Algoritmi per la trasformata di Burrows–Wheeler posizionale con compressione run-length

Davide Cozzi
matr. 829827

Slide 1

Buongiorno, sono Davide Cozzi e oggi presento la mia tesi dal titolo: "Algoritmi per la trasformata di Burrows–Wheeler posizionale con compressione run-length"

Slide 2

Negli ultimi anni si è assistito a un cambio di paradigma nel campo della bioinformatica, ovvero il passaggio dallo studio della sequenza lineare di un singolo genoma a quello di un insieme di genomi, provenienti da un gran numero di individui, al fine di poter considerare anche le varianti geniche. Questo nuovo concetto è stato introdotto da Tettelin, nel 2005, con il termine di pangenoma. Grazie ai risultati ottenuti in pangenomica, ci sono stati miglioramenti sia nel campo della biologia che in quello della medicina personalizzata, grazie al fatto che, con il pangenoma, si migliora la precisione della rappresentazione di multipli genomi e delle loro differenze. Il genoma umano di riferimento (GRCh38.p14), è composto da circa 3.1 miliardi di basi, con più di 88 milioni varianti tra i genomi sequenziati, secondo i risultati ottenuti nel 1000 Genome Project. Considerando che, grazie al miglioramento delle tecnologie di sequenziamento, la quantità dei dati di sequenziamento sia destinata ad aumentare esponenzialmente nei prossimi anni, risulta necessaria la costruzione di algoritmi e strutture dati efficienti per gestire una tale mole di dati. A questo scopo, uno degli approcci più usati per rappresentare il pangenoma è attraverso un pannello di aplotipi, ovvero, da un punto di vista computazionale, una matrice di M righe, corrispondenti agli individui, e N colonne, corrispondenti ai siti con le varianti. Si specifica che, con il termine aplotipo, si intende l'insieme di alleli, ovvero di varianti che, a meno di mutazioni, un organismo eredita da ogni genitore. In questo contesto trova spazio uno dei problemi fondamentali della bioinformatica, ovvero quello del pattern matching. Inizialmente tale problema era relativo alla ricerca di una stringa (pattern) all'interno di un testo di grandi dimensioni, cioè il genoma di riferimento. Ora, con l'introduzione del pangenoma, il problema deve essere risolto sulle nuove strutture di rappresentazione del pangenoma.

Slide 3

Lo scopo di questa tesi è progettare strutture dati e algoritmi efficienti per risolvere il problema del pattern matching, inteso come ricerca dei set-maximal exact match (SMEM) tra un aplotipo esterno e un pannello di aplotipi, in una delle strutture dati più utilizzata per la rappresentazione del pangenoma: la trasformata di Burrows–Wheeler Posizionale (PBWT).
Questo progetto, svolto in collaborazione con il laboratorio BIAS e con diversi ricercatori internazionali (University of Florida, Dalhousie University e Tokyo Medical and Dental University), permetterà la gestione e lo studio (ad esempio nei GWAS) dei sempre più grandi dati provenienti dalle tecnologie di sequenziamento. Inoltre, con tale progetto, si è confermata l'ovvia correlazione tra la BWT e la PBWT, estendendo tale correlazione anche alle rispettive varianti run-length.

Slide 4

In questa breve presentazione è impossibile entrare nei dettagli di tutti i concetti teorici alla base di questo progetto. Tra di essi si hanno:

- bitvector e bitvector sparsi, strutture succinte alla base del lavoro
- intvector compressi, strutture compresse che hanno permesso di lavorare con valori interi
- straight-line program (SLP) e longest common extension (LCE) query, una grammatica context-free compressa che permette random access e LCE query in tempo logaritmico
- trasformata di Burrows–Wheeler (BWT), (inverse) suffix array ((I)SA), (permuted) longest common prefix ((P)PLCP), funzione φ, FM-index, LF-mapping, maximal exact matches (MEM), e tutte le altre teorie allo stato dell'arte su questa trasformata
- trasformata di Burrows–Wheeler run-length encoded (RLBWT), r-index, Toheold lemma, matching statistics (MS) e tutti i più recenti studi sull'uso del run-length encoded
- trasformata di Burrows–Wheeler posizionale (PBWT) e set-maximal exact match (SMEM), che invece per ovvie ragioni vedremo un po' più nel dettaglio

Slide 5

In merito alla RLBWT bisogna citare due recenti lavori, che sfruttano tale trasformata per calcolare le matching statistics e da qui calcolare MEM. Il primo è MONI (di Rossi et al.), che sfrutta il concetto di threshold (minimo lcp in una run) e il random access al pannello per il calcolo delle matching statistics.
Il secondo è PHONI (di Boucher, Rossi et al.), che sfrutta invece le LCE query per

fare il calcolo in una singola passata sul pattern, ottimizzando ancor di più la memoria necessaria.
Tali lavori sono da citare in quanto, in questo progetto, si sono create le varianti ispirate ad entrambi i lavori per la PBWT. A tal fine, come vedremo, tutti i concetti teorici della RLBWT sono stati ripensati in ottica posizionale.

