmi che impiegano un certo grado di casualità durante la loro esecuzione al fine di ottenere delle buone prestazioni nel caso medio

Greedy profile motif search

L'idea

Crea un profilo su un vettore s casuale, calcola, per ogni sequenza, l'l-mer che ha più probabilità di aver generato tale profilo e ripeti sul nuovo profilo ottenuto

Greedy profile motif search

L'idea

Crea un profilo su un vettore s casuale, calcola, per ogni sequenza, l'l-mer che ha più probabilità di aver generato tale profilo e ripeti sul nuovo profilo ottenuto

Come si definisce la probabilità che un l-mer a abbia generato un profilo P?

Greedy profile motif search

L'idea

Crea un profilo su un vettore s casuale, calcola, per ogni sequenza, l'l-mer che ha più probabilità di aver generato tale profilo e ripeti sul nuovo profilo ottenuto

Come si definisce la probabilità che un l-mer a abbia generato un profilo P?

- ✓ $Prob(a|P(s)) = \prod_{i=1}^{l} p_{a_i,i}$
- ✓ con $P(s) = (p_{ij})$ matrice profilo costruita su s

Probabilità: l'esempio di prima

Allineamento		A	T	C	C	A	G	C	T
		G	G	G	C	A	A	C	T
		A	T	G	G	A	T	C	T
		A	A	G	C	A	A	C	C
		T	T	G	G	A	A	C	T
		A	T	G	C	C	A	T	T
		A	T	G	G	C	A	C	T
Profilo	**A**	5	1	0	0	5	5	0	0
	T	1	5	0	0	0	1	1	6
	G	1	1	6	3	0	1	0	0
	C	0	0	1	4	2	0	6	1
Consenso		A	T	G	C	A	A	C	T

$a = ATCCCACT$

Probabilità: l'esempio di prima

Allineamento		A	T	C	C	A	G	C	T
		G	G	G	C	A	A	C	T
		A	T	G	G	A	T	C	T
		A	A	G	C	A	A	C	C
		T	T	G	G	A	A	C	T
		A	T	G	C	C	A	T	T
		A	T	G	G	C	A	C	T
Profilo	**A**	5	1	0	0	5	5	0	0
	T	1	5	0	0	0	1	1	6
	G	1	1	6	3	0	1	0	0
	C	0	0	1	4	2	0	6	1
Consenso		A	T	G	C	A	A	C	T

$a = ATCCCACT$

$Prob(a|P) = 5 \cdot 5 \cdot 1 \cdot 4 \cdot 2 \cdot 5 \cdot 6 \cdot 6 = 36000$

Greedy profile motif search

Pseudocodice

```
1:  procedure GreedyProfileMotifSearch(DNA, t, n, l)
2:      s ← vettore casuale di posizioni iniziali in DNA
3:      P ← P(s)
4:      bestScore ← 0
5:      while Score(s, DNA) > bestScore do
6:          bestScore ← Score(s, DNA)
7:          for i ← 1 to t do
8:              a ← l-mer dell'i-esima sequenza più probabile per P
9:              s[i] ← posizione iniziale di a
10:         end for
11:     end while
12:     return s
13: end procedure
```

Greedy profile motif search

Complessità

Quante **iterazioni** esegue il *while*?

Greedy profile motif search

Complessità

Quante **iterazioni** esegue il *while*?

- ✓ al più $t \cdot (n - l + 1)$
- ✓ in ogni ciclo si calcola *Score* due volte
  - $2(tl + 4l)$ passi
- ✓ ciclo *for*
  - t passi totali
  - in ogni passo si cerca l'l-mer con probabilità più alta
    - ★ $(n - l + 1)$ l-mer da controllare
    - ★ il calcolo di ogni probabilità richiede l passi

Greedy profile motif search

Complessità

Quante **iterazioni** esegue il *while*?

- ✓ al più $t \cdot (n - l + 1)$
- ✓ in ogni ciclo si calcola *Score* due volte
  - $2(tl + 4l)$ passi
- ✓ ciclo *for*
  - t passi totali
  - in ogni passo si cerca l'l-mer con probabilità più alta
    - ★ $(n - l + 1)$ l-mer da controllare
    - ★ il calcolo di ogni probabilità richiede l passi

Costo

$$\mathcal{O}(ln^2t^2)$$

Gibbs sampling

L'idea

Simile a *GreedyProfileMotifSearch* ma aggiornando ad ogni passo una sola sequenza (scelta in modo casuale) anziché tutte e scegliendo l'l-mer migliore in modo probabilistico e non greedy

Gibbs sampling

Pseudocodice

1: **procedure** GibbsSampling(DNA, t, n, l)
2: $s \leftarrow$ vettore casuale di posizioni iniziali in DNA
3: **repeat**
4: $x \leftarrow$ indice di una sequenza di DNA scelta casualmente
5: $P \leftarrow P(s)$ calcolata sulle restanti $t - 1$ sequenze
6: **for** $i \leftarrow 1$ **to** $n - l + 1$ **do**
7: $a \leftarrow$ l-mer della sequenza x-esima con posizione iniziale i
8: $p_i \leftarrow$ probabilità che a abbia generato P
9: **end for**
10: $j \leftarrow$ scelta casuale sulla distribuzione $(p_1, p_2, \ldots, p_{n-l+1})$
11: $s[x] \leftarrow j$
12: **until** convergenza
13: **return** s
14: **end procedure**

Gibbs sampling

Complessità

Assumendo la stessa funzione obiettivo di *GreedyProfileMotifSearch*:

- ✓ stessi passi di prima
- ✓ eccetto il ciclo sulle sequenze
  - viene analizzata una sola sequenza per ciclo

Gibbs sampling

Complessità

Assumendo la stessa funzione obiettivo di *GreedyProfileMotifSearch*:

- ✓ stessi passi di prima
- ✓ eccetto il ciclo sulle sequenze
  - viene analizzata una sola sequenza per ciclo

Costo

$$\mathcal{O}(ln^2 t)$$

Conclusioni

		Costo	Esattezza	ρ
Forza bruta	**Simple MS**	$\mathcal{O}(tln^t)$	✓	1
	Branch & Bound	$\mathcal{O}(tln^t)$	✓	1
Greedy	**Greedy MS**	$\mathcal{O}(tln(n+t))$		???
Randomizzati	**Greedy Profile MS**	$\mathcal{O}(ln^2t^2)$		???
	Gibbs Sampling	$\mathcal{O}(ln^2t)$		???

Fine!

Fine!

Domande?

Appendice

NextElement

```
1:  procedure NextElement(a, L, k)
2:      for i ← L to 1 do
3:          if a[i] < k then
4:              a[i] ← a[i] + 1
5:              return a
6:          end if
7:          a[i] ← 1
8:      end for
9:      return a
10: end procedure
```

Appendice

NextVertex

```
1:  procedure NextVertex(a, i, L, k)
2:      if i < L then
3:          a[i + 1] ← 1
4:          return (a, i + 1)
5:      else
6:          for j ← L to 1 do
7:              if a[j] < k then
8:                  a[j] ← a[j] + 1
9:                  return (a, j)
10:             end if
11:         end for
12:     end if
13:     return (a, 0)
14: end procedure
```

Appendice

Skip

```
1: procedure Skip(a, i, k)
2:     for j ← i to 1 do
3:         if a[j] < k then
4:             a[j] ← a[j] + 1
5:             return (a, j)
6:         end if
7:     end for
8:     return (a, 0)
9: end procedure
```

Riferimenti

- ✓ An Introduction to Bioinformatics Algorithms
  - N. C. Jones, P. A. Pevzner
  - Bradford
  - 2004