Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di laurea in Informatica

Identificazione efficiente di esoni

Relatore: *Prof. Gianluca Della Vedova*

Correlatore: *Dott. Luca Denti*

Relazione della prova finale di:
Davide Cozzi
Matricola 829827

Anno Accademico 2019-2020

Abstract

Con la presente trattazione, ci si propone di descrivere quanto svolto durante l'esperienza trimestrale di stage presso l'Università degli Studi di Milano - Bicocca. Tale esperienza è stata svolta telematicamente in collaborazione con il laboratorio di Bioinformatica e Algoritmica Sperimentale*, chiamato* **BIAS***, del* Dipartimento di Informatica, Sistemistica e Comunicazione (**DISCo**).
Lo scopo del progetto era la ricerca efficiente di esoni non annotati, detti novel exons*, da integrare nel progetto open source denominato* ASGAL*, sviluppato dal gruppo di ricerca del laboratorio stesso, ovvero uno strumento atto all'identificazione di eventi di splicing alternativo, espressi tramite* ***RNA-Seq samples****, con l'ausilio dello studio dell'annotazione genomica.*

Indice

Capitolo 1

Introduzione

1.1 Accenni di biologia molecolare

1.1.1 DNA ed RNA

Prima di iniziare la trattazione più squisitamente computazionale è bene dare un'introduzione, dal punto di vista biologico, di quanto trattato.
Il **DNA**, sigla corrispondente ad **acido desossiribonucleico**, è un acido nucleico contenente le informazioni necessarie al corretto sviluppo di un essere vivente. Dal punto di vista chimico questa particolare macromolecola si presenta nella tipica **struttura a doppia elica**, formata da due lunghe catene di nucleotidi, dette **strand**. Nel dettaglio i singoli nucleotidi sono formati da un **gruppo fosfato**, dal **desossiribosio**, uno **zucchero pentoso,** e da una **base azotata**. Si hanno, inoltre, 4 tipi diversi di basi azotate:

1. **Adenina**, indicata con la lettera A
2. **Citosina**, indicata con la lettera C
3. **Guanina**, indicata con la lettera G
4. **Timina**, indicata con la lettera T

Si hanno quindi due **strand**, uno detto **forward strand** (indicato solitamente col simbolo "+") e uno detto **backward strand** (indicato solitamente col simbolo "−") che sono direzionati nel verso opposto (in termini tecnici si ha che il forward strand va da 5' UTR a 3' UTR, mentre il backward strand da 3' UTR a 5' UTR) e sono *appaiati* mediante coppie ben precise di basi azotate. Infatti, secondo il **modello di Watson-Crick** (figura 1.1), si ha che:

- l'**Adenina** si appaia con la **Timina** e viceversa

- la **Citosina** si appaia con la **Guanina** e viceversa

Questo accoppiamento permette di poter studiare i due **strand** come uno "complementare" all'altro. Infatti, conoscendo la sequenza di basi azotate di uno **strand**, è possibile ricavare la sequenza dell'altro mediante la tecnica del **Reverse&Complement** dove, preso uno strand, si converte ogni sua base secondo il seguente schema:

- le A diventano T
- le T diventano A
- le C diventano G
- le G diventano C

Esempio 1. *Vediamo, per completezza, un esempio di* ***Reverse&Complement****. Prendiamo una sequenza genomica* $S =$ "$TAGGCCATATGAC$" *e definiamo la funzione* $RC(x)$ *come la funzione che, presa in ingresso una stringa* x *costruita sull'alfabeto* $\Sigma = \{A, C, G, T\}$ *(quindi una sequenza genomica), restituisce la* ***Reverse&Complement*** *della stessa. Si ha quindi che:*

$$RC(S) = "ATCCGGTATACTG"$$

Per riferirci al **DNA**, contenuto in una data cellula di un essere vivente, usiamo il termine **genoma**, che a sua volta viene organizzato in diversi **cromosomi**. Si definisce **gene** una particolare regione di un **cromosoma** in grado di codificare una proteina.
Ai fini della trattazione del progetto, è necessario introdurre anche l'**RNA**, sigla corrispondente ad **acido ribonucleico** (avendo il **ribosio** come zucchero pentoso), ovvero una molecola, simile al **DNA**, dotata di una singola catena nucleotidica, sempre con 4 tipi di basi azotate (anche se si ha l'**Uracile**, che si indica con la lettera U, al posto della **Timina**). Tra i compiti dell'**RNA** si ha quello della codifica e decodifica dei **geni**.

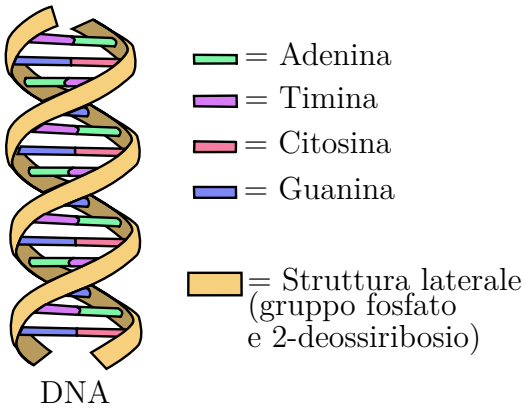


Figura 1.1: Rappresentazione grafica del DNA [1].

1.1.2 Esoni, Introni e Splicing alternativo

Per ottenere una **proteina** da un **gene** si hanno 3 passaggi (rappresentati nella figura 1.2):

1. La **trascrizione**, fase dove la sequenza del gene è copiata nel **pre-messenger RNA (pre-mRNA)**. Nel dettaglio viene selezionato uno dei due strand del gene e un enzima, chiamato **RNA Polimerasi**, procede alla trascrizione della sequenza selezionata creando il **pre-mRNA**. In questa fase la *Timina* viene sostituita dall'Uracile. È bene introdurre subito che in questo progetto non si terrà mai conto, a fini di semplificazione, del passaggio tra Timina e Uracile in quanto verrà usata sempre la *Timina*.

2. Lo **splicing**, fase dove vengono rimosse le parti non codificanti dalla molecola di **pre-mRNA**, formando il **messenger RNA (mRNA)**, detto anche **trascritto**. Per poter trattare al meglio questa fase bisogna parlare in primis di **esoni** e **introni**. In prima analisi si potrebbe dire, peccando di precisione, che gli **esoni** sono le sezioni codificanti di un gene mentre gli **introni** sono le porzioni non codificanti. Solo gli esoni formano il trascritto. Si ha, inoltre, che le prime due basi di un introne sono dette $5'$, nell'uomo solitamente si ha la coppia GT, mentre le ultime due, solitamente AG nell'uomo, sono dette $3'$ e sono meglio identificate come **siti di taglio (splice sites)**. Quindi un esone, in realtà, non coincide esattamente con una regione codificante, detta **CDS**, a causa di queste particolari coppie di basi. Si notifica però che, come spesso accade, i termini vengono usati in sovrapposizione.

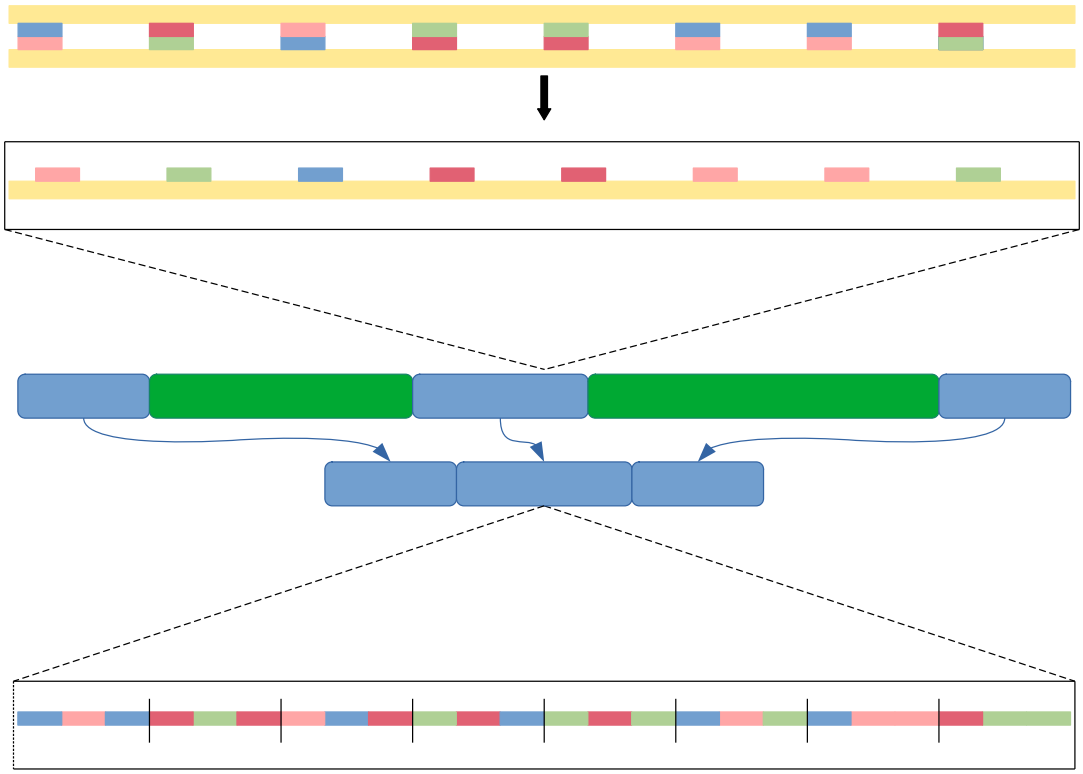


Figura 1.2: Rappresentazione grafica delle tre fasi per la sintesi di una proteina. Abbiamo all'inizio un ingrandimento al fine di poter visualizzare la *trascrizione* con la produzione di *pre-mRNA* a partire dal *DNA*. Seguono poi, allontanando nel disegno il "campo visivo", un evento di splicing "regolare" (dove semplicemente vengono rimossi gli introni, in verde) e l'ottenimento del trascritto. Infine, riavvicinando nuovamente la "visuale", si ha un ipotetico pezzo di proteina, ottenuto con la *traduzione* di tale trascritto, dove vengono segnalati anche i codoni.

3. La **traduzione**, fase dove viene effettivamente codificata la proteina a partire da una sezione dell'**m-RNA**. Bisogna quindi nominare particolari sequenze nucleotidiche di cardinalità 3: i **codoni**. Tali triplette sono tradotte in amminoacidi che, concatenati, formano le proteine. Esistono particolari codoni che sono utili al fine di riconoscere l'inizio e la fine della *sintesi proteica*. In particolare si ha un codone d'inizio, detto **start codon**, che solitamente corrisponde alla tripletta *AUG*, mentre, per il codone di fine, detto **stop codon**, solitamente si ha una tripletta tra *UAA*, *UAG* e *UGA*.

Ai fini di questo progetto ci si concentra sullo **splicing**.

In realtà, un gene è in grado di sintetizzare più di una proteina mediante il cosiddetto **splicing alternativo**, che consiste in diverse varianti dell'evento di splicing al fine di ottenere diversi trascritti. Si descrivono le principali modalità di splicing alternativo, visualizzabili nella figura 1.3:

- L'**exon skipping**, ovvero *salto dell'esone*, dove un esone (o anche più esoni) può essere escluso dal trascritto primario oppure dove un nuovo esone (o più nuovi esoni) può essere incluso nello stesso.
- L'**alternative acceptor site**, ovvero *sito di taglio alternativo* $3'$, dove una parte del secondo esone può essere considerata non codificante o, alternativamente, una porzione dell'introne adiacente può essere considerata codificante.
- L'**alternative donor site**, ovvero *sito di taglio alternativo* $5'$, dove una parte del primo esone viene considerata non codificante o, alternativamente, una porzione di introne adiacente può essere considerata codificante.
- I **mutually exclusive exons**, ovvero *esoni mutuamente esclusivi*, dove solo uno di due esoni viene conservato nel trascritto.
- L'**intron retention**, ovvero *introne trattenuto*, dove un certo introne viene incluso nel trascritto primario.

Le varie modalità di splicing alternativo non si escludono a vicenda, rendendo lo studio di tale fenomeno assai complesso.
Il progetto **ASGAL** [2] [3] (***A****lternative* ***S****plicing* ***G****raph* ***AL****igner*) è nato con lo scopo di studiare, dal punto di vista computazionale, tali eventi e il progetto descritto in questa trattazione si occupa di studiare eventi che considerino porzioni teoricamente non codificanti come codificanti.

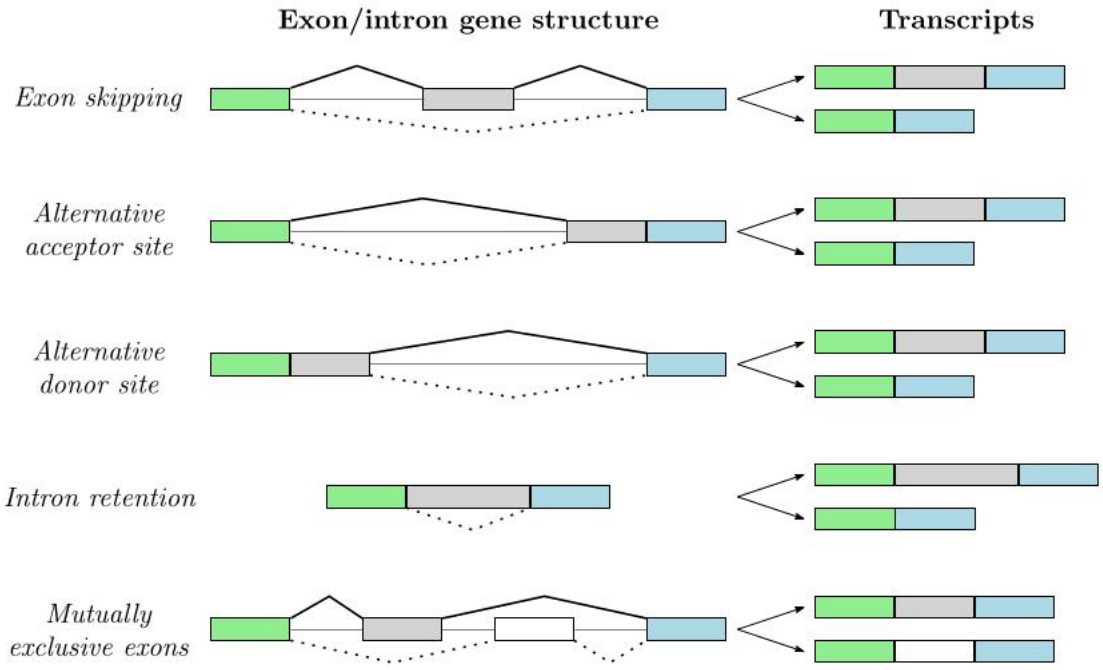


Figura 1.3: Schema riassuntivo per gli eventi di splicing alternativo [2].

Capitolo 2

Preliminari

Passiamo ora alla spiegazione di alcuni concetti e di alcune nozioni necessarie a comprendere al meglio la natura del progetto.

2.1 Formato dei files

Nel campo della bioinformatica vengono usati particolari formati di files di testo per la memorizzazione delle più svariate nozioni necessarie in questo ambito. In questa sezione si andranno quindi ad analizzare, in base agli scopi, i formati usati durante il progetto.

2.1.1 Genoma e RNA-Seq sample

Come abbiamo visto una sequenze nucleotidica può essere facilmente rappresentata mediante una stringa creata a partire dall'alfabeto $\Sigma = \{A, C, G, T\}$, dove ogni lettera corrisponde all'iniziale di una delle 4 basi azotate del DNA.
Per poter memorizzare una sequenza nucleotidica viene usato un file **FASTA** [4] (solitamente indicato con l'estensione `.fa`), il cui contenuto è piuttosto minimale in quanto conta essenzialmente di due sole parti:

1. un *header*, specificato all'inizio dal carattere ">", contenente un'indicatore univoco per identificare la sequenza nucleotidica ed eventualmente informazioni aggiuntive

2. una *stringa* rappresentante la sequenza nucleotidica

In questo progetto il genoma di riferimento, per esempio quello relativo ad un certo cromosoma, è contenuto in un file FASTA.
In un file FASTA è anche possibile memorizzare più di una sequenza nucleotidica,

concatenando headers e stringhe. In questo progetto le reads da allineare (il nostro **RNA-Seq sample**) sono anch'esse contenute in un file FASTA.

