Università degli studi
di Modena e Reggio Emilia

Dipartimento di Scienze Fisiche,
Informatiche e Matematiche
Corso di Laurea in Informatica

Applicazione Web per dati genomici: Studio e implementazione di un visualizzatore circolare

Gaia Ghidoni

Tesi di Laurea

Relatore:
Chiar.mo Dott. Riccardo Martoglia

Correlatore:
Chiar.mo Prof. Cristian Taccioli
Chiar.ma Dott.ssa. Chiara Vischioni

Anno Accademico 2019/2020

Ringraziamenti

Ringrazio sinceramente Ing. Riccardo Martoglia per avermi seguito in questo percorso e per avermi indirizzato nel migliore dei modi. Grazie a Prof. Cristian Taccioli e Dott.ssa Chiara Vischioni per aver reso possibile questo progetto. Grazie per i consigli, la comprensione e gli insegnamenti preziosi.

Parole chiave

Applicazione Web
Base di Dati
Interattività
MongoDB
Plotly

Indice

Elenco delle figure

Introduzione

Lo studio nell'ambito della genomica (struttura, funzione ed evoluzione delle molecole geniche) è in continuo sviluppo, anche grazie alla crescente quantità di informazioni disponibili. L'aumento dei genomi accessibili, insieme alla loro complessa natura, rendono fondamentale l'utilizzo di strumenti in grado di visualizzarne i dati in modo efficiente.

Il formato più adatto a questi scopi è quello circolare, supportato da diversi pacchetti e librerie di visualizzazione che, tuttavia, mostrano diverse limitazioni nell'uso. Spesso richiedono conoscenze di specifici linguaggi di programmazione e richiedono all'utente una elaborazione preliminare dei dati. Ancora più significativa è l'assenza di supporto per la visualizzazione dei grandi genomi, come quello umano, e per la creazione di grafici interattivi.

Il progetto InstaCircos è nato dalla collaborazione con Prof. Cristian Taccioli e Ph.D. Chiara Vischioni dell'Università degli Studi di Padova per la necessità di uno strumento che superasse le limitazioni evidenziate, fornendo, tramite un'interfaccia web di facile utilizzo, la possibilità di generare grafici circolari interattivi, senza alcun limite nelle dimensioni del genoma.

Il presente elaborato tratterà lo sviluppo del progetto InstaCircos, analizzando, nella prima parte, lo stato dell'arte dei visualizzatori circolari (Capitolo 1), il significato dei dati genomici e i requisiti dell'applicazione (Capitolo 2), nonché le tecnologie utilizzate nella sua implementazione (Capitolo 3). La seconda parte illustrerà in dettaglio i formati e la gestione dei dati di interesse (Capitolo 4), la progettazione dell'applicazione (Capitolo 5) e l'implementazione della stessa (Capitolo 6).

Parte I

Caso di studio

Capitolo 1

Stato dell'arte

I dati genomici sono dati complessi, vari, spesso incompleti e dalle grosse dimensioni. La visualizzazione di questi dati in una forma utile alla loro esplorazione è di fondamentale aiuto allo studio e all'interpretazione dei dati stessi.

Questo primo capitolo esporrà le motivazioni che hanno guidato lo sviluppo del progetto InstaCircos (Sezione 1.1), analizzando le funzionalità degli strumenti per la visualizzazione circolare attualmente disponibili (Sezione 1.2).

1.1 Motivazioni

Lo sviluppo delle tecnologie hanno reso possibile il sequenziamento di interi genomi. Nonostante molte analisi sui dati genomici sono automatizzate, alcune ricerche necessitano l'intervento umano, rendendo, quindi, di fondamentale importanza la visualizzazione dei dati in un formato chiaro e adatto allo scopo [21].

Il supporto visivo dei dati di studio è importante per le ricerche scientifiche e, in particolar modo, nel campo della bioinformatica. I dati genomici sono, infatti, complessi: possono derivare da differenti fonti, essere memorizzati in vari formati file dalla diversa organizzazione, a partire da dati strutturati a quelli non strutturati. Inoltre, contengono spesso fattori ancora ignoti e la rappresentazione visiva dei dati genomici può facilitare i ricercatori a trovare ipotesi e soluzioni.

Il primo aspetto rilevante nello sviluppo di strumenti atti alla visualizzazione di dati scientifici è la scelta del formato dei grafici, che deve essere adatto

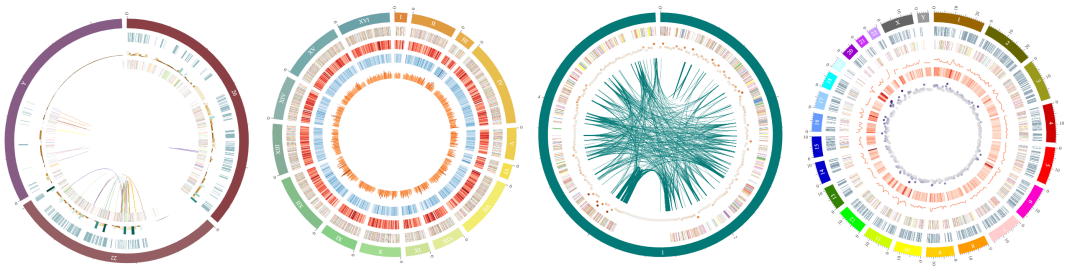

Figura 1.1: Esempi di grafici circolari generati con InstaCircos.

a rappresentare al meglio i dati, evidenziandone gli aspetti di interesse. Inoltre, la dimensione dei dati genomici ne rende la gestione non banale al fine di implementare una visualizzazione interattiva e in tempo reale.

Uno dei formati più efficaci per la rappresentazione dei dati genomici è il formato **circolare**, come mostrano gli esempi in Figura 1.1. Più in particolare, questo formato rende possibile la visualizzazione degli allineamenti di sequenze (*sequence alignment*), ovvero regioni del DNA messe a confronto. Con una struttura circolare, infatti, si possono evidenziare le regioni allineate tramite l'utilizzo di link, o corde: linee che collegano le due sezioni di DNA, che possono essere anche distanti tra loro all'interno del genoma.

Diversamente, con una struttura lineare questi collegamenti risulterebbero molto più complessi e meno leggibili.

La funzionalità dei grafici circolari è quindi quella di visualizzare i dati a livello di genoma: oltre alla visualizzazione degli allineamenti di sequenza, permettono di rappresentare le annotazioni biologiche (informazioni sulle regioni genomiche di importanza funzionale).

1.2 Related work

Molti strumenti sono stati sviluppati per rendere possibile la visualizzazione circolare dei dati genomici.

Uno tra i più conosciuti ed usati è Circos [19]. Tra i primi strumenti a proporre il modello dei grafici circolari, viene usato non solo per visualizzare i

dati genomici, ma anche per la visualizzazione di qualsiasi generico dato di cui si vogliono evidenziare le relazioni tra entità o posizioni.

Circos è scritto in Perl e le operazioni di installazione e configurazione richiedono conoscenze di questo linguaggio, oltre a quelle della linea di comando, aspetto che non lo rendono ideale per gli utenti non esperti.

Alternative a Circos sono state sviluppate per risolvere, almeno parzialmente, questo problema e fornendo, quindi, una facile installazione ma, d'altra parte, richiedendo comunque configurazione dei dati e competenze di programmazione da parte degli utenti.

Tra queste troviamo le applicazioni in R omicCircos [15] e RCircos [27], che richiedono agli utenti di avere familiarità con il linguaggio di programmazione R.

Ulteriori alternative, che richiedono sempre abilità di prorammazione, sono Circoletto [10], una suite di visualizzazione scritta in Perl, e pyCircos [3], sotfware scritto in Python.

Circoletto è stato sviluppato con lo scopo di unire, in una efficiente implementazione, le funzionalità di BLAST (Basic Local Alignment Search Tool) e Circos.

pyCircos è un pacchetto che permette la rappresentazione delle caratteristiche genomiche a partire da un file di input nel formato specifico di Genbank (trattato in dettaglio nei capitoli a venire).

Tutte le sopra menzionate opzioni richiedono che i dati genomici vengano forniti in input dall'utente, e non adottano soluzioni efficienti per l'allocazione degli stessi. Conservare interamente i dati in memoria principale rende spesso difficile, se non impossibile, la gestione e la conseguente generazione di grafici di genomi molto grandi. Ad esempio, pyCircos funziona in modo corretto ed efficiente con genomi piccoli, come quelli dei microrganismi Archea, ma quando si gestiscono file più grandi, come il genoma umano, genera errori di memoria che rendono impossibile la produzione di grafici.

Inoltre, l'interattività è, in genere, non supportata.

Tra le varie alternative a Circos è presente una applicazione web, ClicO FS [7], utilizzabile da qualsiasi utente, anche non esperto di programmazione.

ClicO FS, infatti, è un servizio web user-friendly sviluppato in R e Shiny, che si pone come obiettivo quello di rendere automatico l'utilizzo di Circos direttamente da un browser web. Rimangono, tuttavia, i restanti limiti di Circos: la limitazione a file genomici Genbank di piccole dimensioni e l'assenza di interattività. L'unica interazione possibile è, infatti, la scelta di parametri per la generazione dinamica dei grafici, ma le immagine prodotte sono statiche.

InstaCircos si propone come obiettivo quello di superare queste limitazioni permettendo, quindi, la visualizzazione efficiente di genomi anche molto grandi, offrendo, inoltre, possibilità di interazione con i grafici generati.

Capitolo 2

Progetto InstaCircos

Questo capitolo illustrerà brevemente il significato di *dati genomici*, spiegando concetti utili alla comprensione dei temi trattati nella tesi. Seguirà un paragrafo che illustrerà gli obiettivi e i requisiti del progetto, che hanno portato allo sviluppo di una applicazione web interattiva.

2.1 Informazioni genomiche

Per **genomica** si intende il settore della biologia molecolare che studia la caratterizzazione molecolare (struttura, funzione, evoluzione) dei genomi degli organismi, ovvero l'intero corredo dei cromosomi di una cellula.

La genomica è nata in seguito al progetto di sequenziamento di interi genomi (umani e di altri organismi), che ha dato accesso alla conoscenza della struttura degli acidi nucleici (DNA e RNA), e quindi a tutte le informazioni genetiche ereditarie. Questo fa del sequenziamento un'utile fonte di informazioni per lo studio degli organismi viventi e la ricerca in vari campi applicativi, tra cui la medicina.

Con il termine *genoma* vengono indicati il DNA nucleare e gli acidi nucleici contenuti in organelli come i mitocondri. Nel progetto di studio in esame verrà principalmente considerato il DNA.

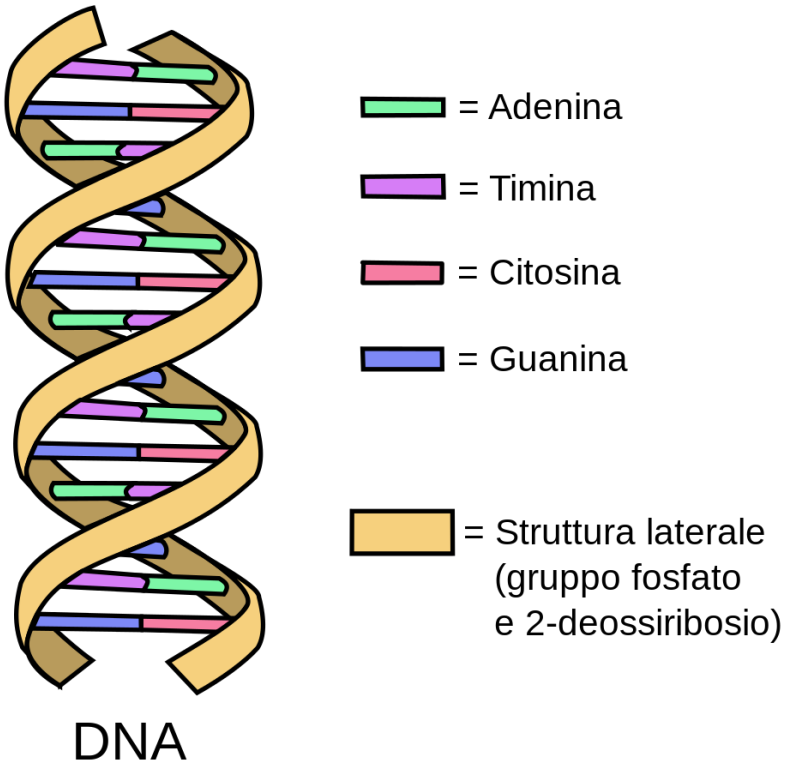

Figura 2.1: Struttura a doppia elica del DNA con basi azotate complementari.

2.1.1 Acido nucleico

Il **DNA** (acido desossiribonucleico) è un polimero organico strutturato in due catene, complementari e antiparallele, che prendono la forma di una doppia elica (Figura 2.1). La natura antiparallela delle catene di DNA permette di individuare un filamento senso (*filamento positivo* o **filamento forward**) ed un *filamento antisenso* (*filamento negativo* o **filamento reverse**). I monomeri che formano i due filamenti sono chiamati nucleotidi e sono costituiti da tre componenti: gruppo fosfato, zucchero pentoso e base azotata. Le basi azotate che costituiscono i nucleotidi sono quattro: adenina, timina, guanina e citosina. Le quattro basi si trovano nella parte interna della catena e formano legami ad idrogeno con la propria base complementare (coppie A-T e C-G).

Sono proprio le quattro basi azotate che, lette in triplette di basi, veicolano

le informazioni del genoma.

2.1.2 Geni, Regioni codificanti e regioni non codificanti

La sequenza di base azotate costituisce l'insieme delle informazioni di un genoma. Non tutto il genoma contiene, tuttavia, informazioni utili alla produzione di molecole funzionali: oltre alle regioni codificanti contenute nei geni sono presenti anche regioni non codificanti (che non contengono informazioni per la produzione di proteine) e regioni regolatrici (che regolano l'espressione genica), come schematizzato in Figura 2.2.

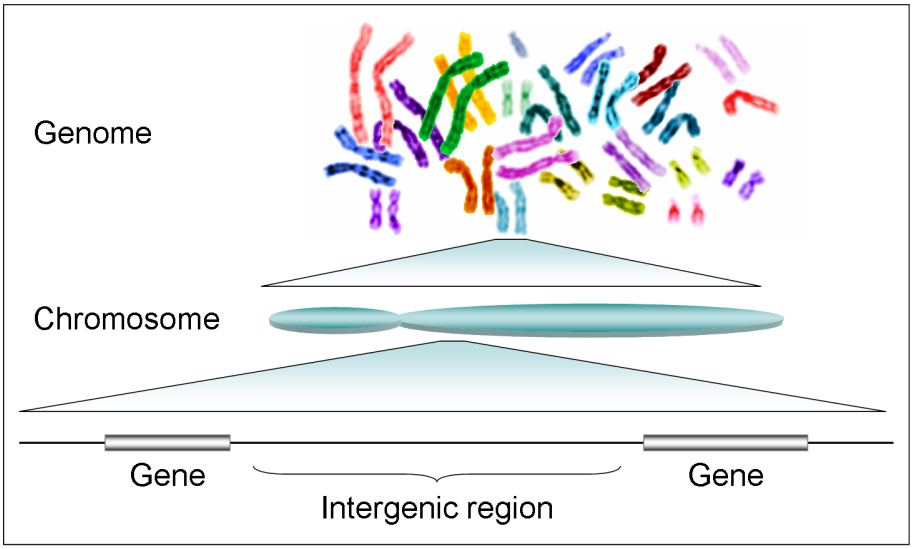

Figura 2.2: Genoma umano costituito cromosomi, ognuno dei quali contiene sequenze di geni separati da regioni intrageniche. [26]

Le sequenze di nucleotidi che codificano per un prodotto genico sono chiamate **geni**. I geni degli organismi procarioti (privi di nucleo) sono costituiti nella quasi totalità da sequenze codificanti, a differenza dei geni degli organismi eucarioti (dotati di nucleo cellulare) che possono anche contenere regioni non codificanti. In particolare, la sequenza codificante del gene eucariotico viene denominata *esone*, mentre quella non codificante *introne*.

