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

Davide Cozzi, 829827, d.cozzi@campus.unimib.it

Introduzione

Negli ultimi anni si è assistiti ad un cambio di paradigma nel campo della *bioinformatica*, ovvero il passaggio dallo studio della *sequenza lineare di un singolo genoma* a quello in cui si studia un grande insieme di genomi, provenienti da diversi individui, al fine di poter considerare anche le *variazioni* o *varianti geniche*. Questo nuovo concetto è stato nominato per la prima volta da Tettelin, nel 2005, con il termine **pangenoma**.
Grazie ai risultati ottenuti in *pangenomica* si sono permessi miglioramenti negli studi sia nel campo della *biologia* che, ad esempio, in quello della *medicina personalizzata*, soprattutto grazie ai **genome-wide association studies (*GWAS*)**. In tali studi si necessita di studiare grandi collezioni di genomi al fine di poter identificare associazioni tra *varianti geniche* e la propensione ad una certa malattia o ad un suo particolare sintomo.
Dando qualche dato quantitativo possiamo pensare al fatto che il genoma umano, ad esempio prendendo come reference il *GRCh38.p14*, è composto da circa 3.1 miliardi di basi, comportanti circa 59,000 geni. Inoltre, i risultati del *1000 Genome Project* hanno portato ad identificare più di 88 milioni di varianti tra i genomi sequenziati. Tra esse si hanno, ad esempio, 84.7 milioni di **Single Nucleotide Polymorphisms (*SNPs*)**, ovvero differenze di singole basi azotate, 3.6 milioni di piccole **inserzioni/delezioni (*indel*)** di basi e circa 60,000 **varianti strutturali**, che coinvolgono più di 50 nucleotidi. Questi numeri fanno capire come, dal punto di vista della *computer science*, siano richiesti algoritmi e strutture dati efficienti in grado di gestire quest'incredibile mole di dati, considerando che è destinata ad aumentare nel prossimo futuro, grazie al miglioramento delle tecnologie di sequenziamento (**Next Generation Sequencing (*NGS*)** e **Third-Generation Sequencing**).
In tale ottica, sempre da un punto di vista computazionale, il *pangenoma* può essere rappresentato in molteplici modi, tramite strutture dati che permettano di memorizzare tutte le variazioni tra i genomi dei vari individui. Solitamente si hanno due rappresentazioni principali: una tramite un *grafo del pangenoma* e una tramite un *pannello di aplotipi*. In entrambi i casi sono necessarie metodologie efficienti sia per la memorizzazione della struttura dati che per le interrogazioni alla stessa. In questa tesi si è studiata la seconda casistica, ovvero la rappresentazione tramite *pannello di aplotipi*, dove con **aplotipo** si intende l'insieme di alleli, ovvero di varianti, che un organismo eredita da ogni genitore. L'informazione combinata di tutti gli aplotipi in un individuo è detta invece **genotipo**.
In questo contesto trova spazio, ovviamente, uno dei problemi fondamentali della *bioinformatica*, ovvero quello del **pattern matching**. Inizialmente tale concetto era relativo allo studio di un breve pattern all'interno di un testo di grandi dimensioni. Ora, con l'introduzione del *pangenoma*, tale problema si è adattato alle nuove strutture dati. Si ha, ad esempio, il caso studiato in questa tesi: il pattern matching tra un *aplotipo esterno* e un *pannello di aplotipi*.