Esempio 2. *Vediamo un esempio relativo ad una porzione di un file FASTA con due sequenze nucleotidiche:*

```
>FBtr0070103_e_5224_X_289258
TGTTTCTACCAACCTTATCACGCTTTGGCTTATCCTCGGCATTGTCGTCCTTCATAAACG
AGGGCTGCAGGCTGAATCGGCGGCGTAAGTGGCGGCCACG
>FBtr0070103_e_2113_X_289177
ACGTTTTTCCCACTTTTGCAGAGCCCATGACGACGATTTTGTGGCGTGCGTTGGCCGGTC
CAATGGCATCATCTACCGCGTTGTTTCTACCAACCTTATC
```

2.1.2 Annotazione

Come abbiamo discusso nell'introduzione, la struttura di un gene è alquanto complessa e viene rappresentata mediante un insieme di **features**, detto *annotazione*. Questo insieme di features viene spesso memorizzato all'interno di un file **GTF** [5] (solitamente indicato con l'estensione `.gtf`). Ogni singola feature viene rappresentata da 9 campi, delimitati da **tabulazioni** ("\t"):

1. l'identificativo del cromosoma in cui è stata localizzata la feature
2. l'identificativo della sorgente da cui proviene l'annotazione, tipicamente il nome della banca dati o del software utilizzato
3. il tipo di feature, per esempio *gene, transcript, exon, start codon, stop codon, CDS...*
4. la coordinata di partenza della feature nel genoma di riferimento
5. la coordinata di fine della feature nel genoma di riferimento
6. il grado di confidenza relativo all'esistenza della feature e alle sue coordinate. Qualora il dato non fosse disponibile si avrà un "." al suo posto
7. lo strand della feature, indicato con "+" o "−"
8. il frame della feature, indicante quale base della feature stessa è la prima base del codone. Qualora il dato non fosse disponibile si avrà un "." al suo posto
9. una lista, opzionale, di informazioni aggiuntive, tra loro separate dal carattere ";"

Esempio 3. *Vediamo un esempio di feature, relativa a un esone, estratta da un file GTF (per comodità sono state omesse le informazioni aggiuntive):*

```
X FlyBase exon 287329 287526 . - . informazioni divise da ";"
```

2.1.3 Allineamenti

Le sequenze nucleotidiche (nel nostro caso le sequenze contenute nelle reads) allineate a un genoma di riferimento vengono memorizzate in un file **SAM (*Sequence Alignment Map*)** [6] (solitamente indicato con l'estensione `.sam`). La struttura di questo formato è complessa e conta di un header, contenente alcune informazioni relative al formato del file stesso e alle sequenze, e di una sezione dedicata all'allineamento, che viene a sua volta divisa in 11 parti, tra cui l'identificatore della read e del genoma di riferimento, le coordinate di inizio allineamento, la stringa CIGAR e la sequenza della read. Di queste parti quella che più ci interessa è la stringa **CIGAR (*Concise Idiosyncratic Gapped Alignment Report*)** che consiste in una rappresentazione compressa dell'allineamento. In una CIGAR si hanno coppie concatenate di numeri interi e operatori, dove l'operatore, nel dettaglio un carattere, rappresenta un determinato evento mentre l'intero che lo precede rappresenta il numero di occorrenze di tale evento. Tra gli operatori principali ricordiamo:

- "M" per i match
- "D" per le rimozioni
- "I" per gli inserimenti
- "S" per quelle parti che non compaiono nell'allineamento

Esempio 4. *Vediamo l'esempio di una riga di un file SAM rappresentante un match perfetto, essendo la read lunga 100:*

```
FBtr0342963_e_896_X_289053 16 X 289054 255 100M * 0 0
GCAGTGTCCAAAATATCAAGGGTAAGGCTCACGCTGGCGATGGAAAAGTTGCCTTGGTGCA
TTTCCTCAATGGTCCGCTTGTACTTGGTGCTAAATGTAT * NM:i:1
```

Una rappresentazione binaria delle informazioni di un file **SAM** può essere contenuta in un file **BAM (*Binary Alignment Map*)** [6]

2.2 Splicing graph

La prima struttura dati che andiamo a descrivere è lo **splicing graph** [2].
Innanzitutto, riprendendo i concetti di biologia molecolare descritti all'inizio, bisogna formalizzare in termini più matematici e computazionali il concetto di **trascritto**. Possiamo quindi dire che un trascritto T è una sequenza di coppie $[b_i, e_i]$ dove ciascuna coppia rappresenta un determinato esone, mediante le coordinate di inizio e fine nel genoma di riferimento. Si ha quindi che:

$$T = \langle [b_1, e_1], [b_2, e_2], \ldots, [b_n, e_n] \rangle$$

Sfruttando la definizione computazionale di esone possiamo definire anche quella di introne. Sapendo infatti che l'i-esimo esone è rappresentato tramite $[b_i, e_i]$, possiamo definire anche l'introne compreso tra l'i-esimo esone e il suo successore come:

$$[e_i + 1, b_{i+1} - 1]$$

Possiamo, quindi, definire lo **splicing graph** di un gene come il **grafo diretto aciclico *(DAG)*** $S_G = (\varepsilon_G, E)$, con ε_G insieme degli esoni e E insieme degli archi che connettono i nodi relativi a due esoni consecutivi in almeno un trascritto. In aggiunta vengono collegati, mediante archi diversi da quelli di E, i nodi relativi a due esoni che non sono consecutivi in alcun trascritto tali per cui il primo sia completamente a sinistra rispetto al secondo nel genoma di riferimento. Questi particolari archi sono chiamati **novel edges** (figura 2.1). In termini computazionali, nella *matrice di adiacenza* del grafo, si avrà il valore 2 per tali archi.

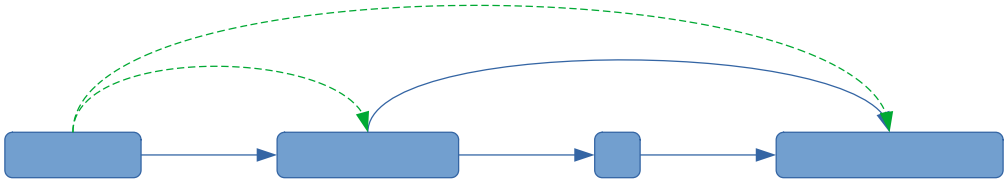


Figura 2.1: Esempio di splicing graph su 4 esoni (in cui si ipotizza un trascritto dove è assente l'esone 3), con gli archi in blu e i novel edges in verde tratteggiato.

2.2.1 Linearizzazione dello splicing graph

Spostiamo quindi l'attenzione sulle sequenze nucleotidiche, individuabili nel genoma di riferimento tramite le coordinate degli esoni che etichettano ogni nodo dello **splicing graph**.
Definiamo la **linearizzazione dello splicing graph** come la concatenazione delle etichette dei singoli nodi, precedute da un carattere sentinella, che, in primis

per motivazioni computazionali di rappresentabilità nel codice, viene identificato mediante il carattere "|". Tale carattere viene aggiunto anche alla fine della linearizzazione, che, per comodità, chiamiamo Z.

Esempio 5. *Ipotizziamo di avere 3 esoni consecutivi etichettati TAT, GCGCT e TGA. Si ha che la loro linearizzazione, usando il carattere "|" come carattere sentinella, è:*

$$|TAT|GCGCT|TGA|$$

Per poter comprendere l'utilità della linearizzazione dobbiamo prima introdurre un ulteriore concetto, quello di *bit vector*.

2.3 Bit vector

Un **bit vector** è un array, di lunghezza definita, formato da elementi binari. Chiamiamo B il bit vector di lunghezza n. Si ha che:

$$B[i] \in \{0, 1\}, \forall\, i = 0, \ldots, n-1$$

Questa particolare struttura dati permette due operazioni estremamente utili, la **rank** e la **select**. Un'implementazione *naive* di queste funzioni richiederebbe un tempo $O(n)$ ma, con l'ausilio di $o(n)$ bits aggiuntivi in memoria, si riesce a raggiungere una complessità temporale costante $O(1)$.

2.3.1 Rank

La prima operazione è la **rank** che permette di calcolare, in un *bit vector* di n elementi, il numero di occorrenze di 1 tra la posizione 0 e una posizione x passata come argomento alla funzione, quest'ultima esclusa. Formalmente si ha:

$$\texttt{rank}(x) = \sum_{i=0}^{x-1} B[i],\ 0 \leq x < n$$

2.3.2 Select

La seconda operazione è la **select** che permette di calcolare, in un *bit vector* di n elementi, l'indice dell'x-esimo 1. Formalmente si ha:

$$\texttt{select}(x) = \max\{j < n \mid\ \texttt{rank}(j) \leq x\},\ 0 < x \leq \texttt{rank}(n)$$

Esempio 6. *Ipotizziamo di avere il seguente bit vector B:*

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*
1	0	0	**1**	0	**1**	0	**1**	0	**1**	0	0	**1**	0

Si ha che, per esempio:

$$\texttt{rank}(6) = 3$$
$$\texttt{select}(5) = 9$$

2.3.3 Corrispondenza tra linearizzazione e bit vector

Tornando quindi alla linearizzazione Z (con carattere sentinella "|") dello **splicing graph**, notiamo che è possibile associare alla stessa un *bit vector* B, di pari lunghezza n, tale che:

$$\begin{cases} B[i] = 1 & \text{sse } Z[i] = \text{"|"} \\ B[i] = 0 & \text{altrimenti} \end{cases}$$

In questo modo, mediante l'operazione `rank`, si può calcolare l'*ID* dell'esone corrispondente ad un certo indice della linearizzazione mentre, grazie alla `select`, è possibile calcolare l'indice di partenza sulla linearizzazione di un determinato esone (passando alla `select` il suo *ID*). Si può, inoltre, fare una considerazione aggiuntiva sulla `select`: considerando la `select` di un esone e dell'esone che risulta essere rappresentato mediante il successore dell'*ID* del primo, è possibile calcolare l'indice di partenza e l'indice di fine dell'esone stesso nella linearizzazione, in modo da poterne estrarre la sequenza. Nel dettaglio bisogna considerare anche il carattere "sentinella" quindi, in realtà, per l'i-esimo esone si ha che il suo indice di partenza sulla linearizzazione è $\texttt{select}(i) + 1$ e quello di fine è $\texttt{select}(i+1) - 1$. L'utilità della `rank` verrà trattata più avanti nel testo.

Esempio 7. *Vediamo quindi un esempio di come viene mappato un bit vector per la linearizzazione:*

$$|AT|G|T|C|GC|A|$$

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*
1	0	0	**1**	0	**1**	0	**1**	0	**1**	0	0	**1**	0	1
	|	*A*	*T*	*|*	*G*	*|*	*T*	*|*	*C*	*|*	*G*	*C*	*|*	*A*	*|*

Possiamo quindi dire che, ipotizzando di aver calcolato la linearizzazione dello ***splicing graph****, per esempio, la base T, all'indice 6, appartiene al terzo esone. Inoltre, si ha che il quinto esone parte all'indice 9 (non contando il carattere sentinella dall'indice 10). Infatti, usando le due operazioni sopra descritte, si ha che:*

$$\texttt{rank}(6) = 3$$
$$\texttt{select}(5) = 9$$

2.4 Allineamento

Il passo successivo, nella pipeline di ASGAL, consiste nell'**allineamento** di ciascuna read dell'**RNA-Seq sample** con lo **splicing graph**. A tal fine bisogna però introdurre il concetto di **MEM**.

2.4.1 Maximal Exact Matches

Definizione 1. *Un **Maximal Exact Match (MEM)** [7] è definito come la più lunga sottostringa comune a due stringhe tale che non possa essere estesa in alcuna direzione senza aggiungere un mismatch.*
*Date due stringhe T e P, rispettivamente di lunghezza N e M, un **MEM** tra queste due stringhe è definito dalla tripla:*

$$m = (t, p, l)$$

dove:

- ***t** è l'indice di partenza del **MEM** sulla stringa T, quindi si ha che $0 \leq t < N$*
- ***p** è l'indice di partenza del **MEM** sulla stringa P, quindi si ha che $0 \leq p < M$*
- ***l** è la lunghezza del **MEM***

Esempio 8. *Vediamo un esempio chiarificatore [2].*
*Siano $T = MISSISSIPPI$ e $P = MIPPISSI$, calcolando i **MEMs** di lunghezza $l \geq 2$, si ottiene un insieme di **MEMs** contenente:*

- $(0, 0, 2)$*, rappresentante la stringa MI*
- $(1, 4, 4)$ *e* $(4, 4, 4)$*, rappresentanti la stringa $ISSI$*
- $(7, 1, 4)$*, rappresentante la stringa $IPPI$*

Si può quindi facilmente notare come il calcolo dei **MEMs** sia fondamentale nello studio di allineamenti tra un genoma (o una sua sottoparte come nel nostro caso) e un insieme di reads.
Qualora si abbia un **MEM** "parziale" non estendibile solo a destra si parla di **Right MEM**.
Senza entrare nel dettaglio, possiamo dire che ASGAL basa il calcolo dei **MEMs** su un algoritmo [7] che riesce a calcolare l'insieme di **MEMs** (di lunghezza minima

arbitraria pari a L), tra due stringhe T e P, rispettivamente di lunghezza N e M, in:

$$O(M + z + k \cdot w)$$

dove:

- z è il numero di right **MEMs** di lunghezza $\geq L$
- k è il numero di **MEMs** di lunghezza $\geq L$
- w rappresenta il tempo di accesso all'*indice* della stringa T (in quanto l'algoritmo si basa sull'*indicizzazione* di tale stringa)

Procedendo con il calcolo dei **MEMs**, tra ogni singola read R e la **linearizzazione dello splicing graph** Z, si nota subito che ogni **MEM** può cadere all'interno di un solo nodo del grafo, ovvero all'interno di un solo esone, grazie al carattere "sentinella" presente solo nella linearizzazione.
Sfruttiamo ora quanto discusso in merito alla linearizzazione per studiare meglio l'allineamento. Si ha che il **MEM** $m = (i_Z, i_R, l)$ appartiene all'i-esimo nodo dello splicing graph, calcolabile tramite $\texttt{rank}(i_Z)$. Ai fini della trattazione è necessario anche stabilire una relazione d'ordine tra due **MEMs**. Ipotizzando quindi di aver calcolato anche $m' = (i'_Z, i'_R, l')$ viene stabilita una relazione d'ordine tra m e m' tale per cui m precede m' sulla read R sse:

$$i_R < i'_R \ \ \wedge \ \ i_R + l < i'_R + l'$$

e tale precedenza viene indicata con la notazione:

$$m \prec_R m'$$

Lo stesso ragionamento viene fatto su Z, ottenendo la relazione $\prec_Z$, ma solo se i due **MEMs** appartengono allo stesso nodo dello splicing graph, ovvero se $\texttt{rank}(i_Z) == \texttt{rank}(i'_Z)$.