Gli esoni sono a loro volta composti da regioni codificanti (**CDS**, coding DNA sequence) e regioni non tradotte (**UTR**, untranslated region).

2.1.3 Cromosomi

Negli organismi eucarioti, e in certi casi anche nei procarioti, il DNA si presenta impacchettato in complesse e ordinate strutture create grazie alla presenza di determinate proteine. Questa associazione di DNA e proteine viene denominata *cromatina*.

Nel momento della divisione cellulare, la cromatina si compatta ad un livello massimo di condensazione e ciascuna unità funzionale del DNA assume l'aspetto di distinti corpuscoli: i **cromosomi**.

Il numero di cromosomi è specie-specifico, ma non indica necessariamente la complessità dell'organismo.

Il genoma umano presenta 24 cromosomi: 22 autosomi (cromosomi non sessuali), denominati con i numeri da 1 a 22, e una coppia di cromosomi sessuali, X e Y.

2.1.4 Reverse complement

L'ultimo concetto che tratterà questo capitolo è quello dei reverse complement.

Come sopra esposto, il DNA si presenta, normalmente, come una coppia di filamenti complementari, dove ogni base A è accoppiata ad una base T, ogni base C è accoppiata ad una base G, e viceversa.

Data una regione di DNA, è possibile ottenere la sua sequenza reverse complement come la sequenza complementare (dove A sostituisce T e viceversa, C sostituisce G e viceversa), considerata in direzione opposta.

Ad esempio, data la seqeunza $ATGC$, il suo complementare è $TACG$. Invertendo questa sequenza si ottiene il reverse complement: GCAT.

Le sequenze reverse complement, ricercate all'interno dello stesso filamento di DNA, saranno parte dei dati che il progetto InstaCircos si propone di visualizzare, ed evidenzieranno un collegamento tra due distinte zone del genoma.

2.2 Requisiti

Il progetto InstaCircos è nato dalla necessità di visualizzare, in modo semplice, veloce e interattivo, grandi quantità di dati genomici in forma circolare.

Il grafico in forma circolare si dimostra, infatti, il metodo più efficace per la rappresentazione di dati genomici, come dimostra la varietà di strumenti attualmente disponibili che ne permettono la realizzazione. Questi strumenti, tuttavia, presentano limitazioni che li rendono incompleti, difficili da usare e, in alcuni casi, rendono persino impossibile la creazione di grafici a partire da genomi di grandi dimensioni.

Il progetto InstaCircos si propone come obiettivo il superamento delle limitazioni degli strumenti attualmente disponibili.

Si elencano i requisiti dell'applicazione.

2.2.1 Visualizzazione circolare dei dati genomici

Il primo requisito del progetto è la visualizzazione dei dati, ottenibili da un particolare formato file, in forma circolare.

Questo particolare formato richiede che i cromosomi del genoma in esame siano rappresentati come settori di una corona circolare. Ulteriori dati e dettagli dei cromosomi dovranno poter essere aggiunti al grafico come anelli concentrici.

Si richiede, inoltre, di visualizzare i reverse complement come collegamenti ad arco disegnati tra due specifiche regioni del genoma, secondo il modello del chord diagram.

2.2.2 Gestione di genomi di grandi dimensioni

Una fondamentale richiesta per il progetto è stata la possibilità di generare grafici per qualsiasi genoma, anche di grosse dimensioni, come nel caso del genoma umano.

Per gli organismi semplici, come il lievito *Saccharomyces cerevisiae*, i dati genomici sono contenuti in file dalle dimensioni di qualche MB (circa 30 per l'organismo riportato). Per organismi più complessi le dimensioni possono crescere raggiungendo qualche GB. È il caso del genoma umano, che con le sue 3,2 miliardi di paia di basi di DNA e oltre 20.000 geni codificanti proteine arriva ad occupare 4,26 GB.

La richiesta è, quindi, stata quella di poter selezionare specifici dati genomici per generare grafici circolari in tempi ragionevolmente veloci, andando così a

superare il limite che alcuni attuali strumenti hanno nella generazioni di grafici per grandi genomi.

2.2.3 Interattività e semplicità d'uso

L'ultimo requisito del progetto è stato lo sviluppo di una interfaccia web intuitiva e semplice da usare per la visualizzazione e la generazione di grafici circolari.

Questa richiesta è stata motivata dalla necessità di fornire uno strumento utilizzabile da qualsiasi utente, anche privo di conoscenze di programmazione, senza alcuna installazione necessaria.

Infine, è stata richiesta interattività per l'interfaccia web, intesa come possibilità di interagire con il grafico, ottenendo informazioni sui dati rappresentati, così come la possibilità di generare nuovi grafici dinamicamente, in base a parametri e dati selezionati dall'utente.

Capitolo 3

Studio delle tecnologie

In questo capitolo verranno illustrate le tecnologie adottate per la realizzazione del progetto InstaCircos. Verranno trattati:

- *Linguaggio di programmazione* (Sezione 3.1). Adottato per la scrittura di codice per la gestione e la visualizzazione dei dati genomici.

- *Database Management System* (Sezione 3.2 e Sezione 3.3). Software per la gestione (creazione, manipolazione e interrogazione) efficiente del database, ovvero dell'insieme dei dati di interesse per l'applicazione.

- *Librerie per la manipolazione e l'analisi dei dati* (da Sezione 3.4 a Sezione 3.6). Librerie scientifiche adottate per svolgere specifici operazioni sui dati.

- *Librerie grafiche* (Sezione 3.7 e Sezione 3.8). Le librerie per la generazioni di grafici per la visualizzazione dei dati genomici.

- *Web framework* (Sezione 3.9). Supporto logico per lo sviluppo di applicazioni web dinamiche, che permette di velocizzare il lavoro dello sviluppatore. I framework offrono, infatti, implementazioni delle più comuni funzionalità delle web application, in conformità con il principio DRY (*Don't Repeat Yourself*).

3.1 Python

La prima fase di progettazione di InstaCircos ha visto come obiettivo l'implementazione di software per la generazione di grafici circolari statici nel linguaggio di programmazione Python.

Data la problematica evidenziata dai ricercatori di Padova sull'assenza di tool bioinformatici scritti in Python, linguaggio di programmazione a loro familiare, in grado di gestire grandi genomi, si è iniziato lo studio con lo sviluppo di un codice atto a superare queste limitazioni. Solo in seguito si è proceduto con la progettazione di una applicazione web.

Il linguaggio di programmazione ad alto livello **Python** è stato utilizzato, nella prima fase, per la scrittura di programmi per l'estrazione e la gestione dei dati genomici e per la visualizzazione degli ultimi in grafici a forma circolare.

Python è stato usato anche nella fase di implementazione della applicazione web, essendo stato scelto un framework basato su tale linguaggio.

L'adozione di Python è giustificata, oltre alla specifica richiesta dei ricercatori di Padova, dalle caratteristiche dinamiche, flessibili e *general purpose* del linguaggio.

3.2 Database Management System: le opzioni

Fondamentale per gestire i dati genomici e migliorare significativamente le performance del progetto è stata la creazione di un database e quindi l'adozione di un Database Management System.

Si sono considerate varie tipologie di database al fine di valutare quella più idonea alle caratteristiche dei dati da memorizzare e alle esigenze dell'applicazione.

Uno dei criteri di scelta è stato il modello di rappresentazione dei dati adottato dal DBMS.

Le tipologie di dato possono essere riassunte nelle seguenti tre categorie:

- *Dato strutturato*: è un dato che possiede una precisa e fissata conformazione, che può essere descritto tramite schemi rigidi quali le tabelle.

- *Dato semi-strutturato*: è un tipo di dato in cui si possono individuare attributi ed elementi distinti, ma questi non possono essere descritti e racchiusi da uno schema formale.

- *Dato non strutturato*: è un dato che non è organizzato secondo alcuno schema.

Sottolineata questa distinzione, si procede alla descrizione delle opzioni considerate.

3.2.1 Database relazionale

I database relazionali (RDBMS) sono databse che supportano il **modello relazionale dei dati** e consistono in collezioni di **tabelle**, dove i dati sono memorizzati in righe (*record*), e le cui colonne ne rappresentano gli attributi, ognuno con un tipo di dato fissato ed eventuali vincoli sui valori.

A causa della natura strutturata dei database relazionali, lo schema del DB deve essere definito e chiaro già a tempo di progettazione.

Gli RDBMS sono stati introdotti all'inizio del 1980 e sono, tuttora, la tipologia di database più utilizzata, occupando le prime quattro posizioni nella classifica di popolarità stilata da DB-Engines [13], come riportato in Figura 3.1.

Rank			DBMS	Database Model	Score		
Aug 2020	Jul 2020	Aug 2019			Aug 2020	Jul 2020	Aug 2019
1.	1.	1.	Oracle	Relational, Multi-model	1355.16	+14.90	+15.68
2.	2.	2.	MySQL	Relational, Multi-model	1261.57	-6.93	+7.89
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1075.87	+16.15	-17.30
4.	4.	4.	PostgreSQL	Relational, Multi-model	536.77	+9.76	+55.43
5.	5.	5.	MongoDB	Document, Multi-model	443.56	+0.08	+38.99
6.	6.	6.	IBM Db2	Relational, Multi-model	162.45	-0.72	-10.50
7.	↑ 8.	↑ 8.	Redis	Key-value, Multi-model	152.87	+2.83	+8.79
8.	↓ 7.	↓ 7.	Elasticsearch	Search engine, Multi-model	152.32	+0.73	+3.23
9.	9.	↑ 11.	SQLite	Relational	126.82	-0.64	+4.10
10.	↑ 11.	↓ 9.	Microsoft Access	Relational	119.86	+3.32	-15.47

Figura 3.1: Classifica di popolarità dei DBMS, agosto 2020 (DB-Engine)

3.2.2 Key-value database

I key-value database memorizzano tutti i dati come una coppia chiave-valore, che li rende la tipologia più semplice tra i database NoSQL (che non adottano il modello relazionale).

Le uniche interrogazioni permesse sono quelle basate sulle chiavi e non è, quindi, possibile interrogare il database in base ai valori dei campi.

Proprio per questo motivo i database key-value permettono un accesso ai dati molto veloce, ma, normalmente, non sono adatti ad applicazioni complesse.

Sono, quindi, adatti per la memorizzazione di grandi quantità di dati, che necessitano l'esecuzione di sole query semplici.

3.2.3 Graph database

Nei database a grafo i dati sono memorizzati come nodi (*nodes*) messi in relazione tramite archi (*edges*).

Si tratta di database adatti a collezioni di dati strutturati come grafi, dove l'analisi delle relazioni tra i nodi è di primaria importanza.

Tuttavia, i DBMS non permettono, in genere, la creazione di indici su tutti i nodi, impedendo, anche in questo caso, l'accesso a questi ultimi in base ai valori degli attributi.

3.2.4 Document-oriented database

I database orientati ai documenti (il tipo più usato di database NoSQL), come suggerito dal nome stesso, organizzano i dati in documenti, spesso nei formati XML, JSON o BSON. Il concetto di documento nei database di questo tipo è assimilabile al concetto di oggetto nei linguaggi di programmazione, aspetto che partecipa alla semplificazione della gestione dei dati nei codici applicativi.

I documenti sono organizzati in collezioni, l'equivalente delle tabelle delle basi di dati relazionali. Normalmente, ogni collezione contiene documenti correlati e che condividono attributi o scopo, ma nei document-oriented database le collezioni non obbligano l'adozione di uno schema standard per i dati che contengono.

Ogni documento di una collezione può possedere attributi differenti dagli altri, così come per un campo che, se anche denominato nello stesso modo, può differire nel tipo tra i documenti della collezione.

Questo aspetto rende i document-oriented database molto più flessibili rispetto ai database relazionali.

Nei DB orientati al documento, ogni documento viene identificato da una chiave univoca e, solitamente, il database genera automaticamente un indice su queste chiavi per velocizzare il recupero dei documenti.

Oltre alla ricerca tramite chiave permettono anche l'interrogazione in base al contenuto del documento. Anche in questo caso, per migliorare le performance di ricerca e recupero dei documenti è possibile creare indici su specifici campi, con alcune limitazioni quali il numero di campi considerati e le dimensioni dell'indice stesso.

Concludendo, i database orientati al documento sono adatti alla memorizzaizone e gestioni di dati semi-strutturati o non strutturati.

3.3 DBMS adottato: MongoDB

MongoDB è un Database Management System non relazionale, document-oriented e open source, e rappresenta il database NoSQL più diffuso, come mostrato in Figura 3.1, occupando la quinta posizione della classifica di popolarità dei DBMS.

Il nome **MongoDB** deriva dal termine *humongous* (enorme, gigantesco), poichè nato nel 2007 per la necessità di un database scalabile in ambito *big data*, da cui "*a humongous database*".

Oltre al vantaggio di scalabilità, la natura document-based di MongoDB sostituisce il concetto di *riga* dei modelli relazionali con il concetto di *documento*, che non possiede uno schema predefinito. In questo modo l'aggiunta, la rimozione e la modifica dei campi del documento diventano azioni semplici, rendendo anche più veloce la sperimentazione e lo sviluppo [8].

Si espongono, di seguito, le principali caratteristiche di MongoDB.

3.3.1 Documenti e collezioni

MongoDB rapresenta i dati come documenti, che vengono raccolti in collezioni. Un insieme di collezioni costituisce un database MongoDB, come illustrato in Figura 3.2

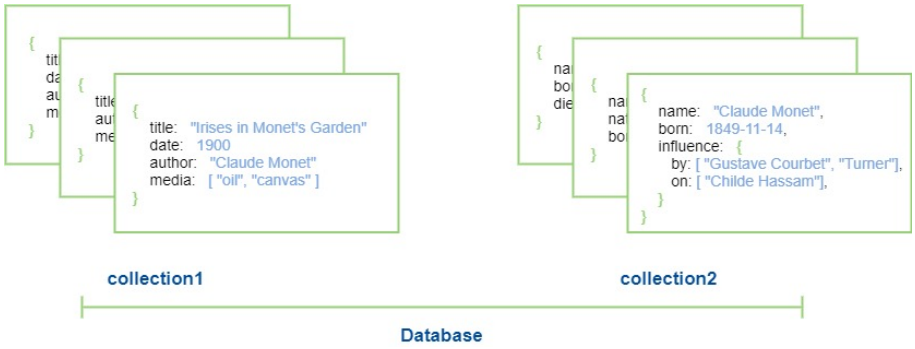

Figura 3.2: Rappresentazione di un database MongoDB, costituito da collezioni contenenti documenti.

Le collezioni sono l'equivalente delle tabelle dei database relazionali, ma, a differenza di questi ultimi, non richiedono uno schema fisso.

Dalla versione 3.2 MongoDB prevede la possibilità di definire ed eseguire, durante inserimento e modifica, **validazioni** sullo schema, a livello di collezione. Il validatore di schema può essere definito al momento della creazione della collezione, o aggiunto in una collezione esistente e già popolata di documenti. In quest'ultimo caso la validazione, sui documenti preesistenti, verrà applicata solo in caso di modifica.