Preliminari

Al fine di comprendere al meglio i metodi usati in questa tesi bisogna introdurre alcuni concetti preliminari. Il primo è quello della ben nota **Burrows-Wheeler transform (*BWT*)**, una trasformata reversbile che ha permesso di ottenere algoritmi efficienti per il pattern matching grazie all'indicizzazione tramite **FM-index**, un *self-index* che permette di lavorare sulla BWT senza averla effettivamente in memoria (avendo in memoria solo l'indice stesso). La BWT è fortemente legata al concetto di **Suffix Array (*SA*)**. Infatti, dato un testo T lungo n si ha che SA è la lista lessicograficamente ordinata delle posizioni di partenza di tutti i suoi suffissi, avendo che $SA[i] = j$ sse $T[j, |T| - 1]$ è l'i-esimo suffisso lessicograficamente minore di T, avendo che $T[SA[i], n-1] \prec T[SA[j], n-1]$. Un altro concetto spesso usato sono le lunghezze del cosiddetto **Longest Common Prefix (*LCP*)** tra ogni suffisso consecutivo in SA.
Una caratteristica di questa trasformata è la tendenza a produrre una sequenza che presenta caratteri uguali in posizioni consecutive. Questo fenomeno è dovuto alle ripetizioni di sotto-stringhe nel testo, avendo che tali eventi, per motivazioni biologiche, sono frequenti nell'ambito genomico, dove si ricordi il testo è costruito