2.4.2 MEMs graph

In ASGAL, si basa lo studio dei **MEMs**, calcolati tra la linearizzazione dello **splicing graph** Z e la read R, sullo studio di un grafo, orientato e pesato, detto **MEMs graph**. Chiamato M l'insieme dei **MEMs** calcolati si ha che il grafo è così definito:

$$G_M = (M, E_M)$$

dove, quindi, i vertici sono i **MEMs** stessi. Due nodi sono collegati da un arco sse, in primis, il nodo di partenza è etichettato da un **MEM** precedente, secondo la

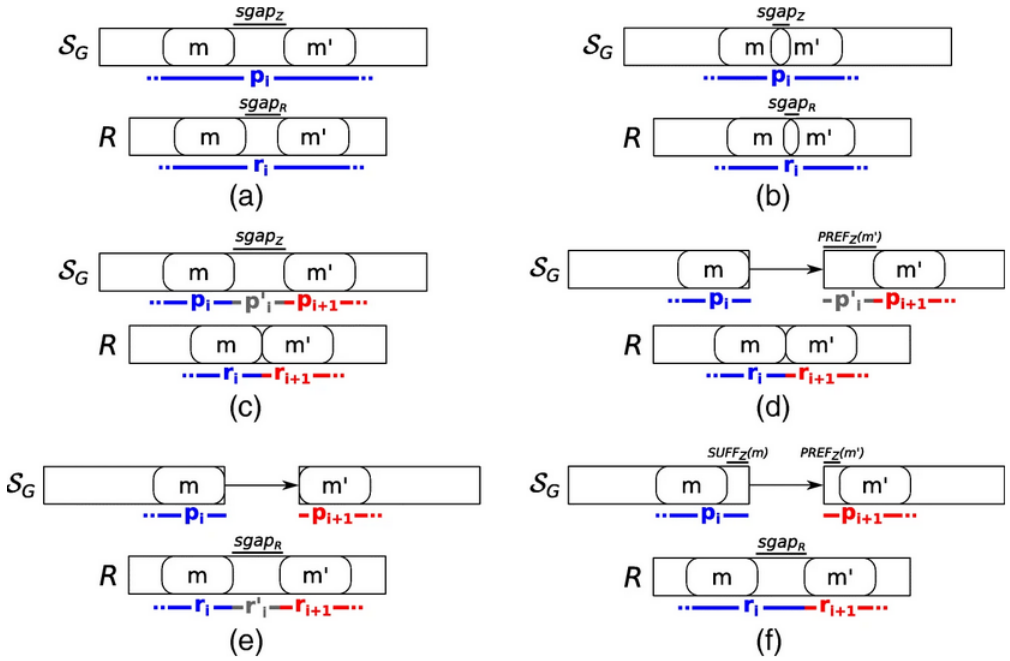


Figura 2.2: Rappresentazione grafica [2] delle possibili situazioni riscontrabili nello studio dei **MEMs**. Un arco tra il **MEM** m e il **MEM** m' viene aggiunto solo in presenza di determinate circostanze, a seconda che i due **MEMs** si trovino o meno sullo stesso esone, se abbiano o meno delle sovrapposizioni o se siano troppo distanti tra loro (analizzando i *gaps* sulla read e sulla linearizzazione, detti, rispettivamente, $sgap_R$ e $sgap_z$, nonché eventuali suffissi e prefissi delle sequenze esoniche). Tali circostanze sono analizzate, in modo approfondito, nello studio del dottor Luca Denti [2].

relazione $\prec_R$, a quello di arrivo. Oltre a questa condizione necessaria serve un'altra tra sei condizioni che possono essere visualizzate nella figura 2.2.
All'inizio della computazione si ha, inoltre, la presenza di due nodi, uno detto **nodo start** e uno detto **nodo end**, etichettati, rispettivamente, con il **MEM** $m = (0, 0, 0)$ e con il **MEM** $m = (-1, -1, -1)$. Tali nodi vengono usati come base per la costruzione del grafo stesso. I criteri verranno approfonditi più avanti nella trattazione.
Al fine di calcolare il vero e proprio allineamento della read viene quindi visitato il **MEMs graph** cercando, qualora siano calcolabili, i **cammini minimi** (con peso massimo arbitrario), mediante il famoso **algoritmo di Dijkstra**, tra il *nodo start* e il *nodo end*, ottenendo una o più liste di **MEMs** rappresentanti l'allineamento.

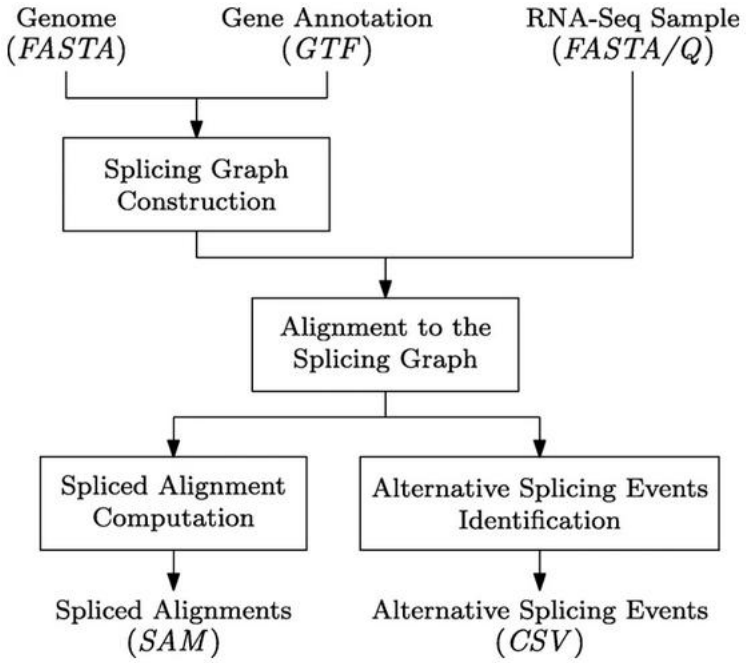


Figura 2.3: Rappresentazione schematica della pipeline di ASGAL [2]

2.5 Pipeline di ASGAL

Per concludere il capitolo è bene schematizzare la pipeline completa di ASGAL. Analizzando lo schema, rappresentato in figura 2.3, si ha, innanzitutto, la costruzione dello **splicing graph** a partire dal genoma di riferimento e dall'annotazione relativa a un singolo gene dello stesso. Durante questa fase viene prodotto anche un particolare file, di un formato custom, detto `.gtf.sg`, contenente informazioni sullo **splicing graph**. Avremo quindi all'interno di tale file:

- indicazioni sulla provenienza del genoma, sulla sua lunghezza ed eventualmente sullo strand
- la stringa relativa alla linearizzazione calcolata tramite lo **splicing graph**
- la matrice di adiacenza relativa allo **splicing graph** (dove con 1 vengono indicati gli archi *normali* mentre con 2 vengono indicati i *novel edges*)
- la lista contenente le posizioni di inizio e fine degli esoni, ordinati secondo l'ordine di aggiunta allo **splicing graph**. Si ha, quindi, il medesimo ordine degli *ID* utilizzabili con le operazioni di `rank` e `select` in quanto

tale ordine corrisponde anche all'ordine con cui sono state concatenate le corrispondenti sequenze, per ottenere la **linearizzazione dello splicing graph**

Si procede quindi con l'allineamento dello **splicing graph** con le varie read presenti nell'**RNA-Seq sample**, tramite lo studio del **MEMs graph**, studiando, per ogni read, anche il suo **Reverse**&**Complement**. Anche in questo caso, il risultato di tale procedura viene salvato in un particolare file di formato custom con estensione `.mem` che conterrà, per ogni riga:

- lo strand su cui è stato calcolato l'allineamento
- il nome della read allineata
- il punteggio relativo alla qualità dell'allineamento, corrispondente al peso del cammino della lista di **MEMs** nel **MEMs graph**
- la lista dei **MEMs** rappresentanti l'allineamento con lo **splicing graph**
- l'intera sequenza della read

A questo punto, tramite diversi scripts, si procede alla costruzione del file **SAM** e al calcolo degli eventi di **splicing alternativo** (salvati in un file di formato **CSV** (*Comma-Separated Values*)).

Capitolo 3

Metodo

Si discute ora della vera e propria ricerca dei cosiddetti *novel exons*, ovvero potenziali esoni non presenti nell'annotazione, che vengono però allineati da determinate read dell'**RNA-Seq sample**. A tal fine, sono state effettuate diverse aggiunte alla *codebase* preesistente di ASGAL e quindi è bene, come prima cosa, analizzare le strumentazioni, ovvero i linguaggi di programmazione e le librerie principali, usate nel progetto.

3.1 Introduzione agli strumenti usati

3.1.1 Linguaggi e librerie

Navigando il codice sorgente di ASGAL, si nota subito come ci sia una divisione netta tra due sezioni:

1. Una prima sezione contenente codice scritto in *C++*, atto a calcolare il vero e proprio allineamento e alla produzione del file `.mem`, contenuto nella cartella `/src`.

2. Una seconda sezione contenente diversi scripts in *Python* atti a gestire l'intera pipeline e a studiarne i risultati (studiando in primis il file `.mem`). Gli scripts in *Python* sono contenuti nella cartella `/script`.

Ai fini del calcolo dei *novel exons* non sono stati aggiunti nuovi sorgenti per la parte in *C++*, in quanto sono state effettuate solo aggiunte e modifiche all'interno della *codebase*. Si ha infine uno script in *Python*, denominato `asgal`, che gestisce l'intero processo di calcolo, concatenando i codici in *C++* e i vari scripts in *Python*. Bisogna, quindi, citare una serie di librerie esterne utilizzate in diverse parti del codice *C++*:

- La libreria **LEMON (*Library for Efficient Modeling and Optimization in Network*)** [8], che fornisce tutte le strumentazioni necessarie a lavorare con i grafi, a partire da classi per gestire grafi orientati e pesati fino a metodi per studiarne il cammino minimo, mediante algoritmi, come quello di **Dijkstra** citato precedentemente. Questa libreria, scritta in *C++*, viene usata per gestire il **MEMs graph**, per la sua creazione, per la sua estensione e per il calcolo dei cammini minimi.

- La libreria **Klib** [9], che fornisce, in una delle sue svariate componenti indipendenti tra loro, metodi efficienti per il *parsing* e per l'accesso a files in formato *FASTA*. La componente principale usata nella *codebase* è `kseq.h`. Questa libreria, scritta in *C*, viene utilizzata per gestire i due files contenenti il genoma di riferimento e l'**RNA-Seq sample**.

- La libreria **Sdsl-lite** [10], che fornisce l'implementazione di diverse **strutture dati succinte**, tra cui una struttura dati per il *bit vector*, comprensiva delle due operazioni `rank` e `select`, implementate nella libreria per essere estremamente efficienti.

Per quanto riguarda il codice *Python* abbiamo invece un uso frequente di due librerie esterne:

- **Biopython** [11] per la lettura e il *parsing* dei files *FASTA*

- **Gffutils** [12] per la lettura, il *parsing* e la manipolazione delle annotazioni contenute nei files *GTF*

3.1.2 Snakemake

Un altro strumento utilizzato in questo progetto, durante la fase di sperimentazione, è stato **Snakemake** [13]. Questo tool, basato su *Python*, permette di gestire grandi quantità di dati all'interno di un determinato *workflow*. A tal fine vengono definite delle **rules** dove si possono gestire:

- input, output ed eventuali parametri

- scripts in *Python* che vengono integrati all'interno del *workflow*, di modo che possano ricevere in ingresso direttamente input e parametri della *rule*

- comandi *Bash* che possono anch'essi gestire input e parametri della *rule*

- opzioni aggiuntive come il calcolo dei *benchmarks*

Inoltre, all'interno del *workflow*, è possibile gestire variabili, *path* di sistema, *checkpoints* e *wildcards*, rendendo possibile la gestione di molte informazioni e di molti files, semplicemente scrivendo un breve file, chiamato `Snakefile`, che contiene tutte le regole e i dati necessari alla gestione del *workflow*. Un altro aspetto interessante è la possibilità di inserire parametri all'interno di un file *YAML*. All'interno del file `.yaml` è quindi possibile definire particolari parametri che verranno poi utilizzati all'interno dello `Snakefile`, che quindi non verrà modificato nel caso in cui si debbano cambiare solo alcune informazioni essenziali (per esempio le cartelle dove porre i risultati o gli indirizzi dei server da cui prelevare i dati). In merito a questi aspetti entreremo nel dettaglio più avanti nella trattazione.

3.2 Riconoscimento degli Introni

Iniziamo quindi a descrivere le aggiunte vere e proprie fatte ad ASGAL al fine di riconoscere *novel exons* all'interno di sezioni introniche del genoma.
Il primo passo è stato quello di riconoscere effettivamente gli introni all'interno dell'annotazione. Ricordando che l'analisi degli esoni in ASGAL non tiene in considerazione il trascritto di provenienza degli stessi, un primo approccio *naive* è stato quello di ordinare gli esoni e prendere le coppie complementari, ovvero data una sequenza di coppie ordinate:

$$(b_1, e_1)\ (b_2, e_2)\ \ldots\ (b_n, e_n)$$

dove con e_i si indica l'indice di partenza dell'esone e con b_i l'indice di fine, si genera la lista delle coppie complementari:

$$(e_1 + 1, b_2 - 1)\ (e_2 + 1, b_3 - 1)\ \ldots\ (e_{n-1} + 1, b_n - 1)$$

che rappresentano, in prima istanza, i nostri introni.
Al fine di uno studio più preciso si è deciso però di ricorrere ad una rappresentazione più complessa degli introni. Si procede comunque al calcolo delle sequenze complementari le quali vengono aggiunte, mediante la coppia di indici, a un `std::set` (che ricordiamo contenere un insieme di elementi unici, quindi diversi tra loro). A tale *set* vengono aggiunte anche tutte le coppie di indici relative agli esoni non presenti in tutti i trascritti. Questa operazione permette di avere tutti i *possibili introni* (eventualmente separati in diverse sequenze) che possono esistere tra due esoni, tenendo conto del trascritto di provenienza. In altri termini qualora un esone fosse presente in un trascritto, ma non in un altro, esso verrà considerato come sequenza intronica (quindi come potenziale "zona" dove cercare *novel exons*). Le coppie di indici presenti all'interno del *set*, che chiameremo *intron-set*, vengono, tramite una `std::map`, associate ad un *ID* crescente da 1 alla cardinalità del *set* e

le sequenze genomiche a esse associate vengono aggiunte a una seconda linearizzazione, costruita appositamente per gli introni, usando come carattere sentinella il carattere "|". In tal modo, dopo aver costruito il *bit vector* associato alla linearizzazione degli introni, si potranno effettuare anche in questo caso le operazioni di `rank` e `select`.
Si procede quindi iterando su tutti gli esoni di ogni trascritto al fine di identificare, per ogni coppia di esoni consecutiva in tale trascritto, quali coppie di indici, relative ad un introne, presenti in *intron-set*, hanno un indice di partenza maggiore o uguale all'indice di fine del primo esone e un indice di fine minore o uguale all'indice di inizio del secondo esone. In altre parole, per ogni coppia di esoni di ogni trascritto, creiamo una `std::map` che associa il nome del trascritto e la coppia di esoni (rappresentati tramite gli indici) a un `std::vector` contenente tutte le coppie di indici rappresentanti una sequenza intronica contenuta tra i due esoni.
Ai fini dello studio, la distinzione dei vari trascritti è superflua quindi tale mappa subisce un'operazione di *folding* al fine di ottenere una mappa finale dove non si avrà più distinzione tra i vari trascritti e, per ogni coppia di esoni, si avranno tutte le possibili coppie di indici, relative a un introne, che possono essere tra loro contenute. Inoltre, sia le coppie di indici relative agli esoni che le coppie di indici relative a un introne, in un ultimo passaggio, vengono rappresentate tramite l'*ID* che è associato a ogni coppia di indici (tramite le due mappe relative agli *ID*), al fine di poter sfruttare l'accesso veloce alle informazioni tramite le operazioni `rank` e `select` sui due *bit vectors* associati alle due linearizzazioni.

Esempio 9. *Vediamo un esempio per spiegare meglio quanto appena descritto.*
In questo esempio si assume di rappresentare esoni e introni, fin da subito, tramite gli *ID* associati e non tramite gli indici di inizio e fine sul genoma di riferimento.
Partiamo innanzitutto dagli esoni annotati a cui viene assegnato, da ASGAL, uno specifico ID. Prendendo un'annotazione d'esempio con 3 trascritti si hanno quindi i seguenti esoni, identificati con il loro ID:

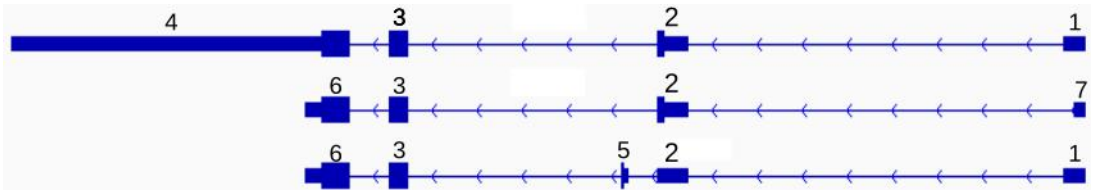


Figura 3.1: Esempio di 3 trascritti, visualizzati con IGV, dove vengono segnalati gli *ID* degli esoni.

