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

Davide Cozzi
matr. 829827

Negli ultimi anni si è assistito ad 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 nominato per la prima volta 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 come la quantità dei dati di sequenziamento sia destinata ad aumentare esponenzialmente nei prossimi anni, grazie al miglioramento delle tecnologie di sequenziamento (Next Generation Sequencing e Third-Generation Sequencing), risulta necessaria la costruzione di algoritmi e strutture dati efficienti per gestire una tale informazione. In merito, uno degli approcci più usati per rappresentare il pangenoma è un pannello di aplotipi, ovvero, computazionalmente, 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 un organismo eredita da ogni genitore.

In questo contesto trova spazio uno dei problemi fondamentali della bioinformatica, ovvero quello del pattern matching. Inizialmente tale concetto era relativo allo studio di un piccolo pattern all'interno di un testo di grandi dimensioni, ovvero il genoma di riferimento. Ora, con l'introduzione del pangenoma, tale problema si è adattato alle nuove strutture dati.

Lo scopo di questa tesi è ottimizzare 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: la *trasformata di Burrows-Wheeler Posizionale (PBWT)*. Il progetto di tesi, svolto in collaborazione con il prof. Gagie (Dalhousie University) e la prof.ssa Boucher (University of Florida), ha quindi permesso lo sviluppo di una variante **run-length encoded** della **PBWT**, detta **RLPBWT**, che permettesse di risolvere tale problema.

## Stato dell'arte

Si introducono ora i principali algoritmi e le principali strutture dati, presenti allo stato dell'arte, posti a fondamento di questo progetto di tesi.

La prima struttura dati che bisogna citare è la ben nota *trasformata Burrows-Wheeler (BWT)*, una trasformata reversibile che ha permesso di ottenere algoritmi efficienti per il pattern matching su un testo. La BWT, inoltre, è fortemente legata al concetto di Suffix Array (SA), definito come la lista lessicograficamente ordinata delle posizioni di partenza di tutti i suffissi di un testo, e a quello delle lunghezze del Longest Common Prefix (LCP) tra ogni suffisso consecutivo indicizzato nel SA.

Durbin, nel 2014, propose una struttura dati ispirata alla BWT, detta *Trasformata di Burrows-Wheeler Posizionale (PBWT)*, la quale viene costruita a partire da un pannello di aplotipi, rappresentato, riferendosi al solo caso bi-allelico, tramite una matrice binaria. Il funzionamento di tale struttura dati prevede la costruzione di due insiemi di array, tramite l'*ordinamento dei prefissi inverso* ad ogni colonna del pannello. Tali insiemi sono detti *insieme dei prefix array* e *insieme dei divergence array*. Il primo, denotato $a$, contiene, per ogni colonna e ogni posizione, l'indice dell'aplotipo nel pannello originale riordinato secondo l'ordinamento inverso alla data colonna. Quindi, si ha, in ogni colonna, la permutazione degli indici di riga secondo l'ordinamento inverso alla colonna $k$-esima. Il secondo insieme, invece, indica l'indice della colonna iniziale del suffisso comune più lungo, che termina nella colonna nella colonna $k$, tra una riga e la sua precedente secondo l'ordinamento inverso ottenuto per la $k$-esima colonna. Il pannello ottenuto con le permutazioni dettate dal prefix array viene chiamato matrice PBWT. Cruciale è il fatto che tale struttura permetta di calcolare gli SMEM tra un aplotipo 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 ($13NM$ bytes secondo l'autore) e superare questo limite è l'obiettivo principale di questo progetto di tesi.

Tornando a parlare della BWT, una caratteristica di questa trasformata è la tendenza a produrre una sequenza con caratteri uguali in posizioni consecutive. Questo fenomeno è dovuto alle ripetizioni di determinate sottostringhe nel testo, avendo che tali ripetizioni, per motivazioni biologiche, sono frequenti in ambito genomico. Al fine di sfruttare tali caratteri uguali consecutivi si è quindi ideata la *trasformata Burrows-Wheeler Run-Length encoded (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 proposto un nuovo tipo di indicizzazione, il cosiddetto *r-index*, per questa struttura dati compressa. In tal modo, si ottiene un indice che non lineare sulla lunghezza del testo ma sul numero di run della sua BWT. L'r-index include la RLBWT e un suffix array sample, ovvero i valori del SA all'inizio e alla fine di ogni run. Tali articoli hanno portato alla produzione di due tool, *MONI* e *PHONI*, alle cui tecniche è fortemente ispirata questa tesi. Queste due soluzioni hanno entrambe l'obiettivo di calcolare i maximal exact match (MEM) tra un testo e un pattern, ovvero sottostringhe del pattern che sono anche sottostringhe del testo, avendo che tale match non può essere

esteso, in entrambe le direzioni, senza introdurre un mismatch.