Slide 6

La Trasformata di Burrows–Wheeler Posizionale (PBWT), presentata da Durbin nel 2014, viene costruita a partire da un pannello di aplotipi, rappresentato, riferendosi al solo caso biallelico, tramite una matrice binaria. La motivazione essenziale della PBWT è considerare match, e quindi anche SMEM, dove anche le posizioni di inizio e fine sono rispettate. Tale vincolo, da cui deriva il termine "posizionale", non è soddisfacibile dalla BWT ed è dovuto al fatto che ogni colonna (o indice della query) rappresenta un preciso sito per una specifica variante genica. Il funzionamento della PBWT prevede la costruzione di due insiemi di array, tramite l'ordinamento dei prefissi inversi a ogni colonna del pannello, detti insieme dei prefix array (che tiene traccia degli indici degli ordinamenti) e insieme dei divergence array (che tiene traccia della colonna d'inizio del prefisso inverso più lungo tra una riga e la precedente nel riordinamento ad una certa colonna). Il pannello, permutato tramite l'insieme dei prefix array, è detto matrice PBWT.
Qui, ad esempio, vediamo la costruzione della trasformata alla colonna 6, basata sul riordinamento fino alla quinta, e la conseguente produzione dei due array. Si noti che il divergence può anche essere sostituito dal Reverse Longest Common prefix, che memorizza la lunghezza del prefisso comune.

Slide 7

La PBWT permette di calcolare gli SMEM, di cui un esempio è qui disponibile, tra un aplotipo esterno e il pannello in tempo Avg. $\mathcal{O}(N + c)$ (dove c è il numero complessivo di SMEM), mentre una soluzione semplice impiegherebbe $\mathcal{O}(N^2M)$ tramite il famoso algoritmo 5 di Durbin che si basa sul mantenere ed eventualmente estendere un intervallo sui prefix array che contiene gli indici delle righe che hanno uno SMEM fino a quella colonna. Se l'intervallo non è più estendibile si reporta lo SMEM e si sfrutta il divergence array per computare il nuovo intervallo. Il tradeoff di questo algoritmo è la richiesta in termini di spazio (13NM bytes), dovuto ad ulteriori array necessari in memoria per il "mapping", ovvero il forward step, tra una colonna e la successiva nella matrice PBWT. Superare questo limite è l'obiettivo principale di questo progetto di tesi.

Slide 8

Si ha qui una breve panoramica delle componenti atomiche che hanno permesso la creazione delle varianti della RLPBWT:

- componenti per il mapping tra una colonna e la successiva nella PBWT, tramite bitvector sparsi (`MAP-BV`) o intvector compressi (`MAP-INT`). Tali componenti includono tutte le informazioni necessarie al mapping tra una colonna e la successiva, memorizzando l'indice di ogni run e gli equivalenti dell'FM-index. Inoltre si memorizza un singolo bool per poter risalire la carattere di ogni singola run
- componenti per le threshold (`THR-BV/THR-INT`), dove si memorizza l'indice ogni threshold
- componente per i prefix array sample (`PERM`), ovvero i valori di prefix array ad inizio e fine di ogni run
- componenti per il random access, tramite pannello di bitvector (`RA-BV`) tramite SLP (`RA-SLP`)
- componente per le LCE query con SLP (`LCE`)
- componente per il calcolo delle funzioni φ e φ^{-1} (`PHI`), che permettono data una colonna e un valore di prefix array, di sapere quale sia il valore precedente e quello successivo
- componente per il reverse longest common prefix (`RLCP`), che non scala sul numero di run

Il senso di queste multiple componenti si ritrova nel fatto che, parlando di strutture dati succinte e compresse, è difficile stimare l'effettivo spazio necessario basandosi solo sulle complessità asintotiche. Inoltre, dal punto di vista temporale, si aggiunge il problema che gli algoritmi tratti dipendono fortemente dalla caratteristica del dato. Al fine di esplorare a pieno le varie soluzioni quindi, oltre ad aver studiato e implementato le varianti di MONI e PHONI, si sono studiati gli usi di varie strutture dati sottostanti.

Slide 9

Possiamo qui velocemente vedere alcune dei confronti tra le componenti con multiple rappresentazioni in termini di complessità temporale. Avendo che $M >> \rho$ si nota come gli intvector compressi si preannunciano più veloci. Si nota inoltre come il random access (o il calcolo delle LCE query), per quanto tale caso peggiore sia irrealistico nel nostro caso dovendosi in realtà basare sulla stringa prodotta dall'SLP, sia nettamente meno performante con tale grammatica compressa.

Slide 10

Possiamo qui confrontare velocemente a sinistra le stime di memoria dell'uso di un bitvector sparso e un intvector compressso, stime derivanti dai dati di SDSL (la lib usata) e dal numero di run attese (proporzionale a quanto visto con i dati del 1000 genome project). Si nota quindi che, per quanto all'inizio gli intvector compressi richiedano

meno memoria tale comportamento è destinato ad invertirsi all'aumentare del numero di sample.
A destra invece possiamo notare il forte vantaggio dell'SLP, anticipando già i risultati, rispetto ai pannelli del 1000 genome project.