Analogamente possiamo identificare anche gli introni seguendo la tecnica descritta sopra. In questo caso gli ID vengono indicati in verde. Si noti come l'esone 5*, non essendo presente nei primi due trascritti, risulti anche come introne, il numero 4:*

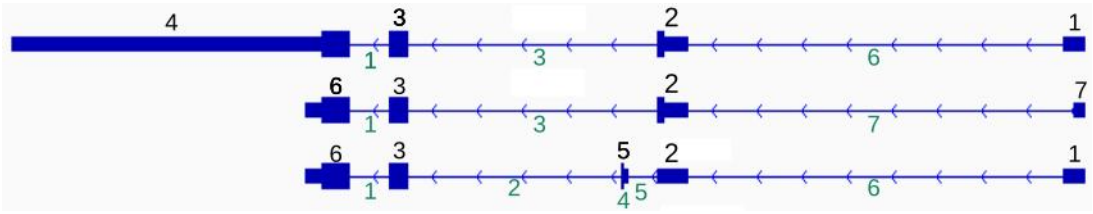


Figura 3.2: Esempio di 3 trascritti, visualizzati con IGV, dove vengono segnalati gli *ID* degli esoni e degli introni, rispettivamente in nero e verde.

Procedendo poi, non tenendo più conto della differenza di trascritto ma solo degli indici delle coppie di esoni, si ottiene la mappa degli introni. Si ha quindi, a sinistra tra parentesi tonde, la coppia di ID dei due esoni e a destra, tra parentesi quadre, la lista degli ID degli introni in essi contenuti in base agli indici di inizio e fine:

$$(2,\ 1):\ [6]$$
$$(2,\ 7):\ [6,\ 7]$$
$$(3,\ 2):\ [2,\ 3,\ 4,\ 5]$$
$$(3,\ 5):\ [2]$$
$$(4,\ 3):\ [1]$$
$$(5,\ 2):\ [5]$$
$$(6,\ 3):\ [1]$$

In termini di codice *C++* le coppie sono rappresentate tramite `std::pair` in modo da poter facilmente accedere sia al primo che al secondo elemento.
Analogamente a quanto fatto con gli esoni si procede al salvataggio su files, in formato custom `.intron.sg`, di alcune informazioni utili:

- le indicazioni sulla provenienza del genoma, sulla sua lunghezza ed eventualmente sullo strand
- la stringa relativa alla linearizzazione degli introni
- la lista contenente le posizioni di inizio e fine degli introni. Tale lista risulterà ordinata in modo crescente secondo l'indice di partenza di ogni coppia di introni

3.2.1 Differenziazione dei MEMs

Prima di poter procedere con l'effettiva estensione del **MEMs graph** è bene studiare come vengono rappresentati i **MEMs** in ASGAL. È stata definita quindi, a tal fine, una `struct` contenente:

- la tripla t, p e l che descrive il **MEM** stesso
- l'attributo $isNew$ che ci segnala, in fase di creazione del **MEMs graph**, se già esiste un nodo per il **MEM**
- i due tipi di nodo con cui rappresento il **MEM** nel **MEMs graph**
- i costruttori che costruiscono la struct a partire da una tripla di attributi
- un metodo di stampa per ottenere la tripla caratterizzante il **MEM**
- i due metodi *setter* per i due tipi di nodo possibili nel **MEMs graph**

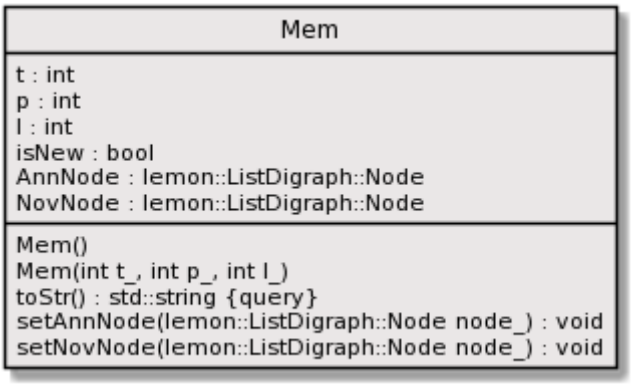


Figura 3.3: Diagramma della struct Mem nel codice originale.

Come già detto, l'algoritmo che permette il calcolo degli stessi non è argomento di questa trattazione ma, per poter usare lo stesso algoritmo sia per il calcolo dei **MEMs** tra la read e la linearizzazione dello **splicing graph** che tra la read e la linearizzazione delle sequenze introniche, è stato leggermente modificato il codice dello stesso per poter permettere la produzione di **MEMs** differenziati per esoni e introni. A tal fine, la *struct* è stata modificata aggiungendo semplicemente un valore `bool` che, quando posto `true`, ci segnala che si sta lavorando su un **MEM** proveniente dalle sequenze introniche. Parlando, come anticipato, dell'algoritmo atto al calcolo dei **MEMs**, chiamato *backwardMEM*, si segnala che è stato leggermente modificato il codice in modo da poter stabilire, tramite il costruttore,

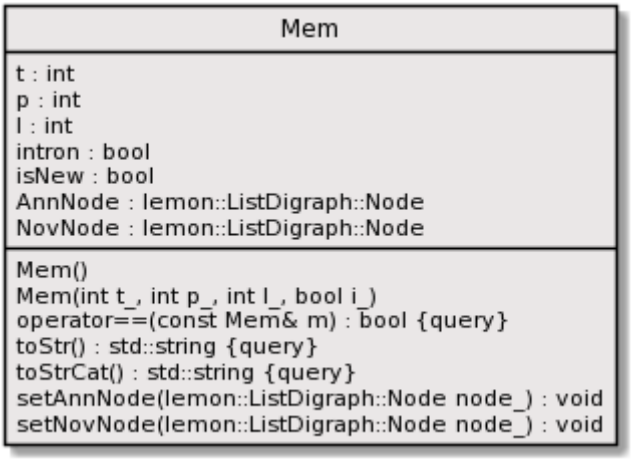


Figura 3.4: Diagramma della struct Mem nel codice modificato. Si nota anche l'aggiunta dell'overload dell'operatore di uguaglianza al fine di riconoscere **MEMs** uguali e di un metodo di stampa che comprende anche la tipologia del **MEM** stampato.

quale tipologia di **MEM** si sta calcolando. L'indicazione della tipologia di **MEM** (figura 3.4), specificata all'interno del costruttore della `struct` (nonché specificato nel costruttore della classe che si occupa del calcolo dei **MEMs**), risulterà essenziale per lo studio del file `.mem` al fine di verificare l'allineamento con esoni non annotati in quanto ora il **MEM** sarà identificato da una quadrupla, avendo anche la specifica *booleana* della tipologia:

$$m = (t, p, l, intron)$$

3.3 Costruzione del MEMs graph

È stata già descritta precedentemente la struttura del **MEMs graph** per cui trattiamo, velocemente, come esso viene costruito.
Introduciamo il parametro L, di default quantificato in 15, che rappresenta la lunghezza minima dei **MEMs** che verranno calcolati. Una volta ottenuta tale lista si procede alla creazione del grafo anche se, in realtà, si hanno due diversi grafi: il cosiddetto *AnnGraph* e il *NovGraph*. La particolarità di questo secondo è di tener conto di archi tra due **MEMs** indipendentemente dal fatto che sia stato calcolato un punteggio non compatibile con gli standard imposti a inizio computazione. Dopo aver calcolato la lista dei **MEMs** tra la read in analisi e la linearizzazione si procede analizzando, per ogni **MEM**, 3 possibili situazioni:

1. si verifica se esso possa essere collegato al nodo iniziale, tramite il metodo `validStart`

2. si verifica se esso possa essere collegato ad un qualsiasi altro **MEM** della lista, tramite il metodo `checkMEMs`

3. si verifica se esso possa essere collegato al nodo finale, tramite il metodo `validEnd`

Questi tre passaggi possono essere estesi, nel caso dell'*AnnGraph*, per considerare anche i **MEMs** provenienti dagli introni, al fine di ricercare possibili *novel exons*.

3.3.1 Lista dei MEMs intronici

Prima di iniziare a catalogare le varie possibili estensioni è necessario fare una breve premessa. La cardinalità della lista dei **MEMs** proveniente dagli introni è, statisticamente, maggiore, se non molto maggiore, di quella calcolata sugli esoni. La causa è la lunghezza della linearizzazione e le motivazioni possono essere riassunte in 2 punti:

1. Una motivazione "biologica" causata dall'oggettiva maggior lunghezza delle sequenze non codificanti rispetto a quelle codificanti. Questa motivazione è visualizzabile nella figura 3.5.

2. Una motivazione "computazionale" data dalla scelta di costruire la mappa degli introni descritta precedentemente. A causa di quella scelta, si avranno sequenze ripetute all'interno della linearizzazione in quanto una determinata sequenza non codificante può essere sia riconosciuta come intera che eventualmente divisa in più sotto-sequenze. Un esempio può essere riscontrato nella figura 3.2, dove l'introne 3 non è altro che la concatenazione degli introni 2, 4 e 5 (comportando che le sequenze relative a questi tre possibili introni saranno in realtà già presenti nella **linearizzazione**, implicando delle ripetizioni).

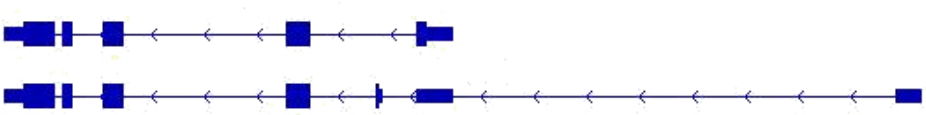


Figura 3.5: Visualizzazione di due trascritti dove è possibile notare gli esoni (rappresentati mediante rettangoli blu) e gli introni contenuti tra ogni coppia di esoni. Si nota come le sequenze introniche siano, statisticamente, più lunghe di quelle esoniche.

Avendo a che fare con una lista di **MEMs** molto estesa si avrà un maggior carico computazionale in alcuni punti della computazione e a causa di ciò sono state imposte particolari limitazioni che verranno analizzate.

3.3.2 Estensione del MEMs graph esonico

Una prima situazione in cui possiamo incorrere durante la creazione del **MEMs graph** è quella di aver ottenuto una lista di **MEMs**, proveniente dall'allineamento della read con la linearizzazione degli esoni, di cardinalità non nulla. Questo fatto ci porta, in primo luogo, a concludere che sicuramente la read in analisi sarà allineata con uno o più esoni già annotati e questi allineamenti devono avere la priorità sugli eventuali allineamenti calcolati sulla linearizzazione degli introni.
Un aspetto che non può essere trascurato è quello della selezione degli introni in cui può essere validato un allineamento. Sempre nell'ottica di dare priorità agli esoni si è scelto di considerare, eventualmente, solo allineamenti provenienti da un introne adiacente all'esone considerato (o contenuto in mezzo a due esoni nel caso in cui si stia valutando un *gap* sulla read, troppo esteso tra due **MEMs** provenienti da due esoni consecutivi). Inoltre, questa scelta permette di studiare, in primis, eventuali estensioni di esoni annotati, in termini di eventi di *alternative acceptor site* e *alternative donor site*, con sequenze introniche adiacenti.
Abbiamo quindi 3 possibili casi da analizzare, che analizzeremo singolarmente nel dettaglio:

1. Il **MEM** considerato presenta un valore p maggiore di L, implicando che un prefisso di read, pari almeno alla lunghezza minima su cui sono stati calcolati i **MEMs**, potrebbe non essere allineato con la linearizzazione degli esoni. Inoltre, l'algoritmo originale non creerebbe un arco tra il nodo *start* e il nodo del **MEM**.

2. La coppia di **MEMs** considerata presenta un *gap* sulla read, ovvero la distanza tra la fine del primo **MEM** e l'inizio del secondo, maggiore di L, implicando che potrebbe esserci una porzione intermedia della read non allineata con la linearizzazione degli esoni. Inoltre, l'algoritmo originale non creerebbe un arco tra i due nodi relativi ai **MEMs**.

3. Il **MEM** considerato presenta una copertura sulla read terminante prima della fine della read stessa, con un margine di lunghezza almeno pari a L. Questo fatto potrebbe implicare che un suffisso di read potrebbe non essere allineato con la linearizzazione degli esoni e il nodo relativo al **MEM** in analisi potrebbe non venir collegato al nodo *end*.

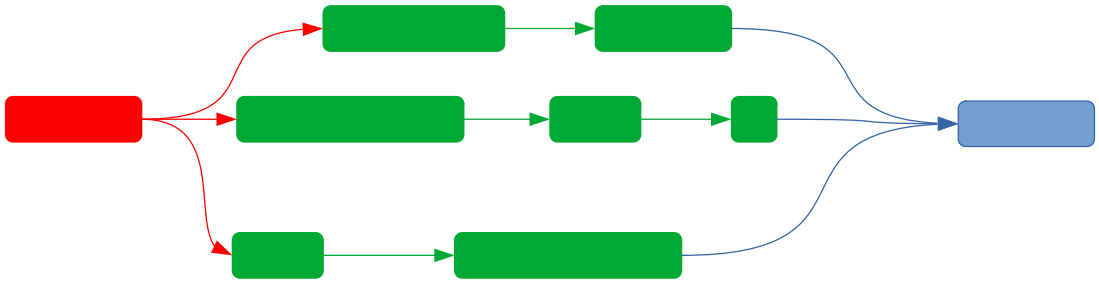


Figura 3.6: Rappresentazione grafica dell'estensione iniziale del **MEMs graph**, con il nodo *start* rappresentato in rosso, i nodi rappresentanti i **MEMs** delle molteplici possibili sequenze introniche in verde e il nodo rappresentante il **MEM** esonico in blu. In rosso sono rappresentati gli archi pesati tra il nodo *start* e i **MEMs** intronici.

Estensione iniziale

È stato già discusso precedentemente quando si necessita di estendere inizialmente il grafo, discutiamo quindi la modalità. Innanzitutto, vengono calcolati i **MEMs** intronici. Una volta calcolati si procede al controllo dell'introne di provenienza di ogni **MEM**, infatti, come già discusso, si studiano solo gli introni adiacenti all'esone da cui proviene il **MEM** che o non risulta essere idoneo per essere collegato al nodo *start* o risulta essere allineato con la read a partire da un indice troppo grande. Nel caso dell'estensione iniziale studiamo, quindi, l'introne che precede l'esone, ovvero, dato un esone $[b_i, e_i]$, studiamo tutti gli introni che, nella mappa precedentemente descritta, si trovano contenuti tra un esone arbitrario $[b, e]$ e l'esone $[b_i, e_i]$, con quest'ultimo posto a destra rispetto al primo. Vengono quindi selezionati solo i **MEMs** validi, e vengono calcolate le possibili sequenze (che saranno ordinate secondo il valore di p in modo crescente) di **MEMs** valide a colmare il *gap* iniziale.
Si procede quindi, tramite l'uso di *LEMON*, a creare archi tra **MEMs** intronici consecutivi in ognuna delle sequenze calcolate. Si procede poi a collegare, con un arco di peso simbolico pari a 1, l'ultimo nodo di ogni sequenza al nodo rappresentante il **MEM** esonico. Infine, si collega il nodo *start* al primo nodo di ogni sequenza di **MEMs** intronici, specificando come peso dell'arco il valore p del nodo intronico e segnalando quindi la cardinalità del prefisso di read eventualmente ancora non allineato di modo che si tenga considerazione della qualità dell'allineamento stesso in fase di ricerca dei cammini minimi. Quanto descritto è rappresentato nella figura 3.6.