su un alfabeto $\Sigma = \{a, c, g, t\}$. Al fine di sfruttare tali caratteri uguali consecutivi si è quindi ideata la **Run-length encoded Burrows-Wheeler transform (*RLBWT*)** dove una sequenza massimale di caratteri uguali, detta *run*, viene memorizzata in modo efficiente come coppia (carattere, lunghezza della run). Ad esempio la stringa `aaaaaa` sarebbe memorizzata come `(a,6)`. Alcuni paper recenti hanno quindi proposto un nuovo tipo di indicizzazione, tramite il cosiddetto **r-index**, per questa struttura dati compressa, al fine di ottenere un indice che non fosse lineare sulla lunghezza del testo ma sul numero di run della BWT ottenuto da esso. Tali articoli hanno portato alla produzione di due tool, *MONI* e *PHONI*, alle cui tecniche è fortemente ispirata questa tesi. Senza entrare nei dettagli, queste due soluzioni avevano entrambe l'obbiettivo di calcolare i **Maximal Exact Matches (*MEMs*)** tra un testo T, lungo n è un pattern P, lungo m. Si ha infatti che una sotto-stringa, di lunghezza l e iniziante all'indice i, del pattern, ovvero $P[i, i+l-1]$ è un MEM di P in T sse $P[i, i+l-1]$ è anche una sotto-stringa di T ed essa non può essere stesa in nessuna direzione, avendo che né $P[i-1, i+l-1]$ né $P[i, i+l]$ sono sotto-stringhe di T. l'algoritmo per il calcolo dei MEM prevede prima la costruzione dell'array delle **Matching Statistics (*MS*)**, ovvero un array lungo m, di coppie (posizione *pos*, lunghezza *len*), tale che $T[MS[i].pos, MS[i].pos + MS[i].len - 1] = P[i, i + MS[i].len - 1]$ e che $P[i, i + MS[i].len]$ non occorre in T. Quindi si ha un match tra P e T lungo $MS[i].len$ a partire da $MS[i].pos$ in T e da i in P che non è ulteriormente estendibile. Gli autori, per il calcolo delle *Matching Statistics* hanno usato, in *MONI*, anche il concetto di *threshold*, definito come il minimo valore dell'*array LCP*. Un ulteriore miglioramento si ha avuto in *PHONI* con l'uso delle **LCE query** per il calcolo dell'array MS. Si ha che, dato un testo T, tale che $|T| = n$, il risultato della **LCE query** tra due posizioni i e j, tali che $0 \leq i, j < n$, corrisponde al più lungo prefisso comune tra le sotto-stringhe che hanno come indice di partenza i e j, avendo quindi il più lungo prefisso comune tra $T[i, n-1]$ e $T[j, n-1]$.
I concetti appena espressi verranno riformulati in questa tesi, in quanto essa tratta la costruzione di una versione **run-length encoded** della **Positional Burrows–Wheeler transform (*PBWT*)**, ovvero la variante posizionale della BWT, che viene costruita a partire da un pannello di aplotipi (nello specifico limitandosi al caso bi-allelico avendo che il pannello è composto da simboli nell'alfabeto $\Sigma = \{0, 1\}$). Tale struttura è frutto degli studi fatti da Durbin, nel 2014. Tale struttura dati assume in input un pannello X, composto da M individui/righe e N siti/colonne, e produce, tramite l'*ordinamento dei prefissi inverso*, ad ogni colonna k, due insiemi di array. Tali insiemi sono detti **insieme dei prefix array** e **insieme dei suffix array**. Il primo, denotato a, contiene, per ogni colonna k e ogni posizione i, l'indice dell'aplotipo m nel pannello originale. Si ha quindi che $a_k[i] = m$ sse X_m (denotando la riga m-esima di X) è l'i-esimo aplotipo secondo l'ordinamento inverso fatto in colonna k. Il secondo insieme, denotato con d, indica l'indice della colonna iniziale del suffisso comune più lungo, che termina nella colonna k, tra una riga e la sua precedente all'ordine inverso-prefisso ottenuto per la k-esima colonna.
Si può quindi apprezzare il collegamento naturale che si ha tra la $PBWT$ e la BWT, avendo che il *prefix array* della prima corrisponde al *suffix array* della seconda mentre il *divergence array* altro non è che una diversa rappresentazione dell'*array LCP*.
Cruciale è il fatto che tale struttura permetta di calcolare i match massimali tra un aplotipi esterno e il pannello in tempo $\mathcal{O}(MN)$, mentre una soluzione naive impiegherebbe un tempo proporzionale a $\mathcal{O}(M^2N)$. Il tradeoff di tale algoritmo, conosciuto anche come **algoritmo 5 di Durbin**, è la richiesta in termini di spazio. L'autore stesso infatti stima la richiesta di $13NM$ byte in memoria per poter eseguire l'algoritmo.
Questo ultimo aspetto è stato il punto di partenza di questi progetto di tesi, il quale ha come obbiettivo il creare una variante *run-length encoded* della *PBWT*, detta *RLPBWT*, che permetta, in modo efficiente dal punto di vista della memoria, di poter calcolare *match massimali* tra una sequenza query e il pannello.
Al fine di raggiungere tale obbiettivo si sono usate altre nozioni. In primis, si sono sfruttate le cosiddette **strutture dati succinte**, ovvero strutture per le quali, assumendo $\mathcal{X}$ sia il numero di bit ottimale per memorizzare dei dati, si richiede uno spazio $\mathcal{X} + o(\mathcal{X})$. Nel dettaglio si sono usati i cosiddetti **sparse bitvector**, strutture che richiedono in memoria $\approx m\left(2 + \log \frac{n}{m}\right)$ bit, con n lunghezza del bitvector e m numero di simboli $\sigma = 1$ in esso. L'efficienza delle operazioni che si possono fare con tali strutture dati, incredibilmente efficienti dal punto di vista dello spazio occupato, sono uno dei punti cruciali del funzionamento della $RLPBWT$.
Infine, al fine di memorizzare il pannello in modo efficiente, si è usata una struttura dati, detta **Straight-line programs (*SLPs*)**. tale struttura altro non è che una **grammatica context-free**, che genera una e una sola parola, la quale permette sia di effettuare *random access* al testo, non in tempo costante, che di calcolare le *LCE query*. Per rendere l'idea della capacità di compressione di un *SLP* si pensi che per un pannello 100000×358653, che normalmente occupa $\sim 35gb$, si produce un *SLP* che pesa appena $\sim 15mb$.

Metodi

Il processo per ottenere la *RLPBWT* è stato un processo incrementale, partito con la creazione di una variante naive, basata sulle intuizioni avute a fine 2021 da Gagie, che propose una prima variante della struttura senza però specificare come effettuare query alla stessa. Tale proposta, inoltre, presentava alcune ridondanze tra i dati memorizzati.

Risultati e conclusioni