Più in particolare, MongoDB memorizza i documenti come file **BSON**: la rappresentazione binaria del formato JSON (JavaScript Object Notation). I file BSON supportano, come i JSON, la definizione di **documenti innestati** e l'uso di **array**, anche contenenti documenti, ed aggiungono altri tipi di dati rappresentabili, come il *Date type* [6].

I documenti BSON possiedono una dimensione limite, 16 megabytes, e per memorizzare documenti che eccedono questo limite è possibile sfruttare la API GridFS.

La struttura dei documenti MongoDB è costituita da coppie campo-valore separati da virgola, come mostrato nell'esempio in Figura 3.2.

Ogni documento possiede sempre un campo "**_id**", che rappresenta la sua chiave primaria, univoca e immutabile. Può essere inserita al momento della creazione del documento; in caso contrario verrà generato in modo automatico, nel tipo di dato di default **ObjectId**.

Come accennato precedentemente, sono supportati i documenti innestati e gli array. Prendendo l'esempio della Figura 3.2, il documento sulla destra contiene un documento innestato (campo *influence*) e degli array (campi *on* e *by* di *influence*). L'accesso ai documenti innestati e agli array avviene tramite notazione puntata. Per accedere al secondo elemento dell'array *by* contenuto nel documento innestato *influence* la notazione è la seguente:

```
influence.by.2
```

3.3.2 Indici

Gli indici sono speciali strutture che esistono a livello di collezione e che memorizzano una parte dei dati della stessa allo scopo di renderli facilmente attraversabili [18]. In particolare, memorizzano il valore di uno o più campi, in un definito ordine, per rendere più efficienti le interrogazioni di uguaglianza o di range effettuate sul valore stesso. Permettono di limitare il numero di documenti da scansionare in seguito ad una query.

Un esempio di creazione di indice tramite la shell MongoDB è la seguente (sempre in riferimento alle collezioni dell'esempio in Figura 3.2):

```
db.collection1.createIndex( { title: -1 } )
```

Verrà creato un indice sul campo *title* in ordine decrescente (valore *-1*).

MongoDB supporta differenti tipi di indici, oltre all'indice sul singolo campo, tra cui i **Compound Index**, creati su più campi. La creazione di questi ultimi è come segue:

```
db.collection1.createIndex( { title: -1, data:1 } )
```

3.3.3 CRUD: Create, Read, Update, Delete

Il termine **CRUD** è l'acronimo di *Create, Read, Update, Delete*, ovvero le operazioni eseguibili sui documenti.

- **Create** - Per inserire documenti in una collezione si possono utilizzare due comandi:

  ```
  db.collection.insertOne()
  db.collection.insertMany()
  ```

  che, rispettivamente permettono l'inserimento di un singolo documento e di un insieme di documenti.

- **Read** - Sono le operazioni da eseguire per recuperare documenti dalle collezioni. È possibile definire dei filtri da applicare alla query. Si veda il seguente esempio (riferito a Figura 3.2):

  ```
  db.collection1.find(
      { date: {$gt: 1850} }, { title: 1, author: 1 }
  ).limit(5)
  ```

  Verranno cercati e recuperati tutti i documenti della collezione *collection1*, aventi come campo *data* un valore maggiore di 1850 (filtro), di cui verranno mostrati solamente i campi *title* e *author* (proiezione). Tra tutti i risultati ottenuti verranno selezionati i primi cinque.

- **Update** - Le operazioni possibili per la modifica di documenti esistenti in una collezione sono le seguenti:

  ```
  db.collection.updateOne()
  db.collection.updateMany()
  db.collection.replaceOne()
  ```

  I primi due comandi aggiornano, rispettivamente, un singolo documento, un insieme di documenti, mentre l'ultimo sostituisce un singolo documento.

- **Delete** - Similmente alle operazioni precedenti è possibile eliminare un singolo documento o un insieme di documenti dalla collezione, in base al filtro specificato:

```
db.collection.deleteOne()
db.collection.deleteMany()
```

3.4 NumPy

NumPy è una libreria Python open source per il calcolo scientifico [22]. Mette a disposizione array multidimensionali e routine per effettuare funzioni matematiche in modo efficiente.

Le due librerie Pandas e Matplotlib, entrambe usate per lo sviluppo del progetto, si basano ed estendono NumPy.

L'oggetto che si trova al centro di questa libreria è il **ndarray**, un array n-dimensionale, dal contenuto omogeneo per tipo di dato.

La popolarità del pacchetto deriva dalle differenze di questi oggetti rispetto alle sequenze standard di Python. Oltre alla sopra citata omogeneità dei ndarray, rispetto agli array Python che possono contenere tipi di dato diversi, una differenza tra questi oggetti risiede nella loro dimensione: i ndarray possiedono una dimensione fissata al momento della creazione, mentre le liste Python possono crescere in modo dinamico. Se si cambia la lunghezza di un array NumPy sarà necessaria la creazione di un nuovo ndarray e l'eliminazione di quello originale.

Infine, NumPy esegue avanzate operazioni matematiche su grandi quantità di dati in modo più efficiente di quanto sia possibile fare con le liste Python. È questo il motivo per cui molte librerie scientifiche e matematiche per il linguaggio di programmazione Python sfruttano gli array di NumPy.

3.5 Pandas

Pandas è una libreria Python open source che offre delle strutture dati veloci e flessibili per la manipolazione e l'analisi dei dati in modo intuitivo [23].

Pandas è adatto per diversi tipi di dati, tra cui dati tabulari con colonne di tipo eterogeneo, dati matriciali, sia omogenei che eterogenei, fino a qualsiasi tipo di dato osservato, poichè per usare le strutture dati di Pandas non è necessario etichettare i dati in alcun modo.

Le due principale strutture dati di Pandas sono le **Series**, strutture dati 1-dimensionali di tipo omogeneo, e i **DataFrame**, strutture dati 2-dimensionali, con potenziali colonne di tipo eterogeneo e dalla dimensione modificabile.

Tra le molte funzionalità di Pandas si ricordano:

- Strumenti per la lettura e la scrittura di dati in differenti formati (CSV, text, Microsoft Excel...);
- Gestione semplice dei dati mancanti, individuati tramite in valore *NaN* di NumPy;
- Possibilità di allineare i dati secondo etichette;
- Flessibilità nel modificare la struttura dei dati e nell'eseguire *group by*, *merge* e *join*;
- Creazione di etichette gerarchiche per gli assi.

3.5.1 DataFrame: introduzione

Il DataFrame è una struttura dati 2-dimensionale che può memorizzare, in colonne, differenti tipi di dato. È possibile eseguire operazioni aritmetiche sulle righe e sulle colonne.

I DataFrame possono essere considerati come contenitori di oggetti *Series* (ndarray con assi etichettati), che rappresentano le colonne del DataFrame stesso.

È possibile istanziare DataFrame fornendo i dati in differenti tipi, come dizionari Python e oggetti ndarray di NumPy, o leggendoli da file. I formati file supportati sono molteplici, quali CSV, generici file di testo delimitati e JSON.

Gli stessi formati file possono essere generati a partire da un DataFrame esistente.

Una volta creato un DataFrame è possibile manipolarne la struttura e i dati. Si può aggiungere, rimuovere, estrarre e rinominare le colonne; combinare i dati di più DataFrame; ordinare i dati in base a criteri specificati; e ottenere statistiche riassuntive sui dati.

3.5.2 DataFrame: esempio d'uso

Si consideri il dizionario Python in Figura 3.3, contente informazioni utili al caso di studio:

```
{'NC_001133.9': ['I', 230218], 'NC_001134.8': ['II', 813184],
'NC_001135.5': ['III', 316620], 'NC_001136.10': ['IV', 1531933],
'NC_001137.3': ['V', 576874], 'NC_001138.5': ['VI', 270161],
'NC_001139.9': ['VII', 1090940], 'NC_001140.6': ['VIII', 562643],
'NC_001141.2': ['IX', 439888], 'NC_001142.9': ['X', 745751],
'NC_001143.9': ['XI', 666816], 'NC_001144.5': ['XII', 1078177],
'NC_001145.3': ['XIII', 924431], 'NC_001146.8': ['XIV', 784333],
'NC_001147.6': ['XV', 1091291], 'NC_001148.4': ['XVI', 948066]}
```

Figura 3.3: Esempio di dizionario Python per la creazione di un DataFrame Pandas.

Per ogni chiave del dizionario il valore associato è una lista di due elementi, contenente una stringa e un numero intero. Questa struttura può facilmente essere memorizzata in un DataFrame di Pandas tramite il metodo **pandas.DataFrame.from_dict**. Nominando il dizionario in Figura 3.3 *data*, l'istanziazione del DataFrame sarà come segue:

```
df = pandas.DataFrame.from_dict(data, orient='index',
                                columns=['name', 'length'])
```

Il parametro *orient* specifica l'orientamento dei dati: il valore di default "columns" organizza i dati del dizionario come colonne del DataFrame finale; il valore "index" le organizza come righe, settando come etichette delle righe i valori delle chiavi del dizionario.

Il parametro *columns* specifica le etichette da assegnare alle colonne, ed è utilizzabile solamente se si specifica l'orientamento "index".

Il DataFrame risultante è riportato in Figura 3.4.

È ora possibile eseguire calcoli sui dati del DataFrame e aggiungere colonne allo stesso.

	name	length
NC_001133.9	I	230218
NC_001134.8	II	813184
NC_001135.5	III	316620
NC_001136.10	IV	1531933
NC_001137.3	V	576874
NC_001138.5	VI	270161
NC_001139.9	VII	1090940
NC_001140.6	VIII	562643
NC_001141.2	IX	439888
NC_001142.9	X	745751
NC_001143.9	XI	666816
NC_001144.5	XII	1078177
NC_001145.3	XIII	924431
NC_001146.8	XIV	784333
NC_001147.6	XV	1091291
NC_001148.4	XVI	948066

Figura 3.4: Esempio di DataFrame Pandas contenente informazioni su cromosomi.

	name	length	theta_s
NC_001134.8	II	813184	0.138021
NC_001135.5	III	316620	0.559249
NC_001136.10	IV	1531933	0.739245
NC_001138.5	VI	270161	1.816073
NC_001139.9	VII	1090940	1.973498
NC_001140.6	VIII	562643	2.529662
NC_001141.2	IX	439888	2.829177
NC_001143.9	XI	666816	3.457525
NC_001144.5	XII	1078177	3.807648
NC_001145.3	XIII	924431	4.357611
NC_001146.8	XIV	784333	4.832884
NC_001147.6	XV	1091291	5.240096
NC_001148.4	XVI	948066	5.796430

Figura 3.5: DataFrame Pandas in seguito a modifiche della struttura.

Nel codice che segue si ottengono tre informazioni dai valori contenuti nella struttura: la somma totale della colonna "length" (*df.length.sum()*), un oggetto Series contenente la somma comulativa della stessa colonna (*df.length.cumsum()*) e la dimensione degli indici del DataFrame (*df.shape[0]*).

```
degree = df.length.sum() / (360. - 1.5*df.shape[0])
cumulative_len = [0] + df.length.cumsum().to_list()[:-1]
df['theta_s'] = [ numpy.deg2rad(len/degree + 1.5*i)
                    for i, len in enumerate(cumulative_len) ]
```

Queste nuove informazioni calcolate possono essere aggiunte al DataFrame tramite un semplice assegnamento in stile dizionario di Python (ultima righa del codice sopra riportato).

Un'altra operazione possibile è la rimozione di intere righe dal DataFrame specificandone l'indice:

```
df.drop(['NC_001133.9','NC_001137.3','NC_001142.9'])
```

Il DataFrame risultante in seguito alle operazioni descritte è riportato in Figura 3.5.

3.6 Biopython

Biopython [9] è un insieme di strumenti per la computazione biologica, scritti in Python e open source, che mettono a disposizione classi per la rappresentazione, la scrittura e la lettura di dati biologici in diversi formati file.

Biopython si propone di rendere il più semplice possibile l'uso di Python nel campo della bioinformatica, offrendo una varietà di funzionalità, tra cui si evidenziano [4] [5]:

- Parsing di file bioinformatici in differenti formati, come Blast, FASTA, GenBank, in strutture Python;
- Possibilità di iterare i formati supportati e accedere a specifici record tramite una interfaccia a dizionario;
- Memorizzazione di informazioni legate alle sequenze genomiche (id, feature e sequenza stessa) in una classe standard;
- Strumenti per eseguire comuni operazioni sulle sequenze, come traduzione (da sequenze di nucleotidi ad amminoacidi) e trascrizione (da DNA a RNA).

3.6.1 Introduzione a SeqIO

Biopython fornisce una interfaccia standard di input e output per le sequenze genomiche in diversi formati file: il pacchetto **Bio.SeqIO**.

La funzione principale di questa interfaccia è **Bio.SeqIO.parse**, che prende in ingresso un file e il nome del suo formato, per ritornare un iteratore di oggetti *SeqRecord*, classe per la memorizzazione di sequenze e dati correlati come identificatore, feature e descrizione.

Per leggere un file è quindi sufficiente eseguire il seguente codice:

```
from Bio import SeqIO
with open('example.gbff', 'rU') as handle:
    for record in SeqIO.parse(handle, 'genbank'):
        print(record.id)
```

Una volta aperto il file in lettura e passato il riferimento alla funzione *parse* del modulo SeqIO, si può iterare sugli oggetti **SeqRecord**.

Tra gli attributi di quest'ultima classe troviamo anche due oggetti del pacchetto Bio: **Seq** e **SeqFeature**.

La classe *Seq* memorizza le sequenze intese come successione di lettere completa di un opzionale alfabeto. Ad esempio, una sequenza di DNA, che potrebbe essere "AGTACACTGGT", possiede un alfabeto composto dalle lettere A, C, G, T (basi azotate), che è diverso dall'alfabeto di una proteina, che dovrà, invece, rappresentare venti amminoacidi.

La classe *SeqFeature* rappresentano le feature delle sequenze. Con il termine *feature* si intendono le caratteristiche di particolari segmenti di un genoma, con determinate funzionalità . Tra le feature più comuni, ad esempio, vi sono quelle che codificano per proteine o RNA.

3.7 Matplotlib

Come menzionato precedentemente, la prima fase di sviluppo del progetto InstaCircos ha visto l'implementazione di un generatore di grafici circolari statici tramite l'utilizzo di Matplotlib.

Matplotlib [20] [16] è una libreria Python per la creazione di grafici 2D in differenti forme e formati. Tra le librerie su cui si è costruita vi è NumPy.

Alla base di questa libreria vi è il concetto di gerarchia: per generare un grafico è necessario specificare azioni su vari livelli, dal più generale al più specifico. Per questo motivo Matplotlib è organizzato in modo gerarchico.

Il livello più alto è occupato dal modulo **matplotlib.pyplot**, che offre una modalità di rappresentazione in stile MATLAB, tramite semplici funzioni per l'aggiunta di elementi al grafico.

Ad un livello più basso, per un maggiore controllo, il modulo pyplot viene usato per la sola creazione delle figura (il contenitore *top level*) e di uno o più assi, che costituiscono il contenitore principale degli elementi del tracciato. Saranno, infatti, questi ultimi oggetti che verranno usati per tutte le altre operazioni sul grafico.

3.8 Plotly

Plotly [17] è una libreria grafica e open source di Python per la creazione di grafici interattivi, che supporta più di 40 differenti tipi di diagrammi.

L'interattività di Potly è resa possibile dall'utilizzo trasparente della libreria JavaScript **plotly.js**. I grafici, denominati anche *figure*, possono essere create come dizionari Python o tramite la classe **plotly.graph_objects.Figure**. Prima di essere passate alla libreria Plotly.js, le figure verranno serializzate in formato JSON.