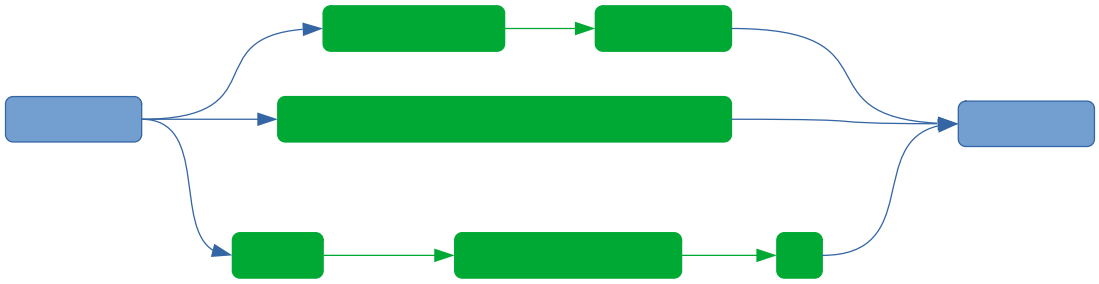


Figura 3.7: Rappresentazione grafica dell'estensione intermedia del **MEMs graph**, con i nodi dei due **MEMs** esonici rappresentati in blu e i nodi rappresentanti i **MEMs** delle molteplici possibili sequenze introniche, che possono colmare il gap sulla read dato dai **MEMs**, in verde.

Estensione intermedia

Analizziamo ora la situazione in cui, durante il tentativo di collegare il nodo relativo al **MEM** esonico in analisi con un altro **MEM** esonico della lista, si trova che il gap sulla read tra questi due **MEMs** è superiore a L. Si procede, quindi, cercando i **MEMs** intronici relativi alle sequenze di introni che, nella mappa precedentemente descritta, presentano come chiave la coppia di *ID* calcolata tramite `rank` sui due **MEMs** esonici in analisi. Anche in questo caso si ottengono diverse sequenze di **MEMs** intronici adeguati e si procede, tramite *LEMON*, per ogni sequenza, alla creazione di un nodo per ogni **MEM** e al collegamento di ogni nodo col successivo, sempre sfruttando l'ordine basato sul valore p di ogni **MEM**. Quindi viene collegato il primo **MEM** esonico con ogni primo nodo di ogni sequenza, con un arco di peso simbolico pari a 1, e ogni ultimo nodo di ogni sequenza viene collegato al nodo del secondo **MEM**, sempre con un arco di peso pari a 1 di modo che la ricerca dei cammini minimi veda il percorso con gli introni come un percorso con "peso" maggiore. Quanto descritto è rappresentato nella figura 3.7.

Estensione finale

Infine, analizziamo l'eventuale estensione finale del **MEMs graph**. Questo accade qualora non sia possibile collegare un **MEM** esonico al nodo *end*, avendo quindi un suffisso di read di lunghezza maggiore a L non allineato con la linearizzazione degli esoni. Anche in questo caso, si procede con il calcolo dei **MEMs** intronici ritenendo però validi solo quelli provenienti da un introne posto immediatamente a destra dell'esone studiato. Verranno quindi utilizzati gli introni di ogni lista, nella mappa precedentemente descritta, corrispondenti a una chiave dove il primo esone corrisponde all'esone studiato. Analogamente ai due casi precedenti, si procede a collegare, per ogni sequenza di **MEMs** intronici calcolata, tutti i nodi. Quindi

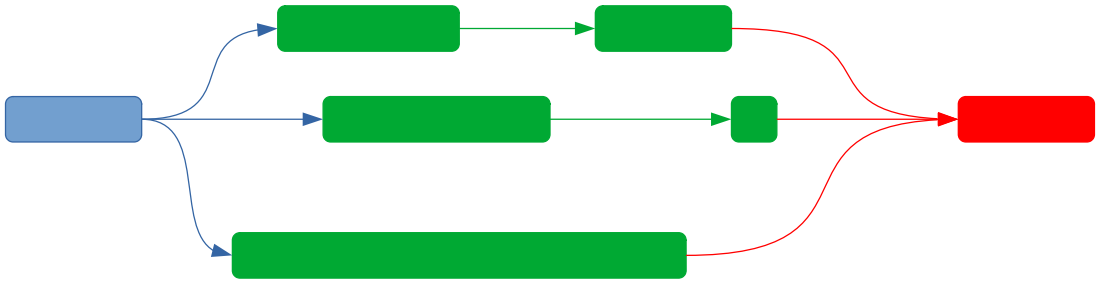


Figura 3.8: Rappresentazione grafica dell'estensione finale del **MEMs graph**, con il nodo rappresentante il **MEM** esonico in blu, i nodi rappresentanti i **MEMs** delle molteplici possibili sequenze introniche in verde e il nodo *end* rappresentato in rosso. Sempre in rosso sono rappresentati gli archi pesati tra i **MEMs** intronici e il nodo *end*.

viene collegato, con un arco di peso simbolico pari a 1, il nodo corrisponde al **MEM** esonico con il primo nodo di ogni sequenza di **MEMs** intronici. Infine, con peso pari alla eventuale lunghezza del suffisso della read non ancora allineato, si collega l'ultimo nodo di ogni sequenza con il nodo *end*. Nel dettaglio, tale peso w, avendo una read di lunghezza l e ultimo **MEM** di una sequenza intronica m, è dato da:

$$w = l - (m_p + m_l)$$

Quanto descritto è rappresentato nella figura 3.8.

3.3.3 Costruzione del MEMs graph intronico

Fino ad ora abbiamo trattato unicamente il caso in cui la lista dei **MEMs** esonici non fosse vuota ma non sempre possiamo ottenere questa situazione. Inoltre, potrebbe anche accadere che non esistano archi, nonostante il tentativo di estensione, che raggiungano il nodo *end* o che, alla fine della costruzione partendo dai **MEMs** esonici si arrivi a un grafo contenete solo due nodi, il nodo *start* e il nodo *end*. In tal caso viene costruito il **MEMs graph** in modo analogo a quanto fatto nel codice originale in merito alla costruzione del grafo in presenza di soli **MEMs** esonici. Si ha però qualche differenza. In primis non si ha la costruzione del *NovGraph*, ma solo dell'*AnnGraph*, come già indicato precedentemente nel caso dell'estensione del grafo. Vengono quindi costruiti tre metodi analoghi ai metodi `validStart`, `validEnd` e `checkMEMs` che sfruttano i metodi `rank` e `select` sugli introni. A causa della strategia di riconoscimento degli introni, non si avrà un equivalente dello **splicing graph** e quindi una lista di nodi *genitore* e di nodi *figlio* per ogni nodo del grafo ma verrà sfruttato l'ordine crescente degli introni al fine di sopperire a questa mancanza.

Come anticipato, la lunghezza della linearizzazione degli introni può risultare un grosso problema dal punto di vista computazionale. Si consideri di avere una grossa annotazione, con decine di trascritti ed esoni piccoli e molto distanti tra loro. A causa della lunghezza degli esoni si otterrà una linearizzazione molto lunga con una gran quantità di **MEMs** di piccola lunghezza (quindi in un intorno del valore L). In fase di costruzione del **MEMs graph**, in particolare durante il tentativo di studiare i collegamenti tra un **MEM** ed un altro appartenente alla lista, si potrebbe incorrere in computazioni non sostenibili in situazioni "normali" nonché spesso inutili. Si procede quindi con una scelta a priori in merito alla massima lunghezza della linearizzazione degli introni per la quale effettuare uno studio completo con valore L pari a quella degli esoni. Si è optato, in via iniziale, per una lunghezza massima pari a 500000 caratteri. Oltre tale lunghezza, nel caso di calcolo dei **MEMs** intronici, verrà utilizzato un nuovo valore di lunghezza minimo pari a metà della lunghezza della read che si cerca di allineare. A causa di questa scelta, in determinati casi, si potrebbero avere risultati di qualità peggiore, nonostante gli allineamenti più consistenti vengano comunque identificati permettendo quindi di identificare, con buona precisione, eventuali *novel exons*.
Bisogna specificare, inoltre, che allo stato attuale, a causa, in primis, di un mancato studio approfondito del peso degli archi in presenza di **MEMs** intronici, agli allineamenti contenenti introni è permesso un peso massimo maggiore di quello usato in presenza di soli **MEMs** esonici.
Tutte le estensioni ad ASGAL fino ad ora introdotte possono essere attivate tramite l'opzione `-i` *o l'opzione estesa* `--introns` *da specificare all'esecuzione dell'eseguibile nominato* `SpliceAwareAligner`.

3.4 Ultimi passaggi

Dopo la ricerca, per ogni read, di ogni allineamento, comprendente anche eventuali allineamenti con porzioni non codificanti del genoma, avremo i risultati salvati in un file `.mem`, la cui struttura è stata già descritta. Notiamo, però, che in questo caso i singoli **MEMs** non sono più rappresentati da una tripla ma da una quadrupla dove il quarto campo rappresenta la provenienza del **MEM**. Si avrà quindi il valore 0 se proviene da un esone annotato mentre si avrà il valore 1 se proviene da un introne. Questa notazione semplificherà l'uso di alcuni scripts che ora verranno descritti.

3.4.1 Creazione del file SAM

Riprendendo la pipeline di ASGAL notiamo come un primo risultato sia quello di ottenere un file **SAM** per rappresentare gli allineamenti. Nel progetto originale si

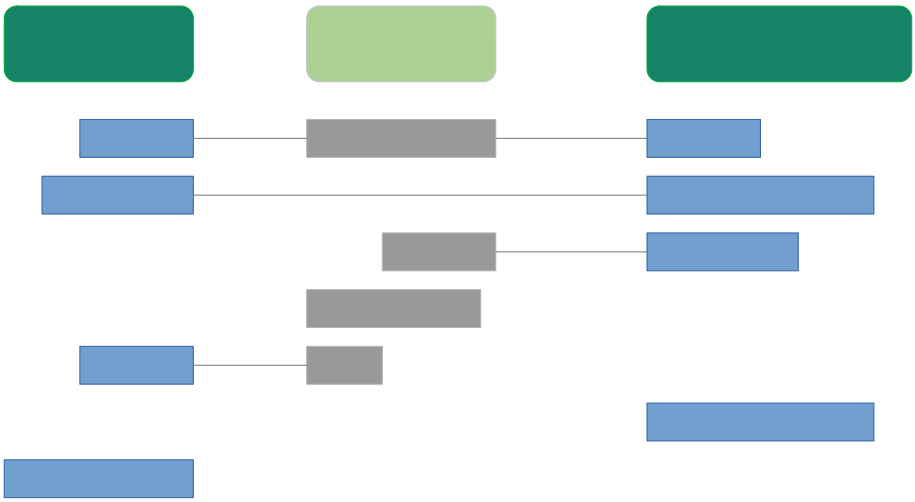


Figura 3.9: Rappresentazione grafica dei possibili allineamenti. In alto si hanno due esoni annotati, in verde, e un esone non annotato in verde chiaro. Si nota che tali esoni sono ordinati nel genoma di riferimento. L'esone non annotato è quindi contenuto nell'introne che, a sua volta, è posto tra i due esoni annotati, consecutivi tra loro nel genoma. Sotto, invece, sono rappresentati diversi allineamenti possibili, segnalando in blu gli allineamenti con esoni annotati e in grigio quelli con l'esone non annotato. Nel *primo caso* abbiamo una rappresentazione dell'*estensione intermedia* tramite un **MEM** intronico; nel *secondo caso* abbiamo una read che "cade" sui due esoni annotati; nel *terzo caso*, invece, abbiamo un esempio di *estensione iniziale* tramite un **MEM** intronico che permette a un prefisso della read di essere allineato (viene esteso il grafo originale che avrebbe calcolato solo un allineamento con il secondo esone annotato, non ritenendolo probabilmente sufficiente); nel *quarto caso* si ha un esempio di read che "cade" interamente in un esone non annotato (per la quale quindi si procede allo studio del solo **MEMs graph intronico**); nel *quinto caso* si ha un esempio di *estensione finale* tramite un **MEM** intronico che permette a un suffisso della read di essere allineato (viene esteso il grafo originale che avrebbe calcolato solo un allineamento con il primo esone annotato, non ritenendolo probabilmente sufficiente); negli *ultimi due casi* si hanno due read che cadono interamente, rispettivamente, nel secondo e nel primo esone.

ha uno script, chiamato `formatSAM.py`, che, preso in ingresso il genoma di riferimento, l'annotazione associata e il file `.mem` precedentemente costruito, genera il file **SAM**.
Nella realtà l'annotazione serve principalmente per poter risalire al file `.sg` con la descrizione dello **splicing graph** associato al fine di risalire alle informazioni in esso descritte.
Al fine di permettere l'integrazione dei **MEMs** intronici si sfrutta il file `.intron.sg` precedentemente descritto, creando, inoltre, un *bit vector* (non ottimizzato) per la **linearizzazione** ottenuta tramite la concatenazione della **linearizzazione** intronica a quella esonica. Si procede, in fase di lettura dei **MEMs** nello script, a creare un'unica lista con **MEMs** esonici e intronici. A tal fine questi ultimi vengono modificati in modo che il loro valore t possa identificare gli introni, la cui lista viene posta in fondo alla lista degli esoni. Saputa quindi la lunghezza della linearizzazione esonica originale, si procede aggiungendo tale valore a ogni campo t dei **MEMs** che presentano 1 come quarto valore (quarto valore che dopo questa procedura non viene più sfruttato). Si procede quindi con il calcolo originale delle stringhe *CIGAR*, calcolo che potrà sfruttare l'omogeneizzazione delle due tipologie di **MEMs**.
Una volta prodotto il file **SAM** verrà sfruttato *SAMtools* [6] al fine di produrre il file **BAM** associato, utile nello studio dei risultati.

Capitolo 4

Risultati

Al fine di poter discutere i risultati è bene prima descrivere sommariamente alcuni strumenti utilizzati a tal fine.

4.1 Script per lo studio dei risultati

Ulteriori scripts sono stati scritti al fine di studiare la qualità dei risultati e permettere una sperimentazione completa su interi cromosomi. Si ha quindi una breve anticipazione di tali scripts:

- Lo script denominato `splitGene.py` che, data in ingresso un'annotazione completa, separa il singolo file **GTF** in una serie di files **GTF**, uno per ogni gene.
- Lo script denominato `removeExon.py` che, data in ingresso un'annotazione contenente un singolo gene, rimuove, ove possibile, un esone, di lunghezza contenuta in un range desiderato, scelto in modo casuale da ogni trascritto. La rimozione dell'esone deve comunque garantire che non rimanga un solo esone, fattore che, per come è stato impostato lo studio sugli introni, renderebbe impossibile lo stesso (avendo scelto di considerare solo porzioni di genoma non codificanti contenute tra due porzioni codificanti). Il risultato prodotto sarà quindi un'annotazione contenuta in un file **GTF**.
- Lo script denominato `compareMem.py` che, dati in ingresso due files `.mem`, provenienti da due annotazioni (una è il risultato della eventuale rimozione di un esone dell'altra), confronta, per ogni read allineata, i due allineamenti e produce un semplice file di testo `.txt` contenente i risultati.

- Lo script denominato `generatePlot.py` che, dato in ingresso un insieme di files di testo con i risultati ottenuti dall'esecuzione di `compareMem.py` genera degli istogrammi, tramite *matplotlib* [14], al fine di poter visualizzare la qualità degli stessi. Tale script genera anche dei files di testo contenenti i risultati numerici.

- Lo script denominato `formatGTF.py` che, dato in ingresso il genoma di riferimento, l'annotazione in formato **GTF** e il file `.mem`, ricostruisce i probabili indici di inizio e fine degli eventuali *novel exons*, al fine di poterli eventualmente includere in una nuova annotazione.

4.2 Analisi dei risultati

Si descrivono ora i risultati ottenuti. Durante lo sviluppo del progetto ci sono stati due momenti di analisi dei risultati:

- Un primo momento, nei primi 2 mesi e mezzo di progetto, in cui veniva studiata una piccola annotazione, con 3 trascritti da 6 o 7 esoni ciascuno, relativa al *cromosoma X* umano. A tale annotazione sono stati manualmente rimossi 2 esoni non consecutivi (figura 4.1) al fine di permettere uno studio più immediato dei risultati man mano ottenuti. Tale genoma e tale annotazione possono essere ritrovati nella cartella d'esempio della repository di *ASGAL* [1].