Gli autori hanno rilevato come il calcolo dei MEM sia direttamente correlato alla costruzione dell'array delle *Matching Statistics (MS)*. Tale array, lungo quanto il pattern e formato da coppie (posizione *pos*, lunghezza *len*), annota, per ogni posizione del pattern, la lunghezza *len* di un match, anche non massimale, con una sottostringa nel testo avente indice iniziale *pos*. Estraendo da tali match i MEM si possono poi ottenere tutte altre occorrenze del medesimo MEM nel testo. Gli autori, per il calcolo delle MS, hanno usato, in MONI, anche il concetto di *threshold*, definito come il minimo valore dell'array LCP tra due run consecutive dello stesso carattere. Un ulteriore miglioramento si è registrato in *PHONI* con l'uso delle *LCE query* per il calcolo dell'array MS. Infatti, tale struttura, dati due indici del testo $i$ e $j$, restituisce il più lungo prefisso comune tra il suffisso $i$-esimo e il suffisso $j$-esimo.

Si può quindi apprezzare il collegamento naturale che si ha tra la PBWT e la BWT, avendo, ad esempio, che il prefix array della prima corrisponde al suffix array della seconda mentre il divergence array è una diversa rappresentazione dell'array LCP. Anche grazie a queste premesse, si è basato il progetto di tesi sulla costruzione di una versione run-length encoded della PBWT.

### Contributo

L'idea con la quale si è sviluppata questa tesi vede, come punto di partenza, alcuni primi risultati teorici, presenti in letteratura, che caratterizzavano una variante run-length encoded della PBWT. Purtroppo, tale proposta presentava alcune ridondanze nei dati memorizzati e non caratterizzava alcuna tecnica algoritmica per il calcolo degli SMEM. Per quanto l'intuizione iniziale fosse quella di adattare l'algoritmo 5 di Durbin all'uso con una struttura run-length encoded, non si è trovato un modo per memorizzare i dati del divergence array in modo proporzionale al numero di run. Inoltre non si è trovata una soluzione efficiente per utilizzare i sample del prefix array ad inizio e fine di ogni run.

Si è quindi deciso di cambiare approccio al problema, ispirandosi ai risultati già ottenuti per la RLBWT. Tale approccio si basa anche sull'uso dei bitvector sparsi, una struttura dati succinta che permette di memorizzare e interrogare vettori binari efficientemente. Quindi, le strutture necessarie all'indicizzazione delle run e al cosiddetto mapping, ovvero il "seguire" una riga dalla sua posizione permutata in una certa colonna alla posizione permutata in quella successiva nella matrice PBWT, sono state implementate tramite bitvector sparsi, con un numero di simboli $\sigma = 1$ proporzionale al numero run. Al fine di avvicinarsi alle idee proposte in *MONI* e *PHONI* è quindi servito uno studio teorico preliminare per ridefinire: matching statistics (e conseguente calcolo degli SMEM), threshold e LCE query. In particolare, grazie agli ultimi due concetti, si è potuto non memorizzare il divergence array. Parlando di matching statistics, si ha che sono definite tramite un array, lungo quanto il pattern e formato da coppie (riga *row*, lunghezza *len*), che, per ogni colonna, tiene traccia del suffisso comune che non sia estendibile a sinistra, terminate in quella colonna e lungo *len*, tra la riga *row* del pannello e l'aplotipo query. Una volta calcolato tale array è possibile estrarre uno SMEM, termi-

nate in colonna $k$ tra la query e la riga *row*, ed estendere, tramite una struttura dati a supporto, tale risultato a tutte le altre righe del pannello che presentano il medesimo SMEM. Il primo metodo di calcolo di tale array è stato basato sull'utilizzo delle threshold. Ragionando sulla *matrice PBWT*, una threshold è definita come il minimo valore dell'*array LCP* all'interno di una run (comprendendo anche la testa, qualora esistente, della run successiva, essendo il suo valore *LCP* calcolato sfruttando anche la coda della run corrente). L'insieme delle threshold per una certa colonna è stato memorizzato come bitvector sparso. Inoltre, si è provveduto a tenere in memoria i sample di prefix array a inizio e fine di ogni run.

Grazie all'uso delle threshold si è potuto sviluppare un algoritmo efficiente, dal punto di vista della memoria richiesta, per il calcolo delle matching statistics in due "sweep" sul pattern, calcolandone prima le posizioni e poi, tramite random access al pannello, anche le lunghezze. Tale soluzione richiedeva in memoria l'intero pannello, per il quale si è scelto di usare uno Straight-Line Program (SLP), ovvero una grammatica context-free che genera una e una sola parola, molto compatta in memoria, sulla quale è possibile effettuare random access (in tempo non costante). Ad esempio, un pannello $70000 \times 46538$, che normalmente richiede $\sim$ 3.3GB in memoria ($\sim$ 451MB se compresso tramite GZIP), viene memorizzato, tramite SLP, in $\sim$ 7MB.