Slide 11 e 12

Si ha quindi lo schema che mostra le 8 strutture dati studiate. Di fatto le soluzioni sono 3, che diventano 8 per il discorso di dualità visto precedentemente. Si hanno quindi:

1. le strutture che vediamo in centro, basate sul rifacimento dell'algoritmo 5 di Durbin e sull'uso dell'RLCP. Non è in grado di sapere quali siano le righe che presentano un certo SMEM ma solo quante
2. le soluzioni ispirate a MONI
3. le soluzioni ispirate a PHONI

Le ultime due soluzioni computano esattamente quali righe presentano uno SMEM.
È superfluo notare come le prime due soluzioni non siano effettivamente utilizzabili ma sono state citate in quanto punto iniziale di questa tesi e dei primi studi sull'uso del run-length.
magari qui si può dire di più sulla pic

Slide 13

In questa slide possiamo visualizzare un semplice esempio di matching statistics con la PBWT. Che sono così definite:

Definizione 1 *Dato un pannello X, di dimensioni $M \times N$, con M individui e N siti, e un aplotipo esterno/pattern z, tale che $|z| = N$, si definisce matching statistics di z su X un array* MS *di coppie* $(\mathsf{row}, \mathsf{len})$*, di lunghezza N, tale che (avendo che x_i indica l'i-esima riga del pannello X):*

- $x_{\mathsf{MS}[i].\mathsf{row}}[i-\mathsf{MS}[i].\mathsf{len}+1, i] = z[i-\mathsf{MS}[i].\mathsf{len}+1, i]$*, ovvero si ha che l'aplotipo query ha un match, terminante in colonna i, con la riga* $\mathsf{MS}[i].\mathsf{row}$
- $z[i - \mathsf{MS}[i].\mathsf{len}, i]$ *non è un suffisso terminante in colonna i di un qualsiasi sottoinsieme di righe di X. In altri termini, il match non deve essere ulteriormente estendibile a sinistra*

E ne segue il seguente lemma:

Lemma 1 *Dato un pannello X, di dimensioni $M \times N$, con M individui e N siti, un aplotipo esterno/pattern z, tale che $|z| = N$, e il corrispondente array di matching statistics* MS *si ha che $z[i - l + 1, i]$ presenta uno* SMEM *di lunghezza l in con la riga* $\mathsf{MS}[i].\mathsf{row}$ *del pannello X sse:*

$$\mathsf{MS}[i].\mathsf{len} = l \wedge (i = N - 1 \vee \mathsf{MS}[i].\mathsf{len} \geq \mathsf{MS}[i + 1].\mathsf{len})$$

Ad esempio, con $k = 5$, abbiamo $\mathsf{row} = 13$ e $\mathsf{len} = 6$, infatti con la riga 6 si ha un suffisso comune lungo 6, terminante in $k = 5$, con la riga 13.

Slide 14

Possiamo qui vedere, ad alto livello, come funzioni la struttura per il calcolo delle funzioni φ che permettono, dato un valore di prefix array e una colonna, di computare il valore di prefix array precedente e quello successivo. Il computo si basa sull'uso dei prefix array sample ed eventualmente dell'ultimo prefix array considerando quando due linee consecutive si separano durante le permutazioni. Quando si separano sono sicuramente una fine di run e una testa di run e quindi si può sapere che fino a quella colonna sono consecutive. Si tiene traccia quindi tramite bitvector di dove una riga sia testa o coda di run e tramite intvector compresso dell'indice della riga sopra e sotto. L'ultimo prefix array serve per quelle righe che non si "spezzano" mai. In tal modo, come visibile nell'esempio per $k = 0$ (dove si cerca chi sia sotto m e chi sopra j), si può computare con la funzione rank a che colonna sia lo spazio e quindi accedere agli intvector per computare l'indice di prefix array.
Quindi si circa in su e in giù a partire dallo SMEM di MS fino a che si ha il medesimo SMEM (avendo che sono tutti consecutivi nel riordinamento in una certa colonna)

Slide 15

La sperimentazione, orchestrata tramite `snakemake`, è stata effettuata su una macchina con processore Intel Xeon E5-2640 V4 (2, 40GHz), 756GB di RAM, 768GB di swap e sistema operativo Ubuntu 20.04.4 LTS.
Si sono confrontate l'implementazione in della RLPBWT e l'implementazione in C ufficiale della PBWT.
Si segnala che la RLPBWT supporta lo studio multithread di stringhe ma è stato usato un singolo thread per questi test a fini di avere risultati più comparabili.
Vediamo qui le caratteristiche dei pannelli usati, relativi alla phase 3 del 1000 Genome Project. Si nota come il numero di run sia molto inferiore all'altezza del pannello, fattore che conferma l'utilità del run-length encoding.

Slide 16