Le *figure* sono rappresentate come alberi, dove ogni nodo corrisponde ad un attributo. Il nodo radice possiede tre principali nodi: *data*, *layout*, *frames*.

- **Data**: deve coincidere con una lista di dizionari, ognuno dei quali è denominato *trace*, e possiede un attributi *type* che ne specifica il tipo di grafico (scatter, bar, pie, etc). Un *trace* è un insieme di punti correlati;
- **Layout**: un dizionario contenente informazioni sulla disposizione degli attributi non correlati dai dati, ma alla figura di per sè. Definisce, ad esempio, la dimensioni della figura, i margini e i colori degli assi;
- **Frames** - attributo necessario per la creazione di grafici animati, è una lista di dizionari che specifica i frame dell'animazione.

Infine, il comportamento della figura in seguito ad azioni del mouse vengono specificate nell'oggetto *config*, un dizionario. È possibile, ad esempio, abilitare la funzione di zoom o rendere il grafico completamente statico.

3.9 Dash

Dash [11] è una libreria open source per lo sviluppo di applicazioni web basato su Flask (micro-framework scritto in Python), Plotly.js e React.js. Nato dagli sviluppatori di Plotly, Dash è un framework particolarmente adatto per creare applicazioni per la visualizzazione interattiva di dati, realizzabili in modo semplice e completamente in linguaggio Python.

Nelle applicazioni Dash è possibile individuare due principali parti: il *layout* della pagina e la sua interattività (*callback*).

3.9.1 Layout delle applicazioni Dash

Il **layout** di una applicazione Dash descrive la struttura dei contenuti dell'applicazione stessa, tramite componenti Python che astraggono tag html ed elementi interattivi di più alto livello (presenti nelle librerie *dash_html_components* e *dash_core_components*).

Segue, in Figura 3.6, un esempio di una semplice applicazione Dash.

```
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

app.layout = html.Div([
    html.H6("Change the value in the text box to see callbacks in action!"),
    html.Div(["Input: ",
              dcc.Input(id='my-input', value='initial value', type='text')]),
    html.Br(),
    html.Div(id='my-output'),

])
```

Figura 3.6: Esempio di codice per la creazione di una applicazione Dash.

Come prima cosa è necessario creare una istanza di Dash, che, nell'esempio, è chiamata *app*. Per descrivere l'aspetto della pagina dell'applicazione si deve specificare l'attributo *layout* di *app*. Il layout può essere visto come un albero di componenti.

Ogni componente possiede specifici attributi, che nel caso dei componenti html, coincidono con gli attributi dei rispettivi tag, con qualche possibile variazione nei nomi (l'attributo *class* html viene chiamato *className* in Dash).

La libreria *dash_core_components* contiene elementi interattivi come i dropdown, slider e i grafici, e anche per questi componenti, le configurazioni sono effettuabili tramite argomenti keyword.

3.9.2 Dash Callback

Per rendere interattiva una applicazione web è necessario usare le callback: funzioni Python chiamate automaticamente da Dash con il cambiamento di una specificata proprietà.

In riferimento all'esempio di Figura 3.6, il contenuto dell'ultimo elemento *div* può essere aggiornato tramite una funzione callback, eseguita in seguito ad un cambiamento del componente *Input*.

```
@app.callback(
    Output(component_id='my-output', component_property='children'),
    [Input(component_id='my-input', component_property='value')]
)
def update_output_div(input_value):
    return 'Output: {}'.format(input_value)


if __name__ == '__main__':
    app.run_server(debug=True)
```

Figura 3.7: Esempio di callback Dash.

Per creare una funzione callback, come illustra l'esempio in Figura 3.7, è necessario usare il decoratore **@app.callback**. Il decoratore prende come argomenti gli input e gli output dell'interfaccia: la funzione *update_output_div* verrà invocata ad ogni cambiamento della proprietà *value* del componente che ha id *my-input*. Il valore di ritorno della funzione verrà, quindi, assegnato alla proprietà *children* del componente *my-output*. Il risultato dell'applicazione in esempio è mostrato in Figura 3.8.

In questo modo è possibile modificare dinamicamente qualsiasi proprietà dei componenti Dash.

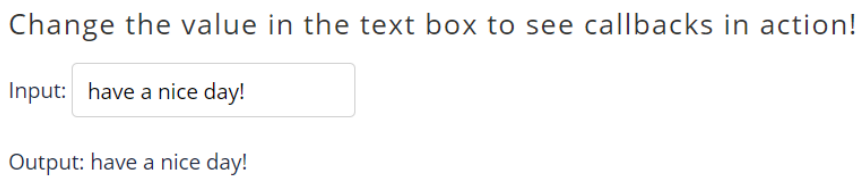

Figura 3.8: Esempio di applicazione web scritta in Dash.

La funzionalità callback di Dash segue il paradigma del **Reactive Programming**: quando una proprietà viene modificata, tutti i componenti che dipendono da essa verranno aggiornati.

3.9.3 Dash Bio

Si conclude questa sezione, nonchè capitolo della tesi, trattando di Dash Bio [12]: una suite di componenti per la visualizzazione interattiva di dati biologici, integrabili nelle applicazioni Dash.

I componenti offerti da Dash Bio possono essere divisi in tre categorie: grafici personalizzati, strumenti per l'analisi delle sequenze e stumenti per il rendering 3D.

L'elemento di interesse per lo studio in esame è **dash_bio.Circos**, che comprende una serie di strumenti per la creazione di grafici circolari interattivi, tramite l'uso della libreria d3.js.

Il componente dash_bio.Circos è costituito da due principali parti: un *layout* e delle *track*.

- Il **layout** consiste nella definizione dell'ideogramma circolare, che nel progetto InstaCircos rappresenterà i cromosomi dei genomi in esame;
- le **track** sono grafici che vengono aggiunti all'ideogramma principale e rappresentano ulteriori informazioni sul gneoma. I tipi di grafico supportati sono: heatmap, highlight, istogrammi, linee, scatter e chords.

Parte II

Progetto e implementazione dell'applicazione web interattiva

Capitolo 4

Studio e gestione dati

Questo capitolo tratterà l'analisi e la struttura dei dati genomici al fine di individuare la soluzione più adatta per l'implementazione di un database che li contenga, illustrando le motivazioni delle scelte effettuate.

In particolare, la Sezione 4.1 descriverà la collezione RefSeq, illustrandone la struttura (Sottosezione 4.1.1) e i file GBFF (Sottosezione 4.1.2) e Assembly Report (Sottosezione 4.1.3), fonte dei dati di interesse. Verranno quindi illustrati i file contenenti i reverse complement (Sezione 4.2) e la loro modalità di estrazione (Sezione 4.3). Verrà, infine, esposta la progettazione del database per la memorizzazione dei dati precedentemente descritti (Sezione 4.4).

4.1 File genomici della collezione RefSeq

I dati genomici di partenza provengono dalla collezione RefSeq [24], una raccolta di sequenze genomiche non ridondanti, riccamente annotate e pubblicamente disponibili. Creata e mantenuta dalla NCBI (National Center for Biotechnology Information), la collezione RefSeq (Reference Sequence) fornisce una selezione di sequenze genomiche assemblate provenienti da GenBank.

GenBank è il database della NCBI contenente i dati di più di 300.000 organismi, ottenuti da laboratori e grandi progetti di sequenziamento di tutto il mondo. Essendo un database di archivio, le informazioni su particolari loci (sequenza significativa di un genoma) possono essere ripetute.

- Taxonomic group
  - Genus_species
    - All assemblies
      - Individual assemblies
    - Latest assembly versions
      - Individual assemblies
    - RefSeq representative genomes (if any)
      - Individual assemblies
    - RefSeq reference genomes (if any)
      - Individual assemblies
    - Annotation releases (for organisms annotated by the NCBI Eukaryotic Genome Annotation Pipeline)
      - Data sets for each annotation release

Figura 4.1: Struttura gerarchica della collezione RefSeq.

RefSeq elimina queste ridondanze recuperando il migliore esempio per ogni molecola biologica degli organismi selezionati, che vengono scelti tra quelli per cui sono disponibili sufficienti dati.

4.1.1 Struttura della collezione RefSeq

La collezione RefSeq è accessibile dal sito FTP della NCBI [25].

I dati contenuti nella collezione RefSeq sono: DNA genico, trascritti genici e proteine derivate da tali trascrizioni.

Il sito è strutturato in cartelle organizzate in modo gerarchico, dai gruppi tassonomici alle singole specie, come mostrato in Figura 4.1.

I dati di interesse per il progetto in esame sono contenuti nella cartella *reference*.

Per accedere ai dati del genoma umano, il percorso da seguire è quello specificato in Figura 4.2.

- genomes
  - refseq
    - vertebrate_mammalian
      - Homo_sapiens
        - all_assembly_versions
        - latest_assembly_versions
        - reference
          - GCF_000001405.39_GRCh38.p13

Figura 4.2: Esempio di gerarchia di cartelle della collezione RefSeq.

Tutti i file presenti in una specifica cartella sono nominati secondo il seguente schema: [*assembly accession.version*]_[*assembly name*]_*content*.[*format*].

4.1.2 File GBFF

La principale fonte dei dati genomici utili alla visualizzazione circolare sono contenuti nei file GBFF (*GenBank Flat File*) della cartella *reference* (o *representative* nel caso in cui la prima non sia disponibile) del sito FTP.

Questo formato file rappresenta intere sequenze genomiche, includendo anche metadati e annotazioni, e si presenta come una lista di *loci*, ovvero una elenco di sezioni del genoma.

In Figura 4.3 è riportato un esempio di file GBFF, tratto dal genoma dell'organismo Saccharomyces cerevisiae.

Ogni sezione del genoma inizia con il campo *LOCUS*, che contiene informazioni quali il nome, la lunghezza e il tipo della molecola. Seguono dati identificativi, informazioni sull'organismo sorgente, e dati sugli autori e sulla sequenza stessa.

Iniziano, quindi, le informazioni riguardanti le **feature**: le caratteristiche della sequenza, come regioni che codificano per geni, proteine e RNA. Oltre alla funzionalità delle feature vengono riportati diversi altri dati, tra cui il filamento di appartenenza e la locazione della regione.

Infine, individuata dal campo *ORIGIN*, viene riportata l'intera sequenza delle basi azotate che costituiscono la sezione del genoma in esame.

4.1.3 File *Assembly Report*

Quando si sequenzia un genoma, a causa dei limiti fisici e tecnologici degli strumenti, non è possibile ottenere l'intera successione in un'unica reazione, ma si ottengono frammenti della stessa. Questi frammenti dovranno essere assemblati. Tuttavia, non sempre risulta possibile trovare la giusta posizione dei frammenti nella sequenza completa. Si individuano, quindi, diversi livelli di assemblaggio: dal massimo livello di ricostruzione della sequenza, il *complete genome*, fino ai livelli più bassi, costituiti da *scaffolds* e *contig*.

Le informazioni sul livello di assemblaggio dei loci dei file GBFF possono essere recuperate dal file **Assembly Report**, anch'esso contenuto nella cartella *reference* (o, in alternativa, *representative*) del sito FTP.

Individuabili dal suffisso *_assembly_report.txt*, gli Assembly Report sono dei file di testo *tab-delimited* che riassumono alcune delle informazioni basilari sulle

sequenze presenti nei file GBFF, specificandone anche alcuni metadati. I metadati costituiscono l'intestazione del file e sono seguiti da un elenco di record e rispettive informazioni per ogni oggetto dell'assemblaggio. Tra le informazioni più importanti si evidenziano le seguenti: nome della sequenza, codice identificativo, lunghezza e ruolo (esempio in Figura 4.4).

```
LOCUS       SCU49845     5028 bp    DNA             PLN       21-JUN-1999
DEFINITION  Saccharomyces cerevisiae TCP1-beta gene, partial cds, and Axl2p
            (AXL2) and Rev7p (REV7) genes, complete cds.
ACCESSION   U49845
VERSION     U49845.1  GI:1293613
KEYWORDS    .
SOURCE      Saccharomyces cerevisiae (baker's yeast)
  ORGANISM  Saccharomyces cerevisiae
            Eukaryota; Fungi; Ascomycota; Saccharomycotina; Saccharomycetes;
            Saccharomycetales; Saccharomycetaceae; Saccharomyces.
REFERENCE   1  (bases 1 to 5028)
  AUTHORS   Torpey,L.E., Gibbs,P.E., Nelson,J. and Lawrence,C.W.
  TITLE     Cloning and sequence of REV7, a gene whose function is required for
            DNA damage-induced mutagenesis in Saccharomyces cerevisiae
  JOURNAL   Yeast 10 (11), 1503-1509 (1994)
  PUBMED    7871890
REFERENCE   2  (bases 1 to 5028)
  AUTHORS   Roemer,T., Madden,K., Chang,J. and Snyder,M.
  TITLE     Selection of axial growth sites in yeast requires Axl2p, a novel
            plasma membrane glycoprotein
  JOURNAL   Genes Dev. 10 (7), 777-793 (1996)
  PUBMED    8846915
REFERENCE   3  (bases 1 to 5028)
  AUTHORS   Roemer,T.
  TITLE     Direct Submission
  JOURNAL   Submitted (22-FEB-1996) Terry Roemer, Biology, Yale University, New
            Haven, CT, USA
FEATURES             Location/Qualifiers
     source          1..5028
                     /organism="Saccharomyces cerevisiae"
                     /db_xref="taxon:4932"
                     /chromosome="IX"
                     /map="9"
     CDS             <1..206
                     /codon_start=3
                     /product="TCP1-beta"
                     /protein_id="AAA98665.1"
                     /db_xref="GI:1293614"
                     /translation="SSIYNGISTSGLDLNNGTIADMRQLGIVESYKLKRAVVSSASEA
                     AEVLLRVDNIIRARPRTANRQHM"
     gene            687..3158
                     /gene="AXL2"
ORIGIN
        1 gatcctccat atacaacggt atctccacct caggtttaga tctcaacaac ggaaccattg
       61 ccgacatgag acagttaggt atcgtcgaga gttacaagct aaaacgagca gtagtcagct
```

Figura 4.3: Esempio di file GBFF (Saccharomyces cerevisiae).

```
# Assembly name:  R64
# Organism name:  Saccharomyces cerevisiae S288C (baker's yeast)
# Infraspecific name:  strain=S288C
# Taxid:          559292
# BioProject:     PRJNA43747
# Submitter:      Saccharomyces Genome Database
# Date:           2014-12-17
# Synonyms:       sacCer3
# Assembly type:  haploid
# Release type:   major
# Assembly level: Complete Genome
# Genome representation: full
# RefSeq category: Reference Genome
# GenBank assembly accession: GCA_000146045.2
# RefSeq assembly accession: GCF_000146045.2
# RefSeq assembly and GenBank assemblies identical: no
#
## Assembly-Units:
## GenBank Unit Accession       RefSeq Unit Accession   Assembly-Unit name
## GCA_000146055.2      GCF_000146055.2 Primary Assembly
##      GCF_000189485.1 non-nuclear
#
# Ordered by chromosome/plasmid; the chromosomes/plasmids are followed by
# unlocalized scaffolds.
# Unplaced scaffolds are listed at the end.
# RefSeq is equal or derived from GenBank object.
#
# Sequence-Name Sequence-Role Assigned-Molecule Assigned-Molecule-Location/Type GenBank-Accn Relationship RefSeq-Accn Assembly-Unit Sequence-Length UCSC-style-name
I       assembled-molecule      I       Chromosome      BK006935.2      =       NC_001133.9     Primary Assembly        230218  chrI
II      assembled-molecule      II      Chromosome      BK006936.2      =       NC_001134.8     Primary Assembly        813184  chrII
III     assembled-molecule      III     Chromosome      BK006937.2      =       NC_001135.5     Primary Assembly        316620  chrIII
IV      assembled-molecule      IV      Chromosome      BK006938.2      =       NC_001136.10    Primary Assembly        1531933 chrIV
V       assembled-molecule      V       Chromosome      BK006939.2      =       NC_001137.3     Primary Assembly        576874  chrV
VI      assembled-molecule      VI      Chromosome      BK006940.2      =       NC_001138.5     Primary Assembly        270161  chrVI
VII     assembled-molecule      VII     Chromosome      BK006941.2      =       NC_001139.9     Primary Assembly        1090940 chrVII
VIII    assembled-molecule      VIII    Chromosome      BK006934.2      =       NC_001140.6     Primary Assembly        562643  chrVIII
IX      assembled-molecule      IX      Chromosome      BK006942.2      =       NC_001141.2     Primary Assembly        439888  chrIX
X       assembled-molecule      X       Chromosome      BK006943.2      =       NC_001142.9     Primary Assembly        745751  chrX
XI      assembled-molecule      XI      Chromosome      BK006944.2      =       NC_001143.9     Primary Assembly        666816  chrXI
XII     assembled-molecule      XII     Chromosome      BK006945.2      =       NC_001144.5     Primary Assembly        1078177 chrXII
XIII    assembled-molecule      XIII    Chromosome      BK006946.2      =       NC_001145.3     Primary Assembly        924431  chrXIII
XIV     assembled-molecule      XIV     Chromosome      BK006947.3      =       NC_001146.8     Primary Assembly        784333  chrXIV
XV      assembled-molecule      XV      Chromosome      BK006948.2      =       NC_001147.6     Primary Assembly        1091291 chrXV
XVI     assembled-molecule      XVI     Chromosome      BK006949.2      =       NC_001148.4     Primary Assembly        948066  chrXVI
MT      assembled-molecule      MT      Mitochondrion   na      <>      NC_001224.1     non-nuclear     85779   chrM
```

Figura 4.4: Esempio di file Assembly Report (Saccharomyces cerevisiae).