Riprendendo quanto fatto in *PHONI* per la RLBWT, si è, poi, deciso di risparmiare ulteriore spazio eliminando l'uso delle *threshold*. Per ottenere il calcolo dell'array della matching statistics si sono quindi usate le LCE query, ridefinite in questo caso come il suffisso comune più lungo tra due righe del pannello, eventualmente fissando il termine a una colonna precisata. Il loro calcolo è permesso in modo efficiente dall'uso dell'SLP. Calcolando la lunghezza di tale suffisso comune, è possibile calcolare anche le lunghezze delle matching statistics contemporaneamente al calcolo delle righe della stessa. Quindi, si è ottenuto il calcolo completo di tale array in un singolo "sweep" sul pattern, passando da un tempo proporzionale a $\mathcal{O}(2N)$ a uno proporzionale a $\mathcal{O}(N)$. Si noti che il calcolo degli SMEM viene fatto contemporaneamente al calcolo delle lunghezze delle matching statistics quindi, con l'uso delle LCE query, si è ridotto sia lo spazio necessario in memoria per la RLPBWT che il tempo di calcolo degli SMEM.

### Risultati sperimentali

L'obiettivo della fase sperimentale è stato quello di confrontarsi con l'algoritmo 5 di Durbin, permettendo di verificare la riduzione della memoria necessaria al calcolo degli SMEM. L'importanza di ottenere dati quantitativi è anche dovuta al limite dato dalla non forte caratterizzazione asintotica, dal punto di vista della complessità temporale, delle operazioni con le strutture dati succinte, fattore che potrebbe comportare errori di valutazione. L'uso di tali strutture comporta, in ogni caso, un aumento atteso dei tempi di calcolo.

Per questa sintesi, si è scelto di riportare alcuni risultati relativi ad un pannello $70,000 \times 46,538$, che è stato interrogato con $30,000$ query, al fine di ridurre la fluttuazione statistica dei risultati. I risultati qui riportati, relativi alla PBWT, sono stati ottenuti usando l'implementazione ufficiale della stessa. Per la RLPBWT, invece, si sono usate

le LCE query.

In merito al tempo di calcolo si è ottenuto il risultato atteso avendo che con la PBWT si ottiene il calcolo degli SMEM in $\sim 411$s mentre con la RLPBWT in $\sim 1824$s, ovvero poco più del quadruplo del tempo. Si è passato quindi ad analizzare i risultati in termini di memoria. Secondo le stime di Durbin, gli array necessari al funzionamento dell'algoritmo 5 richiederebbero $\sim 13NM$ bytes, implicando, per il pannello in analisi, $\sim$ 40GB di memoria. Anche sperimentalmente si sono registrati picchi di memoria prossimi ai 40GB di memoria, confermando le stime. Invece, per la RLPBWT il picco registrato è stato di appena $\sim$ 3GB, richiedendo circa il 93% di memoria in meno rispetto alla soluzione di Durbin.

## Conclusioni

In conclusione, si rileva come i risultati prefissati siano stati raggiunti, producendo una struttura dati efficiente in memoria per la risoluzione del calcolo degli SMEM tra un pannello di aplotipi e un aplotipo esterno. Si segnala inoltre come siano possibili diversi sviluppi futuri, come un'ulteriore ottimizzazione della struttura, sia in termini di gestione del mapping che, eventualmente, di gestione di più query contemporaneamente. In merito a quest'ultimo si segnala, infatti, come Durbin abbia proposto anche un algoritmo per il calcolo degli SMEM tra un pannello di aplotipi e un pannello di query, fondendoli in un unico pannello su cui costruire la PBWT e calcolare i match interni al pannello stesso. Tale implementazione, nel setup di test descritto nella sezione precedente (che si noti comportare il perfetto input per tale soluzione), risulta essere molto performante sia in termini di tempo ($\sim 22$s) che di memoria utilizzata ($\sim 10,084$KB) durante l'esecuzione.

Inoltre, sono possibili diverse generalizzazioni rispetto alle caratteristiche del pannello. Per quanto i pannelli di aplotipi prodotti dal sequencing del genoma umano raramente presentino siti multi-allelici, si ha una presenza stimata, al momento, di circa il 2% di siti tri-allelici, avendo che tale percentuale risulti fortemente sottostimata. Una prima generalizzazione quindi sarebbe quella di studiare pannelli multi-allelici, riformulando la RLPBWT al fine di poter funzionare anche con pannelli costruiti su un alfabeto arbitrario. Un'altra generalizzazione interessante riguarda l'ammissione di dati mancanti all'interno del pannello stesso. La maggior parte delle soluzioni attualmente sviluppate sono basate su una forte assunzione: non si hanno dati mancanti. Una variante della RLPBWT che sia quindi in grado di lavorare, eventualmente con algoritmi parametrici o algoritmi approssimati, su pannelli in cui sono presenti wildcard per rappresentare tali dati, permetterebbe di fare studi più completi su dati reali, migliorandone gli usi, ad esempio, nel campo della medicina personalizzata.
Si segnala, infine, come la RLPBWT sia potenzialmente in grado di risolvere efficientemente anche altri task, come ad esempio la ricerca di k-SMEM, ovvero SMEM con un aplotipo esterno che coinvolgano esattamente $k$ righe nel pannello.
Le potenzialità di tale struttura sono quindi molteplici e, grazie al ridotto consumo di memoria, si hanno le giuste premesse perché venga utilizzata per gestire e interrogare grosse moli di dati reali, incrementando le capacità di studio, previsione e inferenza che si possono avere grazie allo studio del pangenoma.