- Un secondo momento, nelle ultime settimane, in cui lo studio si è spostato sugli interi genomi e sulle intere annotazioni di 3 cromosomi umani, il **cromosoma 1** (interessante in quanto risulta essere il più lungo cromosoma dell'uomo), il **cromosoma 9** e il **cromosoma 21** (uno dei più piccoli cromosomi dell'uomo).

Per quanto riguarda l'**RNA-Seq sample** è stato usato un tool, chiamato *RNASeqReadSimulator* [15] [16], che permette di generare un numero desiderato di reads, di lunghezza ed *error rate* arbitrari, a partire da un genoma e dall'annotazione associata.

Durante il test è stata usata una macchina con le seguenti specifiche tecniche:

- processore *Intel Core i7-8550U* con 4 cores, 8 threads, frequenza base 1.80GHz e frequenza *Turbo Boost* 4GHz

[1] `https://github.com/AlgoLab/galig/tree/master/example`

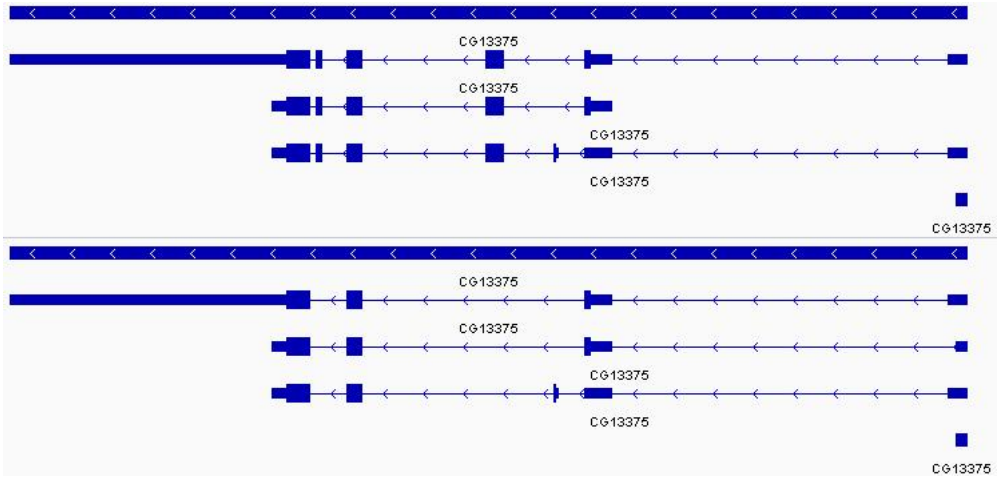


Figura 4.1: Visualizzazione tramite IGV delle due annotazioni. Sopra può essere osservata l'annotazione originale mentre sotto l'annotazione in cui sono stati rimossi i due esoni da ogni trascritto.

- ram di quantità pari a 8Gb

Il sistema operativo utilizzato è una distribuzione *GNU/Linux*.

4.2.1 Visualizzazione con IGV

In questa sezione si introduce l'uso di **IGV** [17], un software, disponibile sia in versione desktop (scritto in linguaggio *Java*) che in versione web, atto alla visualizzazione di sequenze genomiche, di annotazioni in formato **GTF** e di allineamenti in formato **BAM**.
La visualizzazione tramite **IGV** è la motivazione principale per cui è stata implementata la costruzione del file **SAM** anche a partire dai **MEMs** intronici.
Al fine di poter visualizzare gli allineamenti è stato necessario, tramite *SAMtools*, generare il file **BAM** relativo agli stessi e indicizzarlo (al fine di ottenere un più rapido accesso). Si hanno, a tale scopo, due semplici comandi *Bash*:

```
samtools view -bS file.sam | samtools sort - > file.bam
samtools index file.bam
```

Una volta ottenuto il file **BAM** sono stati confrontati, visivamente, i risultati. Sono stati analizzati, quindi, due diversi campioni di **RNA-Seq sample**.

Primo sample

Questo primo campione presenta un **RNA-Seq sample** con le seguenti caratteristiche:

- lunghezza read: 100
- numero totale di reads: 25000
- error rate: 1%

Vediamo quindi i risultati ottenuti:

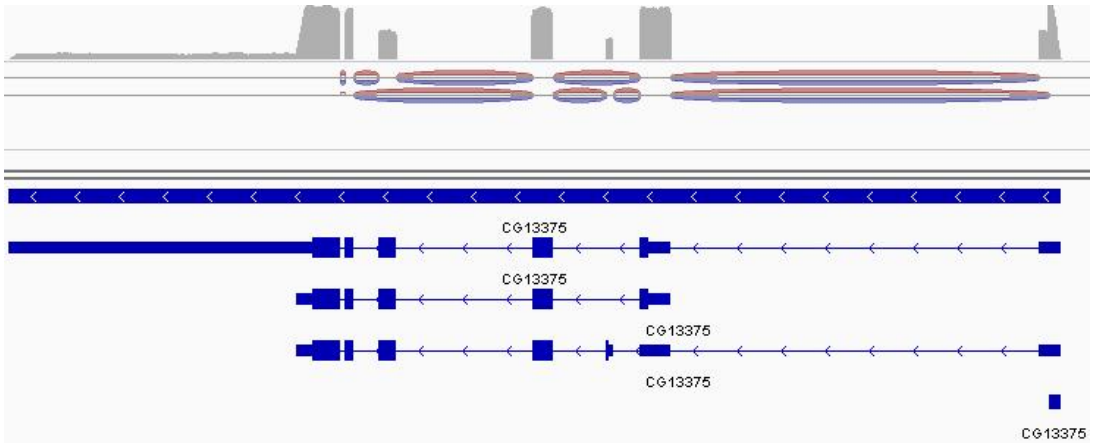


Figura 4.2: Visualizzazione degli allineamenti calcolati tramite *ASGAL* originale e annotazione completa.

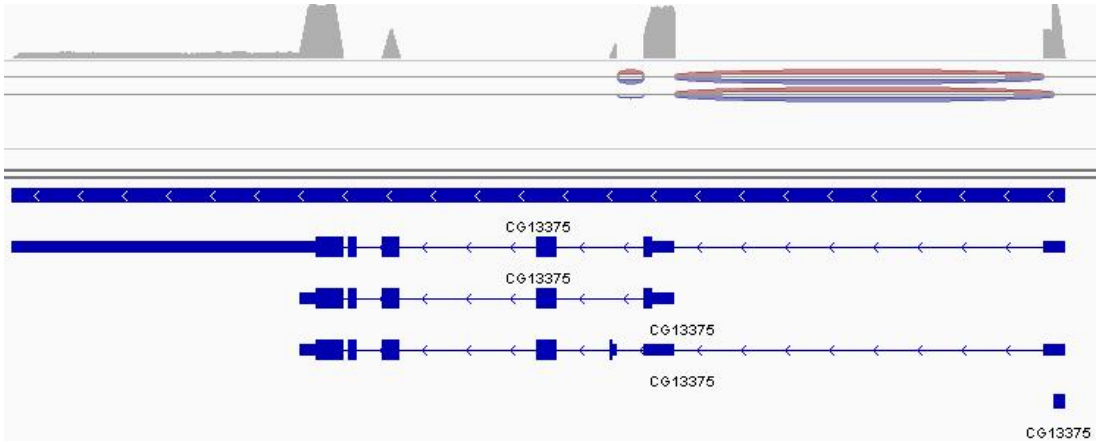


Figura 4.3: Visualizzazione degli allineamenti calcolati tramite *ASGAL* originale e annotazione priva dei due esoni. Si può notare la forte perdita di allineamenti, rispetto alla precedente figura, nella parte superiore dell'immagine.

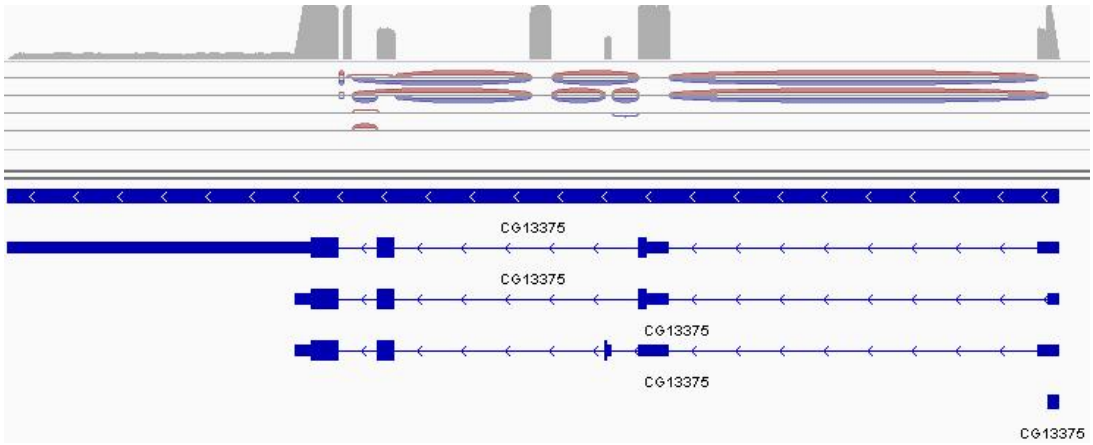


Figura 4.4: Visualizzazione degli allineamenti calcolati tramite *ASGAL*. modificato per allineare con potenziali *novel exons*, e annotazione priva dei due esoni. Si può notare in alto come siano stati "ritrovati" gli allineamenti che erano stati persi con la rimozione dei due esoni.

Si nota, inoltre, che, anche in presenza di un campione di cardinalità ridotta, composto per di più da reads di piccola dimensione, e di un'annotazione semplice, con introni non eccessivamente grossi (non a sufficienza da dover rimodulare il valore L), i tempi di esecuzione aumentano di un fattore non trascurabile:

	tempo
ASGAL originale e annotazione completa	$\sim$ 2.5s
ASGAL originale e annotazione modificata	$\sim$ 2.0s
ASGAL con parametro `--intron` e annotazione modificata	$\sim$ 4.0s

Secondo Sample

Il secondo campione presenta un **RNA-Seq sample** con le seguenti caratteristiche:

- lunghezza read: 300
- numero totale di reads: 100000
- error rate: 1%

In questo caso verranno omesse le immagini in quanto analoghe a quelle del primo sample.
Si annota che all'aumentare della quantità di read, nonché della loro lunghezza, anche in presenza di un'annotazione "semplice", le differenze dei tempi di esecuzione iniziano a essere importanti:

	tempo
ASGAL originale e annotazione completa	$\sim$ 34.5s
ASGAL originale e annotazione modificata	$\sim$ 31.5s
ASGAL con parametro `--intron` e annotazione modificata	$\sim$ 55.5s

Possiamo comunque concludere che le reads di entrambi i samples vengono allineate, con buona qualità, ai due esoni rimossi, che vengono quindi riconosciuti come novel exons*. Tale qualità verrà dimostrata durante la sperimentazione della* seconda fase *nonché con l'esecuzione di un piccolo script, atto a una parziale ricostruzione del file* ***GTF****. La sperimentazione, purtroppo, mostrerà anche come i tempi di calcolo saranno anche superiori al "triplo" nel momento in cui verranno ricercati i* novel exons*.*

Ricostruzione dell'annotazione

Come descritto, in questo primo studio, sono stati rimossi due esoni. Il primo risultava compreso tra gli indici 286949 e 287041 mentre il secondo tra gli indici 289041 e 289272. Lo script è in realtà molto semplice, nonché poco ottimizzato, e si basa su una lista booleana, interamente inizializzata a 0 di lunghezza pari alla distanza tra l'indice d'inizio dell'ultimo esone annotato e l'indice di fine del primo esone annotato (dopo aver ordinato gli esoni in ordine crescente basandosi sull'indice di partenza). Sfruttando unicamente i **MEMs** intronici, quindi con quarto campo pari a 1, la lista delle sequenze introniche, la loro linearizzazione (su cui si costruisce un *bit vector* non ottimizzato), queste ultime contenute nel file `.intron.sg`, e le operazioni di `rank` e `select`, si procede a invertire, nella lista booleana, i valori contenuti nelle sequenze di indici individuate dai **MEMs** intronici. Infine, dopo un'adeguata sistemazione degli indici basata sulla posizione della lista booleana nei confronti del genoma considerato, si procede identificando le sequenze di 1 come *novel exons*. L'esecuzione sui **MEMs** calcolati in questa prima parte di studio dei risultati producono 3 potenziali *novel exons*:

1. un primo di indici: [286946, 287044]
2. un secondo di indici: [289039, 289273]
3. un terzo di indici: [289870, 289885]

Questo è un ulteriore risultato incoraggiante in quanto i primi due *novel exons* risultano includere nei loro indici i due esoni rimossi, con un minimo margine di differenza, mentre il terzo, piccolissimo, *novel exon*, corrisponde all'inizio del piccolo esone presente solo nel terzo trascritto (figura 4.1).

4.2.2 Sperimentazione mediante Snakemake

Durante il *secondo momento* si è deciso di procedere con una vera e propria sperimentazione, al fine di studiare i risultati su interi cromosomi umani e valutarne, in modo più oggettivo rispetto al semplice riscontro visuale con IGV, i risultati. Per gestire questa sperimentazione, come già anticipato, si è deciso di utilizzare *Snakemake*, precedentemente introdotto. *Snakemake* basa il suo funzionamento sulla costruzione di un **DAG (*Directed Acyclic Graph*)**, delle regole ma anche dei singoli *job*, e sul suo studio al fine di eseguire le regole (eventualmente in modo parallelo) per ottenere un determinato output. In generale *Snakemake* prende spunto dal "paradigma" di **GNU Make** [18].

Pipeline della sperimentazione

Innanzitutto, all'interno di un file chiamato `config.yaml` abbiamo specificato le seguenti informazioni, facilmente modificabili dall'utente al fine di permettere variazioni sulla sperimentazione senza effettivamente modificare manualmente la stessa:

- il nome della cartella in cui verranno inseriti tutti i files necessari alla sperimentazione
- il nome di tre diverse sottocartelle della cartella sopra definita, una per i dati riguardanti le annotazioni complete, una per i dati riguardanti le annotazioni eventualmente prive di un esone e una per i risultati
- l'*url* per poter scaricare il genoma corrispondente a un cromosoma
- l'*url* per poter scaricare l'annotazione completa
- il nome, specificato o dal numero o dalla lettera, del cromosoma da studiare, in modo da poter estrarre dall'annotazione completa quella relativa al cromosoma di interesse
- la lunghezza minima e la lunghezza massima dell'esone che eventualmente si cercherà di rimuovere da ogni annotazione
- il numero di reads dell'**RNA-Seq sample** e la lunghezza delle stesse

Si hanno quindi diverse regole, visualizzate nel **DAG** in figura 4.5:

- Un primo insieme di regole che scaricano il genoma del cromosoma scelto e l'annotazione (da cui viene estratta l'annotazione relativa al cromosoma d'interesse). Le due regole di download saranno i due nodi *iniziali*.

- Una regola atta alla divisione dell'annotazione del cromosoma d'interesse in tante diverse annotazioni, una per ogni gene. A questo punto della sperimentazione bisogna necessariamente introdurre una particolare categoria di regole, dette *regole checkpoint*. Tali regole permettono di ridefinire lo studio del **DAG** dopo la corretta esecuzione di una determinata regola. Nel nostro caso, prima dell'ottenimento di tutte le annotazioni relative ad ogni singolo gene, non avremmo potuto conoscere i nomi dei files su cui effettuare la costruzione delle annotazioni con l'esone rimosso, il calcolo dei `.mem` e l'elaborazione dei risultati. Definendo quindi tale regola come *checkpoint* possiamo ridefinire il risultato finale che si vuole ottenere sfruttando i nomi dei files prodotti dalla regola stessa, che verranno usati come *wildcards* per tutta la successiva esecuzione della sperimentazione. A tal fine viene quindi utilizzata la funzione `expand` per ottenere una lista contenente tutti i nomi dei files che saranno necessari per completare la sperimentazione (nel dettaglio sarà sufficiente specificare i files contenenti i risultati finali della stessa in quanto sarà *Snakemake* stesso a occuparsi di automatizzare la procedura per ottenerli).