4.2 Reverse Complement

Gli ultimi dati di interesse per il progetto InstaCircos sono i **reverse romplement** (descritti nel Capitolo 2, Sezione 2.1.4).

Queste informazioni non sono direttamente recuperabili dai file trattati nella precedente Sezione 4.1, ma sono ottenibili da essi tramite una analisi di allineamento delle sequenze.

Come precedentemente citato, i reverse complement si individuano all'interno dello stesso filamento di una sequenza genomica. Inoltre, per la loro proprietà di complementarità, i due filamenti di DNA possiedono gli stessi reverse complement.

Una volta effettuata l'analisi si otterranno, quindi, dei link intra-cromosoma, individuanti le zone interessate al reverse complement.

4.2.1 File FASTA

L'input per l'individuazione dei reverse complement sono file FASTA: un formato file text-based per la rappresentazione delle sequenze genomiche, in cui nucleotidi e amminoacidi sono rappresentati da singole lettere. La sequenza genomica può essere preceduta da una intestazione contenente il nome della sequenza ed eventuali commenti. Un esempio di file FASTA è riportato in Figura 4.5.

```
>NC_001133.9 Saccharomyces cerevisiae S288C chromosome I, complete sequence
CCACACCACACCCACACACCCACACACCACACCACACACCACACCACACCCACACACACACATCCTAACA
CTACCCTAACACAGCCCTAATCTAACCCTGGCCAACCTGTCTCTCAACTTACCCTCCATTACCCTGCCTC
CACTCGTTACCCTGTCCCATTCAACCATACCACTCCGAACCACCATCCATCCCTCTACTTACTACCACTC
ACCCACCGTTACCCTCCAATTACCCATATCCAACCCACTGCCACTTACCCTACCATTACCCTACCATCCA
CCATGACCTACTCACCATACTGTTCTTCTACCCACCATATTGAAACGCTAACAAATGATCGTAAATAACA
CACACGTGCTTACCCTACCACTTTATACCACCACCACATGCCATACTCACCCTCACTTGTATACTGATTT
TACGTACGCACACGGATGCTACAGTATATACCATCTCAAACTTACCCTACTCTCAGATTCCACTTCACTC
CATGGCCCATCTCTCACTGAATCAGTACCAAATGCACTCACATCATTATGCACGGCACTTGCCTCAGCGG
TCTATACCCTGTGCCATTTACCCATAACGCCCATCATTATCCACATTTTGATATCTATATCTCATTCGGC
GGTCCCAAATATTGTATAACTGCCCTTAATACATACGTTATACCACTTTTGCACCATATACTTACCACTC
CATTTATATACACTTATGTCAATATTACAGAAAAATCCCCACAAAAATCACCTAAACATAAAAATATTCT
ACTTTTCAACAATAATACATAAACATATTGGCTTGTGGTAGCAACACTATCATGGTATCACTAACGTAAA
```

Figura 4.5: File FASTA: inizio del cromosoma I dell'organismo Saccharomyces cerevisiae.

4.2.2 File output di LASTZ

In seguito all'analisi delle sequenze genomiche tramite l'utilizzo di LASTZ, che verrà trattato in maggiore dettaglio nelle sezioni a seguire, i file che si ottengono in output sono dei file di testo tab-delimited, con la seguente struttura:

name1	start1	end1	name2	start2+	end2+
chr22	31112257	31112301	chr22	50805867	50805911
chr22	18531481	18531516	chr22	50805868	50805903
chr22	25482021	25482052	chr22	50805668	50805699

Ogni riga del file di testo identifica un link: un collegamento tra due regioni della sequenza genomica considerata, i cui identificativi sono riportati nelle colonne *name1* e *name2*.

Nel caso in esempio i link rappresentano regioni reverse complement trovate all'interno del Cromosoma 22 del genoma umano. Ogni collegamento è descritto dalle coordinate della sezione di partenza e da quelle della sezione di arrivo.

4.3 Pre-processing dei dati

Una volta analizzati i formati dei fati da gestire si descrivono le operazioni di pre-processing effettuate su di essi.

Si possono individuare tre gruppi di operazioni effettuate: l'individuazione delle sequenze di interesse (Sottosezione 4.3.1); la lettura dei file GBFF (Sottosezione 4.3.2); l'estrazione dei reverse complement (Sottosezione 4.3.3).

4.3.1 Individuazione delle sequenze di interesse

Come riportato precedentemente, non tutti i genomi sono completamente assemblati. Ciò di cui si ha più interesse a rappresentare sono le sequenze più complete, dove il genoma raggiunge un livello di assemblaggio tale da permette la distinzione dei **cromosomi**, unità fondamentale per la visualizzazione grafica.

Per poter identificare le sequenze cromosomiche si utilizzano le informazioni contenute nei file Assembly Report.

I file GBFF, infatti, possiedono un campo *description* in cui viene riportato il livello di assemblaggio e il ruolo della sequenza, tuttavia, non presenta una struttura standard e univoca utilizzabile per l'individuazione dei dati di interesse.

Dopo aver studiato la struttura dei file, si è trovato un modo univoco per identificare i cromosomi, che si basa sull'utilizzo dei valori di due colonne: **Sequence-Role** e **Assigned-Molecule** (visibili in Figura 4.4). In particolare, il valore della colonna *Sequence-Role* deve essere uguale ad "assembled-molecule" mentre la colonna *Assigned-Molecule* deve contenere la parola "chromosome".

La colonna *Sequence-Role* sarebbe sufficiente per l'individuazione di tutte le sequenze con il più alto livello di assemblaggio, ma includerebbe anche sequenze di DNA non nucleare (come quello contenuto negli organelli, quale il DNA mitocondriale). L'ultimo criterio serve, quindi, per escludere questi casi e selezionare effettivamente solo i cromosomi.

Siccome questa informazione è cruciale per il recupero dei dati da visualizzare, il valore di "sequence role" delle sequenze selezionate verrà aggiornato a "chromosome" e memorizzato insieme agli altri dati genomici.

4.3.2 Parsing dei file genomici GBFF

Una volta individuati i cromosomi all'interno di tutte le sequenze contenute nel file GBFF, quest'ultimo può essere letto tramite l'uso di Biopython (Sezione 3.6).

Il file viene passato all'interfaccia Bio.SeqIO e i loci vengono letti uno ad uno, per estrarre le informazioni necessarie alla memorizzazione nel database e alla successiva visualizzazione. Ogni locus è letto dall'interfaccia e ritornato in un oggetto SeqRecord i cui campi sono accessibili tramite notazione puntata.

Le informazioni che vengono estratte dagli oggetti SeqRecord sono:

- **Id**: l'identificativo della sequenza (proprio della collezione RefSeq);
- **Annotations**: un dizionario Python contenente informazioni sull'intera sequenza;
- **Description**: una stringa contenente la descrizione della sequenza;
- **Features**: un oggetto della classe SeqFeature, che rappresenta segmenti caratteristici della sequenza e che contiene i seguenti attributi:
  - **Location**: la posizione della feature all'interno della sequenza;
  - **Type**: tipo della sequenza, che ne identifica il ruolo (CDS, rRNA, exon, ect);
  - **Strand**: filamento del DNA in cui la feature è collocata. Il filamento forward è identificato con il valore 1, mentre il filamento reverse con il valore -1;
  - **Qualifiers**: un OrderedDict Python (dalla collezione *collections*) di ulteriori informazioni, come il nome dell'eventuale gene di cui la feature fa parte;
- **Seq**: una stringa contenente l'intero genoma, con alfabeto associato.

Si sottolinea, infine, che i campi *Id*, *Description* e *Seq* sono sempre presenti e unici, ma questo non è vero per i rimanenti attributi. È verosimile pensare che ogni sequenza abbia almeno una feature, e abitualmente ne possiede molteplici, ma non è da escludere la possibilità che non ve ne siano. Inoltre, l'attributo *Location* delle feature è istanza della classe Biopython *FeatureLocation*, anch'essa composta da attributi, di cui alcuni non sempre presenti: solamente se la feature

è costituita da un insieme di frammenti separati la *FeatureLocation* conterrà più coppie inizio-fine sezione. L'attributo *Annotations* è usualmente presente, ma numero e valore delle chiavi di cui è composta possono variare, come avviene per l'attributo *Qualifiers* delle feature.

Queste considerazioni saranno rilevanti nella scelta e nella progettazione del database, trattati nella Sezione a seguire.

4.3.3 Recupero dei reverse complement

Il recupero dei reverse complement viene effettuato tramite LASTZ [14], un programma di allineamento tra due sequenze di DNA.

Per ottenere i reverse complement delle sequenze contenute nei file FASTA il comando utilizzabile è il seguente:

```
lastz chr1.fa[unmask] --self --nomirror --strand=minus
--ambiguous=n --identity=100 --nogapped
--format=general:name1,start1,end1,name2,strand2,start2+,end2+
> chr1.txt
```

L'opzione *–self* indica che l'allineamento e la ricerca verrà effettuata all'interno della stessa sequenza e non tra due sequenze differenti, mentre l'opzione *–strand=minus* specifica la ricerca dei reverse complement.

Questo comando, tuttavia, recupera qualsiasi reverse complement, anche di due basi di lunghezza, e produce un output che cresce in modo quasi quadratico con la lunghezza della sequenza.

Per poter ottenere i link delle sequenze geniche più grandi è quindi necessario sfruttare dei filtri, applicando una selezione per recuperare solamente i reverse complement più rilevanti. Un criterio di filtro applicabile è la lunghezza minima dei reverse complement.

Il comando LASTZ per ottenere i link del Cromosoma 1, la sequenza più lunga del genoma umano, è il seguente:

```
lastz_32 chr1.fa[unmask] --self --nomirror --strand=minus
--ambiguous=iupac --nogapped --seed=match12 --step=2
```

```
--filter=identity:99 --filter=nmatch:100
--format=general:id%,name1,start1,end1,name2,start2+,end2+
> chr1.txt
```

Si evidenzia l'uso di *lastz_32*, eseguibile per la gestione di genomi di grandi dimensioni, e l'applicazione dei seguenti filtri:

- **–filter=nmatch:100**: seleziona tutte le sequenze di lunghezza maggiore di 100 basi;
- **–filter=identity:99**: la percentuale delle basi che fanno match nelle sequenze trovate deve essere almeno del 99
- **–seed=match12**: seleziona la regione di match perfette di 12 basi;
- **–step=2**: indica la dimensione della finestra per la ricerca dei reverse complement.

Grazie all'applicazione di questi criteri di ricerca il file output prodotto per il Cromosoma 1 è ridotto a 29MB, rispetto ai più di 900GB che si otterrebbero altrimenti.

4.4 Progettazione del database

La scelta, la progettazione e l'implementazione di un database sono state operazioni cardinali per la memorizzazione dei dati genomici in un formato conveniente, permettendo di recuperare informazioni in modo veloce ed efficace a tempo di creazione del grafico, con il conseguente miglioramento delle performance dell'applicazione.

Questa sezione illustrerà le motivazioni che hanno portato alla scelta di MongoDB (Sottosezione 4.4.1), per poi trattare la progettazione del database stesso (Sottosezione 4.4.2).

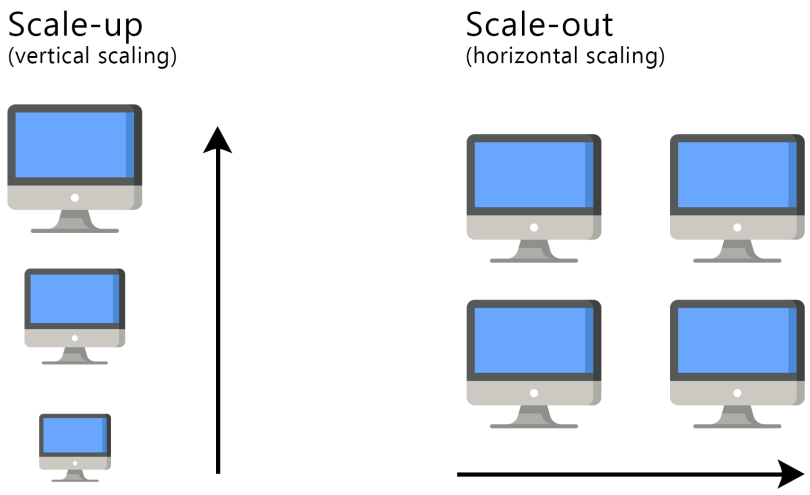

Figura 4.6: Scalabilità: differenza tra scale-up e scale-out

4.4.1 Motivazione della scelta

Il primo criterio che ha portato a scegliere MongoDB come database per il progetto InstaCircos è la natura dei dati da memorizzare. I dati genomici, trattati in dettaglio nelle sezioni precedenti, possono essere racchiusi in uno schema comune. Tuttavia, molti attributi sono spesso assenti, come evidenziato nella Sottosezione 4.3.2, e, considerando l'adozione di un database relazionale, porterebbe alla creazione di tabelle sparse.

Per questo motivo la scelta si è diretta verso un database **NoSQL**.

Tra le varie tipologie di DBMS NoSQL, è stata immediata la selezione dei database **document-oriented**, i più adatti alla memorizzazione dei dati genomici. Tra questi si è scelto di adottare **MongoDB**, sotto diverse motivazioni: è il più usato database document-oriented, provvisto di illustrativa documentazione; non è proprietario; permette un uso facile e intuitivo; supporta lo *scale-out*.

Più in dettaglio, l'ultima argomentazione interviene al momento della crescita del database. Per supportare la crescita dei dati e del loro utilizzo, senza un percepito calo delle prestazioni, le soluzioni adottabili sono due: lo *scale-up* (o vertical scaling) e lo *scale-out* (horizontal scaling), illustrate in Figura 4.6.

Lo scale-up implica l'aumento delle capacità del singolo server, ad esempio, tramite l'aggiunta di RAM o l'adozione di CPU più potenti.

Lo scale-out prevede la suddivisione del dataset tra molteplici server, superano le limitazioni tecnologiche imposte per lo scale-up. L'uso di più macchine non particolarmente potenti, ma che gestiscono un sottoinsieme del carico di lavoro, forniscono, potenzialmente, una maggiore efficienza.

Questa potenzialità potrà essere utile in caso di una crescita notevole del database alla base del progetto InstaCircos: attualmente è stato richiesto di fornire supporto per memorizzare e visualizzare sei differenti genomi, ma rimane aperta la possibilità di aggiungerne altri.

Al riguardo del formato dei dati, che ha spinto alla scelta di un database document-oriented, si sottolinea che i dati estratti tramite Biopython possiedono una struttura facilmente assimilabile al concetto di documento.

Questo è specialmente visibile per i campi più complessi, come per l'attributo *Qualifiers* delle feature. I due esempi in Figura 4.7 mostrano come questo attributo, successivamente ad un casting da OrderedDict a semplice dizionario Python, sia già in un formato adatto alla memorizzazione nel database MongoDB. Ogni *Qualifires* rappresenta un documento innestato all'interno degli oggetti feature, e al suo interno sono presenti valori di tipo array. Entrambe queste caratteristiche (documenti innestati e liste) sono supportate da MongoDB.

Si nota, inoltre, come l'unica chiave in comune tra i due oggetti sia "db_xref". L'assenza di uno schema comune nella struttura di questi attributi giustifica ulteriormente la scelta del database document-oriented, in cui la flessibilità è una caratteristica fondamentale. Anche l'eventualità che il formato dei dati genomici della collezione RefSeq venga alterato, con l'aggiunta o la rimozione di campi, non costituirebbe un problema per database senza schema come MongoDB.

```
{
 'gene': ['PHO11'],
 'locus_tag': ['YAR071W'],
 'product': ['acid phosphatase PHO11'],
 'transcript_id': ['NM_001178239.1'],
 'db_xref': ['GeneID:851299']
}

{
 'note': ['TEL01R; Telomeric region on the right arm of Chromosome I; composed of an X element core sequence
          and a short terminal stretch of telomeric repeats'],
 'db_xref': ['SGD:S000028937']
}
```

Figura 4.7: Esempio di due attributi *Qualifiers* delle feature genomiche.

4.4.2 Progetto della struttura del database

Esplicata la scelta del database MongoDB, si procede con la delineazione del progetto della base di dati.

Tutte le informazioni da memorizzare, contenute in tutti i genomi che verranno considerati, possono essere suddivisi in quattro categorie: i loci, ovvero cromosomi e sequenze ad altri livelli di assemblaggio; le feature; le sequenze geniche; i reverse complement.

In base alle considerazioni fatte durante lo studio dei dati e alle caratteristiche di MongoDB, che permettono la creazione di database, collezioni e documenti, si sono individuate due possibili strutture della base di dati, che differiscono nella composizione delle collezioni.

In entrambi i casi verranno create tante istanze di database MongoDB quanto sono i genomi da memorizzare. Questa organizzazione, che riflette, a suo modo, la struttura della collezione RefSeq, permette di selezionare e quindi ricercare e visualizzare i dati di uno specifico organismo.

Le due strutture alternative per le collezioni verranno di seguito illustrate.

A. *Due collezioni.* La prima struttura vede la creazione di due sole collezioni: una collezione *chromosome* ed una collezione *sequence.* La prima collezione è dedicata a contenere i loci (cromosomi e sequenze ad altri livelli di assemblaggio), memorizzati uno per documento insieme a tutte le informazioni disponibili, ad esclusione della sequenza genica. Oltre ai dati descrittivi del locus, ogni documento dovrà quindi contenere una lista di feature ed una lista di link reverse complement. Le sequenze geniche, considerate le loro dimensioni e il loro ruolo secondario nella creazione dei grafici , occuperanno una collezione a parte;

B. **Quattro collezioni**. La struttura alternativa consiste nella creazione di una collezione per ogni categoria di dati individuate all'inizio di questa Sottosezione, separandoli logicamente: loci, feature, reverse complement e sequenze geniche.

La struttura adottata e implementata è quella descritta nel punto B.

La creazione di una sola collezione per memorizzare loci, feature e link non è, infatti, ottimale. Mentre alcune sequenze possono essere brevi e contenere

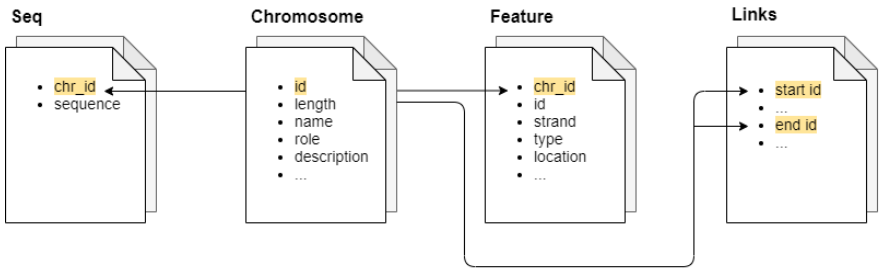

Figura 4.8: Struttura del database.

poche feature e link, altre superano facilmente la dimensione massima imposta da MongoDB sui documenti (16MB). I link dei reverse complement del cromosoma 1 del genoma umano, con i loro 29MB di dati, oltrepassano, da soli, questo limite.

Un altro motivo che ha spinto alla creazione di più collezioni riguarda l'aggiunta di informazioni ai documenti. L'aggiornamento di un documento, ad esempio tramite l'inserimento di reverse complement, che ne aumentano le dimensioni, può necessitare lo spostamento in una allocazione di memoria maggiore. Separando in una collezione differente i link, quest'ultimo problema viene evitato.

La soluzione di quattro separate collezioni permette, infine, una suddivisione logica dei dati e una maggiore semplicità nelle loro interrogazioni.

Più in dettaglio, le collezioni individuate, illustrate graficamente nella Figura 4.8, sono le seguenti:

- **Chromosome**. Questa collezione di documenti contenenti le informazioni principali di ogni sequenza: *id, annotations, description, length, name, sequence_role*;

- **Feature**. Ogni documento in questa collezione rappresenta una feature. Per ogni feature vengono memorizzare: il riferimento all'*id* della sequenza in cui si trova, lo *strand* e la posizione, come coordinate inizio-fine (*start* e *end*), all'interno di essa. Se presente, un documento innestato, *qualifiers*, includerà ulteriori informazioni sulla feature. Infine, se la feature è costituita da una serie di sezioni non contigue, un campo addizionale *locations* conterrà la lista di delle coordinate di tutti i segmenti;

- **Links**. La collezione *Links* contiene i dati riguardanti gli allineamenti delle sequenze. Ogni link è un documento contenente due documenti innestati, *start* e *end*, contenenti le informazioni sulle sequenze coinvolte nel collegamento. I campi rappresentati sono: *id* della sequenza; una lista *location* che include la coppia inizio-fine link; la lugnhezza, *length*, del link;

- **Seq**. Questa collezione memorizza le sequenze geniche nel tipo di dato stringa e un riferimento all'identificativo del suo locus memorizzato nella collezione *Chromosome*.

Per concludere questo capitolo si riporta un'ultima scelta progettuale. Nonostante i loci di interesse per la visualizzazione siano le sequenze cromosomiche, si è deciso di inserire nel database l'intera collezione di dati contenuta nei file GBFF, comprese le sequenze a più basso livello di assemblaggio. Questo per supportare, se richiesto in futuro, la ricerca e la visualizzazione di qualsiasi sequenza.

Capitolo 5

Progettazione dell'interfaccia web

Il presente capitolo descriverà la progettazione dell'interfaccia web in base ai requisiti precedentemente trattati nella Sezione 2.2.

Si esporranno le motivazioni della scelta di Dash come framework per l'implementazione dell'applicazione web (Sezione 5.1), per poi trattare della progettazione della stessa (Sezione 5.2). Il capitolo si conclude con una panoramica ad alto livello dell'applicazione (Sezione 5.3).

5.1 Scelta del web framework

Come supporto logico allo sviluppo dell'applicazione web dinamica si è scelto di utilizzare il framework **Dash**.

In primo luogo, Dash, essendo perfettamente integrato con Plotly, permette la realizzazione di applicazioni web per la visualizzazione interattiva di dati in modo semplice e intuitivo, tramite l'uso del solo linguaggio di programmazione Python.

Lo scopo principale del progetto InstaCircos è la creazione di grafici circolari, e si presta perfettamente all'uso del framework scelto.

Dash è sviluppato sul micro-framework Flask: un framework leggero e minimale, che non supporta direttamente alcune delle funzionalità comuni alle web application, come invece succede per i full-stack framework. Flask, e quindi

Dash, non supportano, ad esempio, l'astrazione per le basi di dati e l'autenticazione di utenti. Queste funzionalità possono però essere aggiunte tramite estensioni e librerie di terze parti.

Il progetto InstaCircos non prevede la necessità di creare e gestire utenti, e la interazione con la base di dati verrà scritta direttamente in Python, evitando probabili incompatibilità tra il modello document-oriented e l'ORM (Object-relational mapping) dei framework full-stack quali Django.

Si sono considerate, infine, le necessità legate alla tipologia di grafici da visualizzare.

In fase di progettazione dell'applicazione web si sono scartate tutte le librerie grafiche che non permettevano la creazione di grafici interattivi e circolari, non rispettando, quindi, i requisiti sulla visualizzazione dei dati genomici.

La librerie grafiche ottimali per questi scopi sono state individuate in Plotly e nel suo framework Dash, che oltre a supportare una forte interattività con l'utente, offre, nella suite Dash Bio, supporto specializzato per la visualizzazione di dati genomici, in un formato simile a quello realizzabile, staticamente, con la libreria Matplotlib.

5.2 Funzionalità e struttura dell'applicazione

In seguito ad una approfondita analisi e discussione dei requisiti (Sezione 2.2), si sono individuate le funzionalità e l'interattività da implementare e fornire nell'applicazione web InstaCircos, che vengono di seguito riportate:

- **Visualizzazione dei dati genomici in forma circolare**. Come sottolineato più volte nel corso dei capitoli precedenti, questa è la funzionalità primaria di InstaCircos, e, in quanto tale, sarà il fulcro dell'applicazione. Un esempio di grafico circolare è mostrato in Figura 5.1, dove ogni traccia è descritta in dettaglio;

- **Selezione di differenti genomi**. Si richiede di poter specificare una specie, tra le disponibili nel database alla base del progetto, e di visualizzarne i dati nel grafico;

- **Interattività del grafico**. Per poter visualizzare meglio e comprendere più a fondo i dati del grafico si richiede la possibilità di ingrandire la figura

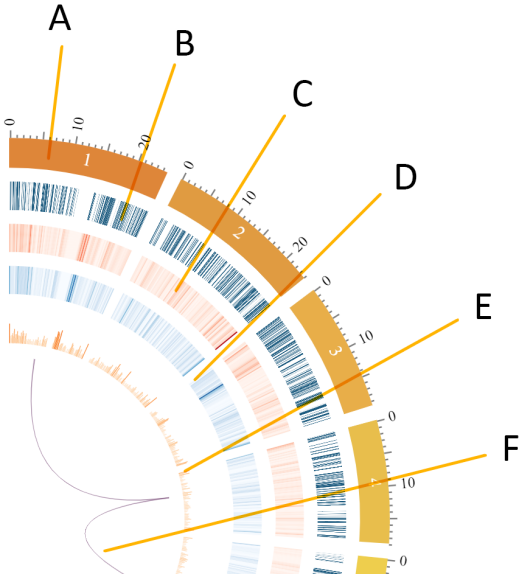

Figura 5.1: Dettagli del grafico circolare del genoma umano. I cromosomi, etichettati con i loro nomi, sono arrangiati nel cerchio più esterno (**A**), dove è visibile una scala di lunghezza. La prima traccia (**B**) rappresenta tutte le regioni CDS situate nel filamento forward del DNA, ed è seguito da due tracce *heatmap* che mostrano la densità CDS rispettivamente sul filamento forward (**C**) e sul filamento reverse (**D**). La quarta traccia riporta la densità genica calcolata su entrambi i filamenti (**E**). Infine, le tracce più interne mostrano alcuni link di esempio tra cromosomi (**F**).

ed ottenere dettagli sugli elementi rappresentati, tramite il passaggio del mouse;

- **Visualizzazione di informazioni riassuntive**. Per avere più chiarezza sul contenuto del grafico a più alto livello, si riportano delle informazioni riassuntive su ogni sua traccia (lunghezza complessiva del genoma, numero di feature trovate, eccetera);

- **Creazione dinamica di nuovi grafici**. Fondamentale è la richiesta di poter generare nuovi grafici, una volta selezionato il genoma di interesse. La creazione dei grafici deve essere effettuata in modo dinamico, intuitivo e veloce, specificandone i dati di interesse ed i semplici parametri necessari per le impostazioni del tracciato;

- **Possibilità di scaricare il grafico creato**. Deve essere possibile scaricare, in un formato file grafico, il tracciato circolare che si sta visualizzando.

La penultima funzionalità, la creazione dinamica di nuovi grafici, può essere ulteriormente specificata descrivendone le possibili azioni che l'utente può svolgere per creare il grafico:

- **Selezione dei cromosomi da visualizzare**. Oltre alla creazione di grafici circolari di tutte le sequenze cromosomiche è necessario poterne visualizzare un sottoinsieme, per avere una prospettiva di dettaglio e, soprattutto, per rendere più leggibili i link rappresentanti i reverse complement;
- **Visualizzazione delle feature**. Si deve poter visualizzare la posizione della feature, specificandone il tipo e il filamento su cui ricercarle;
- **Rappresentazione della densità delle feature**. La densità delle feature deve essere specificata tramite il tipo e il filamento delle feature di interesse e il tipo di grafico della traccia;
- **Visualizzazione dei reverse complement**. I reverse complement sono rappresentati tramite link (tipo di grafico *chord*). Deve essere possibile filtrare i dati in base a due fattori: la lunghezza e la posizione. Un valore di soglia indicherà la lunghezza minima dei link da disegnare e, eventualmente, tipo e filamento delle feature indicheranno di rappresentare solamente i link che si trovano completamente all'interno di queste caratteristiche regioni.

Individuate tutte le funzioni dell'applicazione web si è proceduto alla progettazione della sua struttura.

Per rispettare il requisito di semplicità e usabilità si è determinata una applicazione strutturata in **una sola pagina**, in modo da evitare complicate organizzazioni di pagine innestate. Le varie funzionalità verranno quindi implementate in un pannello *Tab*, dove ogni scheda sarà dedicata all'esecuzione di determinate operazioni.

Mediante questa struttura sarà possibile mantenere in posizione centrale il grafico circolare, anche durante l'esplorazione delle schede e delle funzionalità dell'applicazione web.