- Una regola che si occupa di generare le annotazioni eventualmente prive di un esone e porle nella cartella specificata nel file `.yaml`. Qualora non sia possibile rimuovere alcun esone in tale cartella viene posta una copia dell'annotazione originale, con nome però modificato, affinché le wildcards possano funzionare ugualmente senza ulteriori specifiche alla pipeline.

- Una regola che, partendo da ogni annotazione originale, crea, secondo le specifiche definite dall'utente, un **RNA-Seq sample**. Tale insieme di reads verrà utilizzato per tutti i 3 possibili calcoli di **MEMs**, quello sull'annotazione originale con *ASGAL* originale, quello sull'annotazione modificata con *ASGAL* originale e quello sull'annotazione modificata con *ASGAL* con l'opzione `--intron`.

- Tre regole che producono effettivamente i files `.mem` per ognuna delle situazioni sopra descritte. In queste regole viene usata un'altra particolare *keyword* offerta da *Snakemake*: *benchmark*. Specificando tale *keyword*, nonché un percorso dove salvare, per ogni esecuzione della regola stessa, i risultati in un file in formato **TSV (*Tab-Separated Values*)**, si può tenere traccia di:

  - informazioni sul tempo di esecuzione
  - informazioni sull'uso di memoria

– informazioni sul carico di *input* e *output*

Le informazioni relative al tempo di esecuzione e alla quantità di memoria utilizzata verranno utilizzate per studiare i risultati della sperimentazione.

- Due regole per utilizzare `compareMem.py` al fine di comparare, per ogni annotazione, i files `.mem` calcolati con *ASGAL* originale e annotazione completa con quelli calcolati, mediante *ASGAL* originale e *ASGAL* con l'opzione di ricerca sugli introni, sull'annotazione eventualmente priva di un esone.

- Una regola per utilizzare `generatePlot.py` al fine di produrre gli istogrammi relativi alle differenze di performances e di risultati ottenuti con il calcolo delle tre tipologie di files `.mem`. Tale regola fornisce anche risultati testuali salvati in appositi files `.txt`.

- Una regola, definibile come *finale*, che specifica, ponendoli come *input*, i files che vogliono essere generati come passo finale della pipeline (ovvero i files contenenti i risultati numerici e le immagini relative agli istogrammi). Specificando solo l'input *Snakemake* orchestrerà l'esecuzione delle regole in modo da poter permettere l'esecuzione di quest'ultima, cosa che può accadere sse tali files sono stati effettivamente generati. Tale regola, chiamata `run_after_checkpoint` in quanto definita dopo la regola checkpoint sopra descritta, sarà il *nodo terminante* del **DAG**.

Dal punto di vista del calcolo parallelo *Snakemake* ci offre l'opzione `--cores` che ci permette di specificare il numero di cores (anche non fisici) da dedicare alla pipeline. Ogni *job* verrà eseguito da un core diverso in modo del tutto automatico.

4.2.3 Confronto della qualità dei risultati

Passiamo ora allo studio definitivo dei risultati. Lo script `compareMem.py` studia i files `.mem` e tiene conto, in una prima analisi che poi verrà approfondita, due situazioni:

1. Situazioni di *match* dove i due insiemi di **MEMs** identificati sono i medesimi o dove comunque non si ha perdita di reads.

2. Situazioni di *mismatch* dove non si ha uno dei due insiemi di **MEMs** per una certa read (che non viene quindi allineata in uno dei due studi).

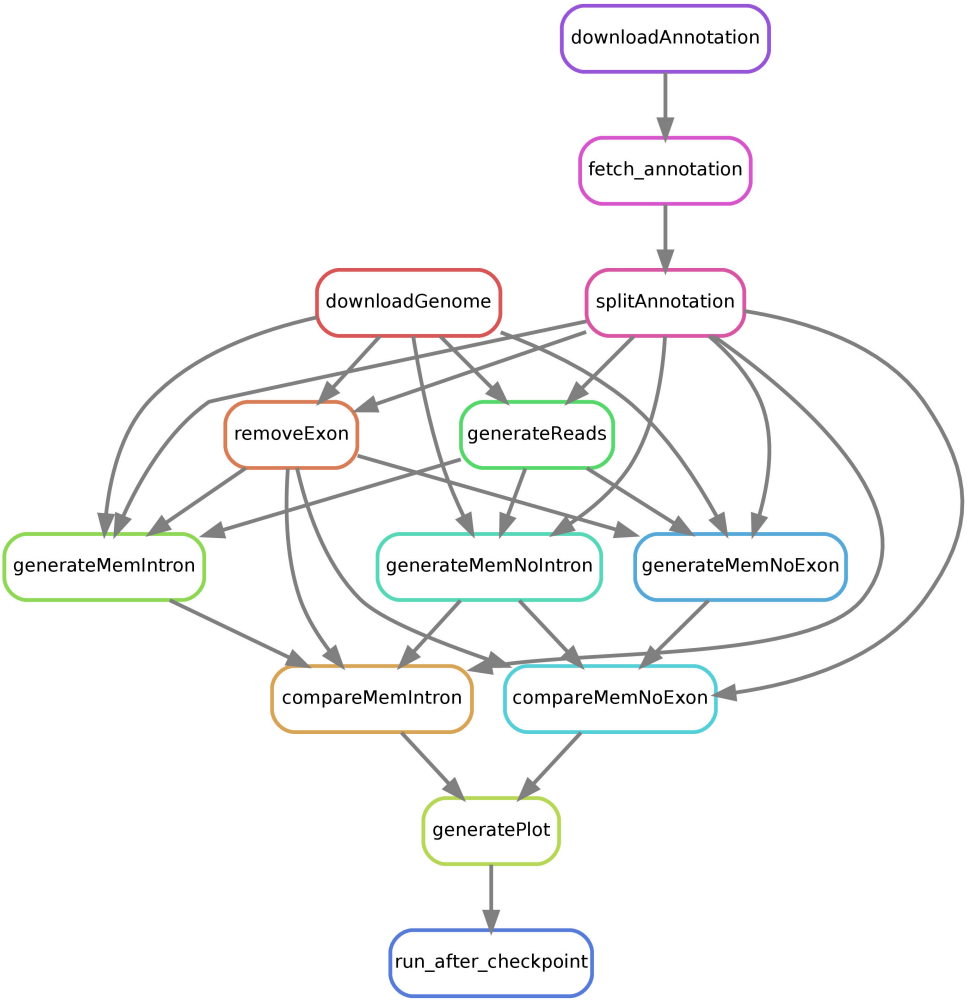


Figura 4.5: **DAG** delle regole di *Snakemake*. Si specifica che tale **DAG** non è il **DAG** ottenibile usando l'opzione `--rulegraph` in quanto le regole relative al download e alla lavorazione iniziale delle annotazioni non verrebbero mostrate, per la regola checkpoint. Il **DAG** è stato, quindi, adeguatamente esteso al fine di poter visualizzare anche tali regole. Possiamo notare come si abbia in primis il download dei files necessari con le regole `downloadGenome` e `downloadAnnotation` (a cui seguono le regole `fetch_annotation` e `splitAnnotation` per produrre le singole annotazioni). I risultati del download sono necessari per generare i vari **RNA-Seq sample** (tramite la regola `generateReads`), le annotazioni modificate (tramite la regola `removeExon`) e i vari **MEMs** (essendo necessari, per eseguire *ASGAL*, un genoma e un'annotazione). Si hanno poi le regole per la comparazione dei **MEMs** che, necessitando sia dei **MEMs** che delle annotazioni, sono collegate di conseguenza alle altre regole. Si ha infine la regola `generatePlot` che raccoglie i risultati e produce i files richiesti dall'input di `run_after_checkpoint`.

Si ha, quindi, una seconda verifica, anche se ancora non molto dettagliata, della bontà dei risultati ottenuti.
Vengono quindi generati dei semplici istogrammi con 3 colonne:

1. In **verde** i dati relativi ad *ASGAL* originale con annotazione non modificata. Tali valori saranno, in ogni istogramma, da considerarsi come il *valore ideale*.

2. In **giallo** i dati relativi ad *ASGAL* con l'opzione `--intron` e con annotazione eventualmente priva di esone. Tali valori corrispondono ai risultati a cui si è giunti con la ricerca di *novel exons* all'interno di sequenze ritenute non codificanti.

3. In **rosso** i dati relativi ad *ASGAL* originale con annotazione eventualmente priva di un esone. Tale dato ci conferma che effettivamente c'è stata, all'interno dell'intera sperimentazione su ogni cromosoma, un'effettiva rimozione di esoni in quanto *ASGAL* non riesce più a ottenere gli allineamenti ottenuti originariamente.

Analizzando l'istogramma (figura 4.6) troviamo 3 blocchi:

1. le colonne rappresentanti la media dei *match*

2. le colonne rappresentanti la media dei *mismatch*

3. una verifica sul numero totale delle reads, al fine di avere un riscontro sulla bontà dei conteggi

Si segnala, fin da ora, come dal punto di vista dei risultati non si abbiano sostanziali differenze, nonostante verranno in seguito analizzati risultati più approfonditi. La media degli insiemi di **MEMs** persi risulta praticamente nulla se comparata al numero totale di reads mentre, di contro, i match perfetti e quelli con un certo errore sono elevati. È stato verificato, inoltre, che la scelta di favorire le performances, aumentando il valore del parametro L in presenza di una linearizzazione intronica troppo estesa, perlomeno da un punto di vista statistico, non influisce sulla qualità dei risultati in quanto si potrà notare dalle tabelle a fine sezione, dove verrà conclusa l'analisi dei risultati, come la maggior parte delle reads vengono allineate. Un discorso analogo può essere fatto in merito al cambiamento di *error rate*.
Si riscontra come anche lo studio di diversi cromosomi produce risultati statisticamente paragonabili, come del resto ci si poteva attendere. L'unica differenza tangibile tra le diverse sperimentazioni è la quantità di geni presenti nelle annotazioni complete, che risulterà nel numero di files **GTF** da analizzare. Si ha infatti che:

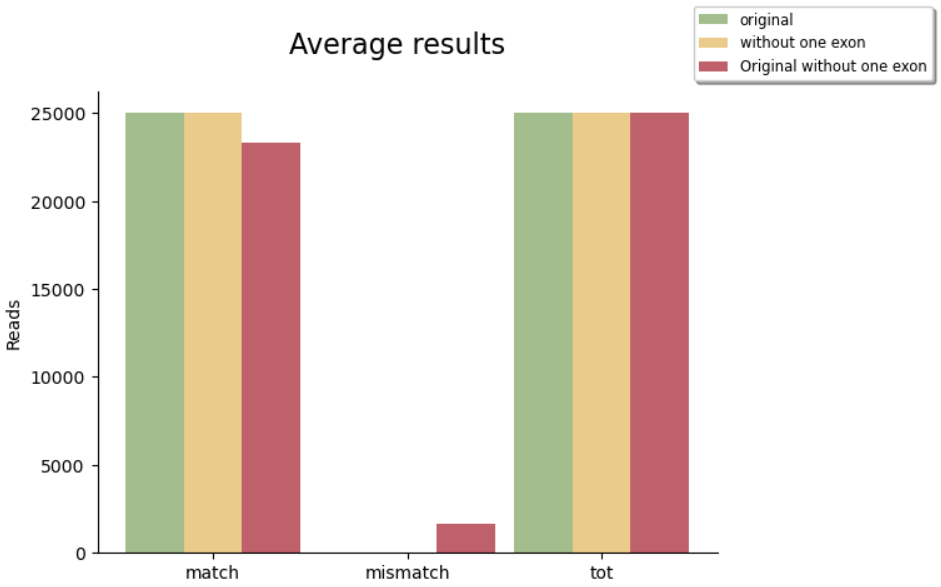


Figura 4.6: Esempio di istogramma relativo allo studio dei risultati sul cromosoma 1 con *error rate* pari all'1%.

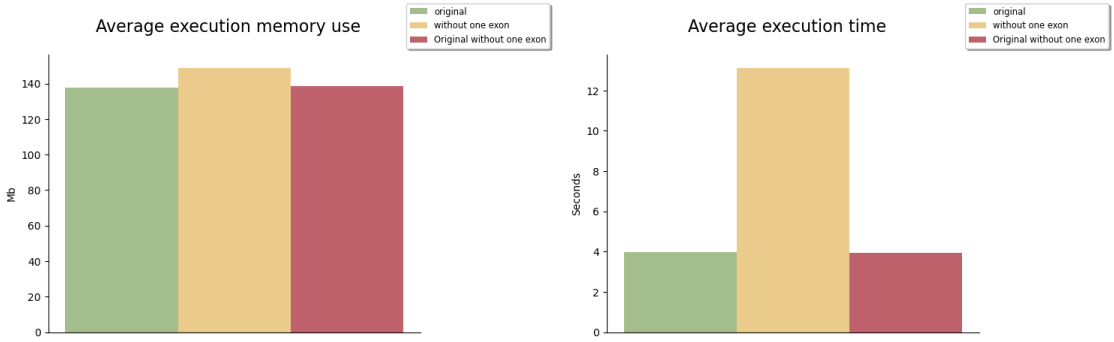
- il *cromosoma 1* conta 5475 geni (e quindi differenti files **GTF**)
- il *cromosoma 9* conta 2333 geni
- il *cromosoma 21* conta solo 875 geni

4.3 Confronto delle performances

Nonostante la qualità dei risultati della sperimentazione risulti essere parecchio incoraggiante non si può dire altrettanto delle performances, in primis dal punto di vista dei *tempi di esecuzione*.

Uso medio della memoria

In merito all'uso della memoria non si hanno particolari comportamenti da segnalare in quanto i valori delle varie esecuzioni sono comparabili sia a parità di *error rate* che al variare dello stesso. Si segnala, nel caso del *cromosoma 9*, un calo dell'uso della memoria all'aumentare dell'*error rate*, ma non si hanno evidenze del fatto che sia un miglioramento intrinseco o di altra natura. Un esempio di istogramma relativo all'uso medio di memoria è visualizzabile nella figura 4.7a



(a) (b)

Figura 4.7: Esempio di istogrammi relativi allo studio dell'uso medio della memoria (espresso in *Mb*), nella figura 4.7a, e del tempo medio (espresso in *s*), nella figura 4.7b, registrati durante la sperimentazione effettuata sul *cromosoma 1* con *error rate* pari all'1%.

Tempo medio di esecuzione

Analizzando il tempo di esecuzione medio della ricerca dei **MEMs** si segnala che la simulazione, prima dell'introduzione della modifica sulla lunghezza minima dei **MEMs** intronici in presenza di linearizzazioni introniche troppo lunghe, non riusciva, sulla macchina di test, a essere portata a conclusione a causa di interruzioni imposte dal sistema operativo a loro volta causate da *jobs* che, dopo lunghe attese di tempo (nell'ordine di diverse ore), non riuscivano a essere conclusi, da qui la necessità di introdurre tale ottimizzazione. Si riscontra, inoltre, che la modifica di *error rate* non influisce in modo rilevante sui tempi di esecuzione. Si noterà comunque dai dati che i tempi medi con la ricerca di *novel exons*, sull'annotazione priva di un esone, incrementeranno all'aumentare della grandezza del genoma e all'aumentare del numero di geni, arrivando, nel caso del *cromosoma 1*, a essere oltre il triplo dei tempi originali, probabilmente a causa di singole annotazioni geniche particolarmente complesse. Un esempio di istogramma relativo ai tempi medi d'esecuzione è visualizzabile nella figura 4.7b.

4.3.1 Tabelle riassuntive dei risultati

Vediamo quindi, più nel dettaglio, le tabelle dei risultati ottenuti sulle 4 sperimentazioni.
La prima tabella rappresenta i dati dell'esecuzione di ASGAL sull'annotazione originale. Sono quindi riportati, per ogni riga, i diversi cromosomi con l'*error rate* ($e.r$) usato per la generazione delle reads. Si hanno poi le colonne relative al tempo medio d'esecuzione e alla memoria media usata per il calcolo degli allineamenti.

Passando alla tabella relativa alle esecuzioni sull'annotazione modificata si hanno le righe dedicate ai vari cromosomi, con l'*error rate* (*e.r.*) associato, e ognuna di queste righe è a sua volta divisa in due, differenziando i due *tipi* di esecuzione:

1. La prima riga con i risultati dell'esecuzione di ASGAL con l'opzione `--intron` sull'annotazione modificata. Nominiamo tale riga *intron*.