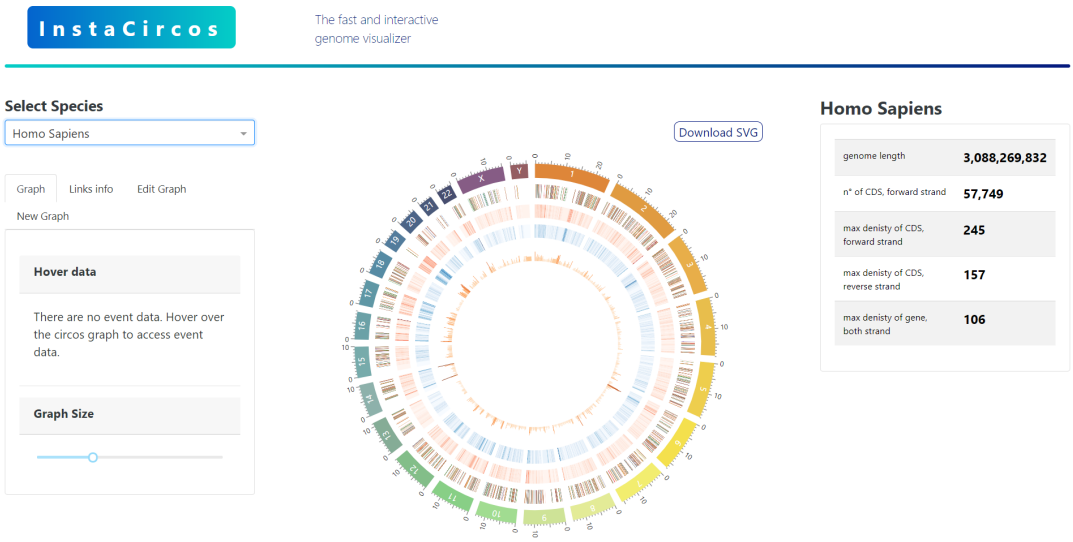

Figura 5.2: Applicazione web InstaCircos, con grafico di esempio (al centro), pannello a schede (a sinistra) e pannello riassuntivo (a destra).

5.3 Panoramica dell'applicazione

La Figura 5.2 mostra l'interfaccia grafica dell'applicazione web InstaCircos.

Il punto focale dell'interfaccia utente è la visualizzazione dei dati genomici. Il grafico si trova in posizione centrale, mentre ai suoi lati sono posti un pannello di riepilogo e un pannello multi-tab, che espongono informazioni ed aggiungono interattività all'applicazione web.

Un menu a discesa è posto sotto al logo *InstaCircos* e permette all'utente di **scegliere una delle specie** a disposizione nel database dell'applicazione. Una volta selezionata la specie, il suo genoma viene rappresentato in un grafico circolare che potrà essere esplorato e modificato.

Il grafico può essere **ingrandito** e **spostato** all'interno del suo *canvas* ("tela", elemento che lo contiene), e **informazioni sui dati rappresentati** verranno mostrato non appena il mouse si posiziona su un componente del grafico. Quest'ultima ed altre funzionalità sono riportate in Figura 5.3

Un bottone posto al di sopra del tracciato rende possibile **scaricare**, sotto forma di immagine vettoriale (formato immagine SVG), il grafico visualizzato.

Il pannello sulla destra viene creato in modo dinamico nel momento della

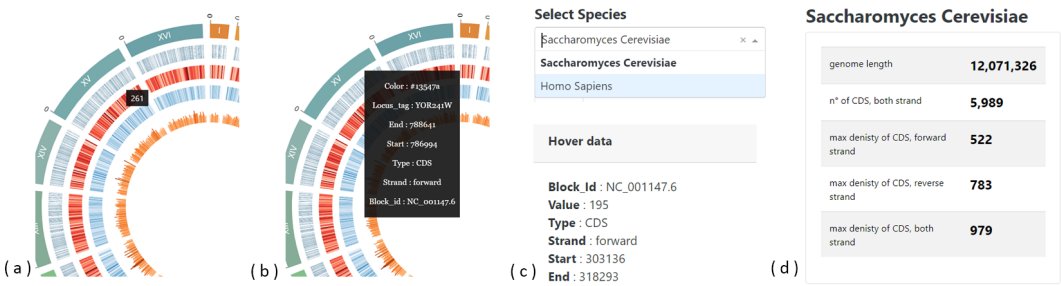

Figura 5.3: Dettagli dell'interfaccia: informazioni sulla densità (a) e sulle feature (b) mostrate al passaggio del mouse; menù a discesa per la selezione delle specie (c); pannello di riepilogo (d).

creazione e della modifica del grafico attuale, e ne riporta, per ogni traccia visualizzata, un **riassunto delle informazioni** rappresentate, oltre alla **lunghezza totale** del genoma. Le informazioni di riepilogo possono essere: il numero delle feature recuperate, ad esempio, il numero dei geni nel filamento forward; il valore massimo della densità delle feature.

Il pannello multi-schede sulla sinistra permette di operare differenti azioni sul grafico, e leggere ulteriori informazioni sui dati rappresentati. La scheda **Graph**, infatti, riporta ed estende tutti i dettagli disponibili sul componente individuato tramite mouse hover.

La Figura 5.4 mostra il pannello **Edit Graph** (a), dove è possibile rimuovere le tracce del grafico, selezionandole tramite un menu a discesa.

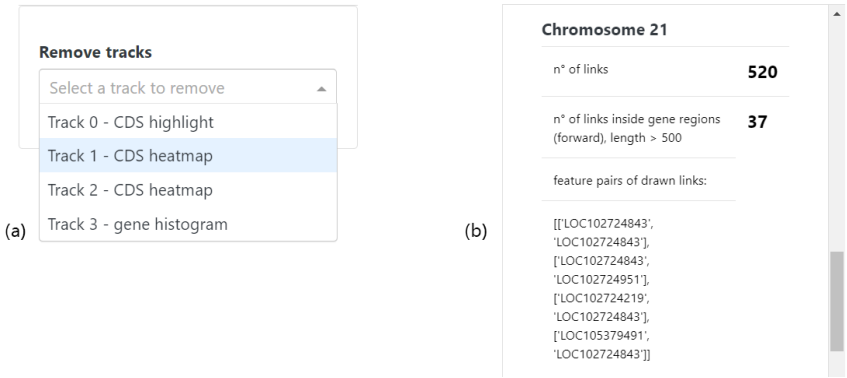

Figura 5.4: Dettagli del'interfaccia: pannelli *Edit Graph* (a) e *Links Info* (b).

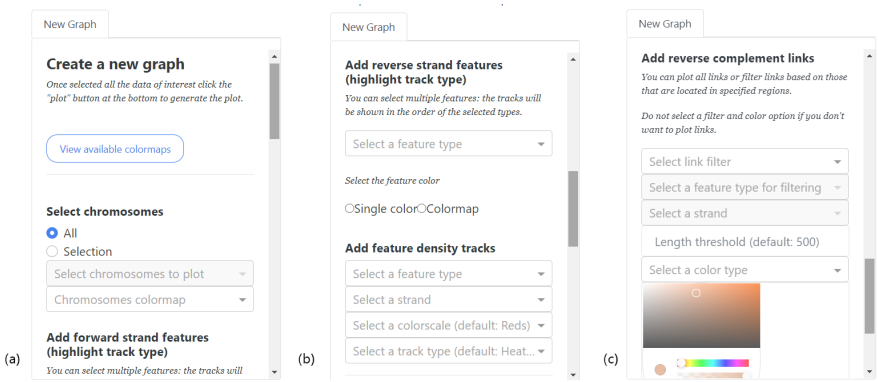

Figura 5.5: Dettagli dell'interfaccia: la scheda *New Graph* permette di selezionare i cromosomi da visualizzare (a), aggiungere feature e feature density (b) e rappresentare i reverse complement (c).

La scheda **Links info**, sempre visibile in Figura 5.4 (b), riporta **informazioni riassuntive sui link** rappresentati nel grafico. Se si visualizzano tutti i reverse complement, eventualmente filtrati per lunghezza, verranno riportati i numeri di link per ogni cromosoma. Nel caso in cui i link vengano filtrati in base alla posizione all'interno di caratteristiche sequenze, verranno mostrati ulteriori informazioni che specificano le coppie di feature evidenziate dai reverse complement.

L'ultima scheda, **New Graph**, permette la **creazione dinamica di nuovi grafici**. Alcuni dettagli del pannello sono visibili in Figura 5.5.

Per creare un nuovo grafico diverse informazioni devono essere fornite. La presenza di valori di default e testi esplicativi facilitano la selezione dei parametri del grafico.

La prima scelta effettuabile riguarda i **cromosomi da visualizzare**: possono essere rappresentati tutti i cromosomi (valore di default) oppure un loro sottoinsieme, specificabile tramite l'opzione *Selection* ed il menu a selezione multipla che verrà abilitato. Il colore dei cromosomi potrà essere scelto tra un insieme di colormaps supportate.

Sarà, quindi, possibile aggiungere le **feature**, specificandone il tipo, il filamento ed il colore o la colormap. Eventuali **tracce di densità** possono essere aggiunte tramite i comandi che seguono, e sarà necessario specificare: tipo di feature, filamento, colorscale e tipo di grafico (tra i disponibili *heatmap, histogram, scatter, line*).

Infine, vi è la sezione dedicata all'**aggiunta dei link reverse complement**. Il primo menu permette di scegliere se rappresentare tutti i link o filtrarli in base alla loro posizione. Nel secondo caso sarà necessario specificare il tipo di feature ed il filamento in cui ricercare le posizioni dei link. I link contenuti nel database verranno filtrati in base alla loro lunghezza per permettere maggiore leggibilità del grafico. Di default verranno recuperati i reverse complement di lunghezza maggiore o uguale a 500, ma questo valore può essere cambiato tramite un campo di input numerico. Il colore dei link può essere scelto tra tre possibili alternative: un singolo colore selezionabile; il colore del cromosoma in cui il link ha inizio, recuperato in modo automatico; il colore della feature in cui ricade il link, opzione presente solo per i reverse complement filtrati per posizione. In quest'ultimo caso sarà anche possibile richiedere di rappresentare le feature in questione, nel consueto stile di grafico *highlight*.

Una volta specificati tutti i parametri necessari, il grafico può essere generato tramite un bottone posto alla fine della scheda.

Capitolo 6

Implementazione e valutazione delle prestazioni

Verrà ora trattata l'implementazione dell'applicazione InstaCircos, seguendone le fasi di sviluppo che hanno portato alla realizzazione del progetto. La prima implementazione vede la scrittura di un codice per la generazione di grafici statici (Sezione 6.1), la cui gestione dei dati è stata ottimizzata grazie alla creazione del database (Sezione 6.2) L'integrazione della base di dati con il codice prodotto e ulteriori modifiche alle funzioni di visualizzazione hanno portato ad incrementare le performance del programma (Sezione 6.3). A questo punto è stato possibile implementare l'applicazione web interattiva (Sezione 6.4).

A conclusione del capitolo (Sezione 6.5) verranno esaminati i tempi di esecuzione dei vari codici prodotti e confrontati con il pacchetto Python pyCircos. Lo stesso verrà fatto per quanto riguarda l'occupazione di memoria.

6.1 Implementazione di base

La prima fase di sviluppo ha viso la creazione di un codice (che verrà chiamato *Prototipo 1*) per la generazione di grafici statici. Il primo obiettivo del progetto era, infatti, velocizzare questa operazione, rendendo possibile leggere e gestire anche genomi dalle grasse dimensioni, come il genoma umano.

In modo simile a quanto avviene nel pacchetto pyCircos si è sfruttata la libreria grafica Matplotlib per la creazione dei grafici circolari, ma, a differenza

di pyCircos, si è ottimizzata la gestione dei dati genomici, selezionando solamente quelli di interesse e memorizzandoli in un DataFrame Pandas.

Il codice è stato organizzato in due principali classi: **Genome** e **Circos**.

6.1.1 Lettura e memorizzazione dei dati genomici

La classe **Genome** contiene il codice per la lettura e la gestione dei file genomici e richiede, quindi, i riferimenti ai file GBFF e assembly report.

Come prima operazione viene letto il file assembly report, per identificare i loci che rappresentano cromosomi e quindi ricavare **identificativo**, **nome** e **lunghezza**, come mostrato nel codice in Figura 6.1:

```
with open(report,"r") as assembly:

    rows = [line.strip('\n').split('\t') for line in assembly
            if not line.startswith('#') and  line.split()]
    #dizionario di cromosomi id:nome_sequenza
    chrs = {row[id_col]:[row[0], int(row[8])] for row in rows
                if "chromosome" in row[3].lower()
                and "assembled-molecule" in row[1]}

self.chrs = pd.DataFrame.from_dict(chrs, orient='index', columns=['name','length'])
```

Figura 6.1: Codice per la lettura dei file assembly report (Prototipo 1).

Quando il file viene letto, si estraggono le righe, ignorando quelle di commento, facilmente individuabili dal carattere iniziale "#". I valori delle colonne vengono separati tramite la funzione **split()** ed ogni riga viene quindi rappresentata come una lista di stringhe. L'insieme delle righe selezionate è memorizzata nella variabile *rows*.

A questo punto è possibile ricercare, all'interno di esse, quelle che contengono informazioni riguardanti i cromosomi. Come riportato nella Sezione 4.3, l'individuazione dei cromosomi avviene grazie ai valori delle colonne *Assigned-Molecule* e *Sequence-Role* del file assembly, che corrispondono alle colonne di indici 3 ed 1 (con gli indici che partono da 0).

Solo per le righe che soddisfano i requisiti ne verranno estratte identificativo (valore della colonna con indice *id_col*), nome e lunghezza.

I dati così estratti sono memorizzati in un DataFrame Pandas, che risulterà simile al DataFrame in Figura 3.4.

```
if seq:

    for l in  SeqIO.parse(self.record,"genbank"):
        if l.id in self.chrs.index.tolist():
            self.seqs[l.id] = str(l.seq)


for l in SeqIO.parse(self.record,"genbank"):
    if l.id in self.chrs.index.tolist():
        self.features[l.id] = l.features
```

Figura 6.2: Codice per la lettura di feature e sequenze dai file GBFF (Prototipo 1).

L'estrazione delle feature e delle sequenze geniche avviene tramite l'utilizzo dell'interfaccia SeqIO della libreria Biopython. Come mostrato nel codice in Figura 6.2, si memorizzano, in due differenti dizionari, feature e sequenze dei soli cromosomi di interesse, individuandoli tramite gli identificativi contenuti nel DataFrame precedentemente creato. Inoltre, per ottimizzare tempi e allocazione di memoria, l'estrazione delle sequenze geniche verrà effettuata solamente se necessario, e quindi se specificato dalla variabile booleana *seq*.

Gli altri metodi contenuti nella classe *Genome* riguardano la lettura dei link e la gestione delle feature per il calcolo della densità.

6.1.2 Creazione del grafico statico

La classe **Circos** contiene il codice per la manipolazione dei dati genomici e quindi la creazione di grafici circolari. In quanto tale, una istanza della classe Genome dovrà essere precedentemente creata e passata nel momento nell'inizializzazione della classe Circos.

L'inizializzazione prevede l'impostazione dei parametri della figura sulla quale verrà creato il tracciato, come visibile in Figura 6.3.