2. La seconda riga con i risultati dell'esecuzione di ASGAL senza l'opzione `--intron` sull'annotazione modificata. Nominiamo tale riga *noIntron*. Tale riga è necessaria per valutare l'effettiva rimozione di un esone in modo da poter correttamente analizzare i dati delle sperimentazioni.

Per quanto riguarda le colonne della tabella riportante i risultati ottenuti con l'annotazione modificata si hanno:

1. I *match*, ovvero quante reads mediamente allineano le medesime sequenze genomiche rispetto all'esecuzione di ASGAL sull'annotazione originale. La colonna viene nominata *match*.

2. I *semi-match*, ovvero quante reads mediamente allineano sequenze genomiche al più di un 10% della lunghezza della read rispetto all'esecuzione di ASGAL sull'annotazione originale (quindi con errore inferiore al 10% della lunghezza della read). In altri termini si parla dei casi in cui una read effettivamente allinea in entrambi gli studi e la differenza di copertura tra i due allineamenti è accettabile. La colonna viene nominata $\leq$*10%*.

3. I *semi-mismatch*, ovvero quante reads mediamente allineano sequenze genomiche oltre al 10% della lunghezza della read rispetto all'esecuzione di ASGAL sull'annotazione originale (quindi con errore superiore al 10% della lunghezza della read). In altri termini si parla dei casi in cui una read effettivamente allinea in entrambi gli studi, ma la differenza di copertura tra i due allineamenti è eccessiva. La colonna viene nominata $>$*10%*.

4. I *mismatch*, in altre parole qualsiasi situazione diversa da quelle precedentemente descritte, ovvero, in primis, quante reads risultano allineate solo in una delle due esecuzioni, quella in studio o quella con ASGAL sull'annotazione originale. A tal proposito si segnala come potrebbero esserci reads allineate solo nel caso di studio degli introni a causa dell'aumento del massimo peso accettabile durante la ricerca del cammino minimo, fattore che comporterebbe un *mismatch*. La colonna viene nominata *mismatch*.

5. Il tempo medio di esecuzione. La colonna viene nominata *tempo*.

6. La memoria media utilizzata. La colonna viene nominata *RAM*.

Si segnala che, in caso di mancata rimozione dell'esone si potrebbero avere comunque files `.mem` *(in ogni caso molto pochi) diversi tra l'esecuzione originale di ASGAL e quella con l'opzione* `--intron` *in quanto viene aumentato il massimo peso accettabile durante la ricerca del cammino minimo.*
Si ricorda, inoltre, che tutti i dati sono stati calcolati partendo da un ***RNA-Seq sample*** *di 25000 reads lunghe 100. Per tutte le sperimentazioni è stata usata l'opzione* `--cores 4` *per quanto riguarda* Snakemake.

Abbiamo quindi la tabella relativa all'esecuzione originale, tali valori medi possono essere visti come il risultato ideale. Si nota come idealmente ci si aspettino 25000 allineamenti, ma annotazioni estremamente piccole, con un solo esone, non permettono neppure di poter generare le reads, comportando, di fatto, files `.fa` e `.mem` vuoti. Nel dettaglio, nelle sperimentazioni effettuate, si ha che:

- Per il `cromosoma 1`, che conta 5465 geni, si hanno 278 annotazioni che comportano risultati nulli nel caso di generazione delle reads con *error rate* pari al 1%.
- Per il `cromosoma 9`, che conta 2333 geni, si hanno:
  - 137 annotazioni che comportano risultati nulli nel caso di generazione delle reads con *error rate* pari al 1%.
  - 119 annotazioni che comportano risultati nulli nel caso di generazione delle reads con *error rate* pari al 2%.
  - 118 annotazioni che comportano risultati nulli nel caso di generazione delle reads con *error rate* pari al 10%.
- Per il `cromosoma 21`, che conta 875 geni, si hanno 44 annotazioni che comportano risultati nulli nel caso di generazione delle reads con *error rate* pari al 1%.

Quindi circa il 5% delle annotazioni di ogni cromosoma non permette la generazione di un ***RNA-Seq Sample*** *con il metodo adottato nel progetto.*
Passiamo quindi alla tabella delle *performances medie ideali* (ottenute con l'esecuzione originale di ASGAL su ogni annotazione non modificata):

	tempo	RAM
cromosoma 1, e.r=1%	4.6s	139.3Mb
cromosoma 9, e.r=1%	4.7s	55.4Mb
cromosoma 9, e.r=2%	4.7s	59.1Mb
cromosoma 9, e.r=10%	4.7s	47.7Mb
cromosoma 21, e.r=1%	4.2s	29.7Mb

Passiamo quindi alla tabella con i dati medi relativi all'esecuzione, sempre con un **RNA-Seq sample** da 25000 reads (lunghe 100) per gene, di ASGAL originale e con l'opzione `--intron` su annotazioni alle quali è stato eventualmente rimosso un esone:

	tipo	match	≤10%	>10%	mismatch	tempo	RAM
cromosoma	intron	24727	212	29	32	14.9s	148.7Mb
1, e.r.=1%	noIntron	23026	75	7	1892	4.4s	139.1Mb
cromosoma	intron	24747	194	23	36	9.6s	59.1Mb
9, e.r.=1%	noIntron	22993	64	6	1937	4.6s	57.2Mb
cromosoma	intron	24748	194	23	35	9.6s	59.2Mb
9, e.r.=2%	noIntron	22993	64	6	1937	4.6s	57.9Mb
cromosoma	intron	24748	193	23	36	10.0s	49.7Mb
9, e.r.=10%	noIntron	22991	64	6	1939	4.6s	49.5Mb
cromosoma	intron	24732	220	26	22	8.1s	30.6Mb
21, e.r.=1%	noIntron	22750	85	9	2156	3.9s	29.7Mb

Si possono, in conclusione, riscontrare numericamente tutte le discussioni fatte in precedenza. In tutte le sperimentazioni è possibile rilevare un netto incremento del numero degli allineamenti, fattore che comporta l'avvenuto riconoscimento di *novel exons*. I dati tra i vari cromosomi risultano statisticamente paragonabili avendo differenze medie minime. Nonostante ciò è bene segnalare come ci siano ancora discordanze, benché minime, con i risultati ideali. Tra le motivazioni si ha, in primis, la modifica del valore L ma anche la modifica del peso massimo possibile usato durante il calcolo dei cammini minimi. Si nota, inoltre, come nel caso del *cromosoma 9* il cambiamento di *error rate* non sembri portare a differenze sostanziali nei risultati.

Capitolo 5

Conclusioni

Per quanto l'implementazione del riconoscimento di *novel exons* sia da ritenersi prototipale e assolutamente sperimentale si è arrivati alla produzione di risultati incoraggianti, segno che tale funzionalità possa, in futuro e solo dopo profonde revisioni e miglioramenti, essere inclusa nel progetto *ASGAL*.
Dal punto di vista implementativo, può essere sicuramente migliorato l'aspetto relativo alle performances al fine di non dover ricorrere a "ottimizzazioni" alquanto superficiali come quella riguardante il valore L. Partendo dal principio ogni passaggio può essere migliorato, dal riconoscimento degli introni, a partire dall'annotazione, alla ricerca di **MEMs** nelle sequenze non codificanti. Il primissimo passo resta sicuramente quello di studiare, in modo più approfondito, il peso degli archi in caso di **MEMs** intronici al fine di poter anche rimuovere l'aggiustamento sul peso massimo consentito per la ricerca del cammino minimo, causa di alcune imprecisioni (per quanto assolutamente minime) nella sperimentazione finale.
Uno spunto di miglioramento, in merito invece alle performances, potrebbe essere quello di tener conto di eventuali segmenti di genoma non codificante già allineati nel corso dell'esecuzione al fine di tentare di allineare prima di tutto quelle sezioni genomiche. Tale soluzione, con molte probabilità, permetterà un calo di tempi e uso di memoria all'aumentare della grandezza dell'**RNA-Seq sample**, fino ad allinearsi coi tempi di una regolare esecuzione di *ASGAL*, in quanto eventuali possibili *novel exons* risulteranno, al progredire del numero di reads, come dei veri e propri esoni annotati. Nonostante queste considerazioni si può asserire di aver raggiunto un buon traguardo anche in termini di efficienza, avendo sfruttato, anche per gli introni, strutture dati ad alte performances, come il *bit vector*, riuscendo a limitare il rapporto tra i tempi di esecuzione e la lunghezza delle sequenze introniche.
Un ulteriore margine di miglioramento potrebbe essere ricercato nell'esecuzione parallela dello studio delle reads. Allo stato attuale dell'arte l'esecuzione di `SpliceAwareAligner` risulta essere *mono thread*, ma un'esecuzione parallela dello studio di più reads dovrebbe essere, almeno in linea teorica, priva di rischi in

quanto:

- L'ordine di studio delle reads non inficia sulla qualità del risultato.
- Non si hanno accessi pericolosi a risorse condivise.
- Non si hanno situazioni che possano provocare **deadlock** o **starvation**.

Si potrebbe inoltre meglio ottimizzare la sezione di codice in *Python*, soprattutto in merito alla comparazione dei files `.mem`, nonché della sperimentazione con *Snakemake*, al fine di ottenere una *pipeline* più pulita e performances migliori (in primis in merito all'uso dei *bit vectors* nel codice *Python*).
Inoltre, sia per la parte in *C++* che per quella in *Python* risulta necessaria un'operazione di *refactoring* del codice e una miglior gestione dei commenti, appoggiandosi, per esempio nel caso del codice *C++*, a **Doxygen** [19] al fine di ottenere una documentazione di maggior qualità.

Concludendo si può dire di aver ottenuto i risultati che erano stati prefissati nel progetto formativo e posso, a titolo personale, ritenermi soddisfatto dell'intera esperienza e degli esiti della stessa.

Bibliografia e sitografia

[1] Forluvoft / Public domain. *DNA — Wikipedia, L'enciclopedia libera*. https://it.wikipedia.org/w/index.php?title=DNA&oldid=113170912, https://commons.wikimedia.org/wiki/File:DNA_simple2_(it).svg e https://upload.wikimedia.org/wikipedia/commons/2/25/DNA_simple2_%28it%29.svg. 2020 (cit. a p. 6).

[2] Luca Denti et al. «ASGAL: aligning RNA-Seq data to a splicing graph to detect novel alternative splicing events». In: *BMC Bioinformatics* 19.1 (nov. 2018), p. 444. ISSN: 1471-2105. DOI: 10.1186/s12859-018-2436-3 (cit. alle pp. 8, 9, 13, 16, 18, 19).

[3] *ASGAL: a tool for detecting the alternative splicing events expressed in a RNA-Seq sample with respect to a gene annotation*. https://github.com/AlgoLab/galig (cit. a p. 8).

[4] *What is FASTA format?* https://zhanglab.ccmb.med.umich.edu/FASTA/ (cit. a p. 10).

[5] *GFF/GTF File Format - Definition and supported options*. https://www.ensembl.org/info/website/upload/gff.html (cit. a p. 11).

[6] H. Li et al. «The Sequence Alignment/Map format and SAMtools». In: *Bioinformatics* 25.16 (ago. 2009), pp. 2078–2079 (cit. alle pp. 12, 35).

[7] Enno Ohlebusch, Simon Gog e Adrian Kügel. «Computing matching statistics and maximal exact matches on compressed full-text indexes». In: *In Proc. 17th SPIRE (2010*. 2010, pp. 347–358 (cit. a p. 16).

[8] Balázs Dezső, Alpár Jüttner e Péter Kovács. «LEMON – an Open Source C++ Graph Template Library». In: *Electronic Notes in Theoretical Computer Science* 264.5 (2011). Proceedings of the Second Workshop on Generative Technologies (WGT) 2010, pp. 23–45. ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2011.06.003. URL: http://www.sciencedirect.com/science/article/pii/S1571066111000740 (cit. a p. 22).

[9] *Klib - a generic library in C*. https://attractivechaos.github.io/klib/ (cit. a p. 22).

[10] Simon Gog et al. «From Theory to Practice: Plug and Play with Succinct Data Structures». In: *13th International Symposium on Experimental Algorithms, (SEA 2014)*. 2014, pp. 326–337 (cit. a p. 22).

[11] Peter J. A. Cock et al. «Biopython: freely available Python tools for computational molecular biology and bioinformatics». In: *Bioinformatics* 25.11 (mar. 2009), pp. 1422–1423. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btp163. eprint: https://academic.oup.com/bioinformatics/article-pdf/25/11/1422/944180/btp163.pdf. URL: https://doi.org/10.1093/bioinformatics/btp163 (cit. a p. 22).

[12] *Gffutils - a Python package for working with GFF and GTF files in a hierarchical manner*. https://github.com/daler/gffutils (cit. a p. 22).

[13] Johannes Köster e Sven Rahmann. «Snakemake—a scalable bioinformatics workflow engine». In: *Bioinformatics* 28.19 (ago. 2012), pp. 2520–2522. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bts480. eprint: https://academic.oup.com/bioinformatics/article-pdf/28/19/2520/819790/bts480.pdf. URL: https://doi.org/10.1093/bioinformatics/bts480 (cit. a p. 22).

[14] J. D. Hunter. «Matplotlib: A 2D graphics environment». In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55 (cit. a p. 37).

[15] *RNASeqReadSimulator: a set of scripts generating simulated RNA-Seq reads that runs on Python2.7*. https://github.com/davidliwei/RNASeqReadSimulator (cit. a p. 37).

[16] *RNASeqReadSimulator: a fork of RNASeqReadSimulator that runs on Python3*. https://github.com/ldenti/RNASeqReadSimulator (cit. a p. 37).

[17] Helga Thorvaldsdóttir, James T. Robinson e Jill P. Mesirov. «Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration». In: *Briefings in Bioinformatics* 14.2 (apr. 2012), pp. 178–192. ISSN: 1467-5463. DOI: 10.1093/bib/bbs017. eprint: https://academic.oup.com/bib/article-pdf/14/2/178/546734/bbs017.pdf. URL: https://doi.org/10.1093/bib/bbs017 (cit. a p. 38).

[18] *GNU Make: a tool which controls the generation of executables and other non-source files of a program from the program's source files*. https://www.gnu.org/software/make/ (cit. a p. 42).

[19] *Doxygen: tool for generating documentation from annotated C++ sources*. https://www.doxygen.nl/index.html (cit. a p. 53).

Ringrazio, in primis, il mio relatore, il professor Gianluca Della Vedova, per avermi permesso di vivere questa esperienza e per i consigli e il supporto dati durante il periodo di lavoro sul progetto.

$\sim \cdot \sim$

Ringrazio il mio correlatore, il dottor Luca Denti, per avermi supportato (e sopportato) in questi mesi, essendo stato sempre disponibile a risolvere ogni dubbio e a guidarmi verso un corretto svolgimento del lavoro.

$\sim \cdot \sim$

Ringrazio i miei genitori, mia nonna Maria, mia nonna Laura e mia zia Rita per avermi sostenuto e incoraggiato in questi anni.

Ringrazio mio nonno Angelo e mio zio Orazio che sono sicuro sarebbero orgogliosi di me per questo traguardo tanto importante. Mi mancate tanto.

$\sim \cdot \sim$

Ringrazio Davide, Marco e Luca per essere i miei amici da sempre (o quasi).

Ringrazio Mattia A., Lello, Mattia C., Gian Marco, Clara e tutti gli altri fisici*, incontrati in un percorso universitario controverso, per avermi insegnato a non mollare mai e a non fermarmi.*

Ringrazio Gabriele, Federica, Stefano, Jack, Fra e tutti gli altri informatici*, sia quelli incontrati durante lo studio che quelli conosciuti grazie ad* unixMiB*, per aver reso questi tre anni davvero speciali.*

$\sim \cdot \sim$

Ringrazio infine Greta, per essere stata sempre con me, in questi anni, anche nei momenti più complessi. A lei, che mi ha sempre appoggiato nonché criticato in ogni circostanza in cui fosse necessario, devo buona parte di questo risultato.