Per realizzare il particolare formato dei grafici circolari sarà necessario utilizzare Matplotlib nei livelli più bassi della sua organizzazione gerarchica. Non esistono, infatti, funzioni del modulo *pyplot* in grado di generare tracciati che soddisfano i requisiti del progetto; ogni elemento del grafico dovrà essere generato singolarmente.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import math
import colorsys
import genome_nuovo as gen


class Circos():


    def __init__(self, genome, figsize=(10,10), gap=1.5, dpi=900):
        '''
        genome:
            Genome object to plot
        figsize:
            plot figure size (x,y)
        gap:
            size gap between locus/chromosomes
        dpi:
            dpi of the final plot to save
        '''

        plt.ioff()

        self.genome = genome
        self.chr_colors = {}
        self.fig = plt.figure(figsize=figsize, dpi=dpi)
        self.ax = self.fig.add_axes([0,0,1,1], polar=True)
        self.ax.axis('off')
        m = max(figsize)
        self.ax.set_ylim(0,m)

        self.radius = [min(figsize)]
        self.chr_height = 0

        self.len_per_degree = self.genome.tot_len / (360. - gap*self.genome.chrs.shape[0])

        self.len_per_theta = self.genome.tot_len / (np.pi*2 - np.deg2rad(gap)*self.genome.chrs.shape[0])

        cum_length = [0]+list(self.genome.chrs.length.cumsum())[:-1]

        self.genome.chrs['theta_s'] = [np.deg2rad(l/self.len_per_degree+gap*i)
                                       for i,l in enumerate(cum_length)]
```

Figura 6.3: Codice per l'inizializzazione della classe Circos (Prototipo 1).

A tale scopo, come riportato nella precedente Sezione 3.7, si crea un oggetto *pyplot.figure*, contenitore di tutti gli elementi grafici, a cui si aggiungono gli assi tramite la funzione *add_axes*. Il parametro *polar=True* rende gli assi polari, così che ogni elemento che verrà disegnato nella figura sarà posizionato in un tracciato circolare.

L'ultima operazione preliminare alla creazione dei grafici è il calcolo delle coordinate di inizio di ogni cromosoma, che vengono aggiunte al DataFrame come illustrato nella Sottosezione 3.5.2 e visualizzato nella Figura 3.5.

I metodi contenuti nella classe Circos sono finalizzati alla creazione del grafico e dovranno eseguire le seguenti operazioni: disegnare i cromosomi, le loro etichette e la scala delle lunghezze; rappresentare le feature selezionate nelle loro posizioni all'interno del genoma; disegnare la densità delle feature specificate; aggiungere i link rappresentanti i reverse complement. Sono, inoltre, presenti alcune funzioni di servizio per svolgere compiti il calcolo automatico del rag-

gio per le nuove tracce, il calcolo delle coordinate in radianti e il salvataggio dell'immagine.

In tutte queste funzioni il modulo numpy è frequentemente usato per effettuare operazioni sui dati contenuti del DataFrame Pandas, in particolare per gestire le coordinate di cromosomi ed altri dati da visualizzare. Un esempio può essere estratto dalla funzione *draw_labels* per l'aggiunta al grafico delle etichette dei cromosomi, riportato in Figura 6.4.

```
rotation=0
deg=np.rad2deg(self.get_theta(indexes,pos))
rotation=deg.copy()
rotation=np.add(rotation,90)
rotation=np.mod(rotation,360)
rotation=[r + 180 if 90< r<270 else r for r in rotation]

for t, p, d, rot in zip(lbls,pos, deg, rotation):
    plt.text(np.deg2rad(d), r, t, rotation=rot, verticalalignment='center',
             horizontalalignment='center', fontweight='bold')
```

Figura 6.4: Frammento di codice per l'aggiunta delle etichette ai cromosomi (Prototipo 1).

Come prima operazione è necessario ricavare i punti in cui aggiungere le scritte, che da radianti vengono trasformate in gradi grazie alla funzione *numpy.rad2deg*. L'uso diretto di queste coordinate creerebbe etichette disposte in modo radiale. Per aumentarne la leggibilità si applicano, quindi, delle rotazioni alle scritte. Come prima trasformazione si applica una rotazione di 90 gradi e alla lista così ottenuta viene applicata la funzione *numpy.mod* per ottenere valori compresi tra 0 e 360. È ora possibile applicare un'ultima rotazione di 180 gradi alle etichette che si trovano nella parte superiore del grafico che sarebbero, altrimenti, alla rovescia.

Riguardo l'aggiunta dei componenti grafici, la densità delle feature, a seconda dello stile di grafico selezionato, verranno sfruttate le funzioni *bar*, *scatter*, *heatmap* e *fill_between* dell'oggetto **Axes** (gli assi della figura).

Si riporta in Figura 6.5 il metodo per l'aggiunta al grafico delle posizioni delle feature. Il suo funzionamento si basa sulla scansione, per ogni cromosoma, di tutte le sue feature. Ogni regione caratteristica che soddisfa i requisiti specificati viene aggiunta al grafico tramite la funzione *Axes.bar()*, in cui si specifica: coordinata d'inizio (in radianti), altezza della barra, raggio, larghezza della barra, colore e spessore del bordo.

```
def draw_features(self, r=None, feat_type='gene', strand=None, height=0.9,
                  color='#4287f5',second_color=False,requirement=lambda x:1):

    if r==None:
        r=self.compute_radius(height)
    self.radius.append(r)

    if strand == 'forward':
        strand_check = lambda x: x == 1
    elif strand == 'reverse':
        strand_check = lambda x: x == 1
    else:
        strand_check = lambda x: 1
        shade = self.get_color_shade(color) if second_color else color


    for chrid in self.genome.chrs.index:
        for feat in self.genome.features[chrid]:
            if feat.type==feat_type and requirement(feat) and strand_check(feat.strand) :
                c = color
                if (strand == None):
                    c = color if feat.strand == 1 else shade;
                s= self.get_theta(chrid, feat.location.start)
                e= self.get_theta(chrid, feat.location.end)
                self.ax.bar([s],[0,height], bottom=r, width=e-s, color=c, linewidth=0)
```

Figura 6.5: Metodo per l'aggiunta delle feature al grafico (Prototipo 1).

Questa funzione sarà oggetto di ottimizzazione nelle successive fasi di sviluppo dell'applicazione.

6.2 Implementazione del database

Grazie alla selezione delle sole sequenze cromosomiche e della gestione dei dati implementata nel Prototipo 1 del progetto è stato possibile generare grafici del genoma umano; cosa che non era possibile effettuare con la libreria pyCircos, a causa di errori di memoria.

Le performance del primo prototipo, tuttavia, erano ancora scarse: la generazione del grafico del genoma umano richiedeva circa quaranta minuti. La maggioranza del tempo di esecuzione veniva impiegato per la lettura e la memorizzazione dei file genomici.

La progettazione e quindi l'implementazione di un database che organizzasse i dati genomici sono state operazioni fondamentali alla riduzione dei tempi di esecuzione e dell'occupazione di memoria.

In questa sezione ne verrà trattata l'implementazione.

6.2.1 Creazione dei database MongoDB

Per ogni genoma di interesse verrà creata una istanza di database MongoDB, come esposto in dettaglio nella Sezione 4.4.

La creazione dei database e delle collezioni può essere effettuata tramite la shell di MongoDB, Mongo shell, oppure attraverso una GUI (Graphical User Interface) quali l'open-source Robo 3T o Compass, la GUI offerta da MongoDB.

Di seguito si farà riferimento ai comandi shell della versione 4.2.5.

Per creare una istanza di database MongoDB, dal nome *homo_sapiens* è possibile eseguire il seguente comando:

```
use homo_sapiens
```

In questo modo, se non ancora esistente, il database *homo_sapiens* viene creato e impostato come database corrente; tutte le operazioni che verranno eseguite sulla parola chiave **db** saranno applicate all'istanza *homo_sapiens*.

La creazione delle collezioni è effettuabile in due modalità: tramite l'uso di un comando esplicito oppure in modo indiretto, nel momento dell'inserimento dei dati.

Il comando in questione è:

```
db.createCollection("chromosome")
```

dove *chromosome* è il nome della collezione. A differenza della creazione automatica, il comando *createCollection* permette , opzionalmente, di specificare le impostazioni della collezione tramite ulteriori parametri, come, ad esempio, uno schema di validazione.

Non dovendo specificare particolari impostazioni per le collezioni dei dati genomici la loro creazione verrà effettuata nel momento dell'inserimento dei dati.

6.2.2 Creazione degli indici

Al fine di ottimizzare la ricerca dei dati e, conseguentemente, la creazione dei grafici, si sono analizzate le interrogazioni che dovranno essere eseguite per recuperare i dati di interesse.

Le query che verranno richieste sono le seguenti:

- *db.chromosome.find('sequence_role':'chromosome')* - La collezione **chromosome** verrà principalmente interrogata per recuperare tutte le informazioni riguardanti le sequenze cromosomiche, ovvero le sequenze di interesse per il grafico;

- *db.chromosome.find('id':valore)* - Eventuali altre ricerche nella collezione "chromosome" implicheranno la ricerca di una determinata sequenza tramite il suo identificativo;

- *db.feature.find('chr_id': id, 'type': tipo)* - I documenti della collezione **feature** saranno recuperati secondo due modalità. La prima consiste nella ricerca delle feature di un particolare cromosoma (*chr_id*) e di un determinato tipo (*type*);

- *db.feature.find('chr_id': id, 'type': tipo, 'strand': strand)* - La seconda interrogazione sottoposta alla collezione "feature" consiste nell'aggiunta, rispetto alla query precedente, dello specifico filamento su cui le feature si devono trovare (*strand*);

- *db.links.find('start.id': id)* - I link attualmente memorizzati nel database sono tutti reverse complement, ovvero link intra-cromosoma. Una prima possibile interrogazione a questi dati riguarda, quindi, la richiesta dei link di un certo cromosoma;

- *db.links.find('start.id': chr_s, 'start.length': '$gte': len)* - Un'alta possibile interrogazione della collezione "links" riguarda l'aggiunta di una soglia di lunghezza, tramite l'operatore **$gte**;

Per ognuna di queste interrogazioni verrà creato un opportuno indice al fine di ottimizzarne l'esecuzione.

La collezione *seq*, contenente le sequenze geniche di ogni locus, potrà essere interrogata in base all'identificativo della sequenze stessa, che andrà a sostituire il campo *_id* dei documenti MongoDB. In questo modo non sarà necessario creare ulteriori indici per la collezione, ma basterà l'indice creato in modo automatico da MongoDB.

Concludendo, i comandi per la creazione degli indici sono i seguenti:

```
db.chromosome.createIndex({'sequence_role' : 1})
```

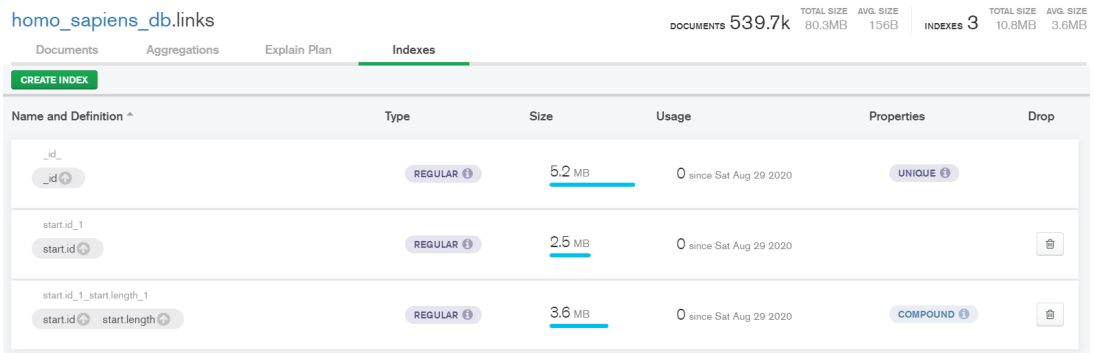

Figura 6.6: Panoramica degli indici della collezione *links*.

```
db.chromosome.createIndex({'id' : 1})
db.feature.createIndex({'chr_id' : 1, 'type' : 1})
db.feature,createIndex({'chr_id' : 1, 'type' : 1, 'strand' : 1})
db.links.createIndex({'start.id' : 1})
db.links.createIndex({'start.id' : 1, 'start.length' : 1})
```

Gli indici costruiti su più di un attributo sono **compound index**.

La Figura 6.6 mostra la panoramica degli indici della collezione *links*, ottenuta da MongoDB Compass. Oltre ai due indici creati manualmente si osserva la presenza dell'indice sul campo *_id*, creato in modo automatico da MongoDB.

6.2.3 Popolamento dei database

L'inserimento dei dati genomici nei database avviene grazie ad un codice Python che sfrutta le librerie **Biopython** e **PyMongo** [2], un insieme di strumenti per lavorare con MongoDB in Python. La libreria Biopython servirà a leggere i file GBFF che, insieme ai dati letti dai file assembly, verranno organizzati in un dizionario Python per render possibile l'inserimento nel database, come documento MongoDB. Verranno impiegate le funzioni *insert_one* e *update_one* di PyMongo per creare e aggiornare i documenti delle collezioni.

La Figura 6.7 riporta l'uso delle suddette librerie e funzioni per la lettura e l'inserimento dei dati genomici all'interno della collezione *chromosome*.

```
def populate_chromosome(record, report, collection):
     for l in  SeqIO.parse(record,"genbank"):
        chrid = l.id
        annotations = l.annotations
        description = l.description

        chr_doc = {
            'id':chrid,
            'annotations':{k:v for k,v in annotations.items() if k != 'references'},
            'description':description,
        }

        collection.insert_one(chr_doc)

     id_col = 6

     with open(report,"r") as assembly:
            rows = (line.strip('\n').split('\t') for line in assembly
                    if not line.startswith('#') and  line.split())
            chrs = {row[id_col]:[row[0],'chromosome' if 'chromosome' in row[3].lower()
                    and "assembled-molecule" in row[1] else row[1], int(row[8])
                    ] for row in rows }

     for i in chrs.items():
         collection.update_one({'id':i[0]},{'$set':{'name':i[1][0],
                                'sequence_role':i[1][1], 'length':i[1][2]}}, upsert=True)
```

Figura 6.7: Codice per la lettura e l'aggiunta dei dati genomici alla collezione *chromosome*.

I dati contenuti nei file GBFF vengono estratti ed inseriti nella collezione. In un secondo momento viene letto il file assembly report per recuperare i rimanenti dati, che verranno inseriti nei rispettivi documenti grazie alla funzione *update_one*. Il criterio di selezione dei documenti da aggiornare è l'identificativo del locus. I file GBFF contengono, infatti, un insieme di record, riportanti ciascuno un univoco locus genetico caratterizzato dal proprio identificativo. Le informazioni principali di ognuno di questi loci sono riportate nel file assembly report, e la ricerca di un documento tramite l'identificativo della sequenza deve quindi restituire un unico risultato.

In modo analogo saranno aggiunti, alle rispettive collezioni, i dati delle feature e dei link.

La collezione *seq* necessita, tuttavia, un'ulteriore accortezza. Le sequenze geniche possono, infatti, essere molto lunghe e quindi superare il limite imposto da MongoDB sulla dimensione dei documenti.

Per memorizzare dati di grosse dimensioni è possibile sfruttare la specifica **GridFS**. Questo sistema permette di gestire dati dalle dimensioni maggiori di 16MB grazie alla loro suddivisione ed organizzazione in due differenti collezioni. In particolare, i file vengono suddivisi in parti, detti *chunk*, da 255kB (ad eccezione dell'ultimo frammento che occuperà solo lo spazio necessario) e memorizzate

```
_id: ObjectId("5f4832db8f030dba7351a1c0")
files_id: "NC_000001.11"
n: 0
data: Binary('Tk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5OTk5O...', 0)
```

```
_id: ObjectId("5f4832db8f030dba7351a1c1")
files_id: "NC_000001.11"
n: 1
data: Binary('QUFUVENDVEdDQUdHQUdDQUdHVEFBVFRBQUFBQUdBR1RDVEdHQ1RUQ0NUVEdHQ1RUQ0NDVENUVEdDVFRUQ0FDVFRDVEdDVEFUR1RH...', 0)
```


Figura 6.8: Panoramica MongoDB Compass dei documenti memorizzati nella collezione *fs.chunks*.

in documenti separati nella collezione **fs.chunks**.

La collezione **fs.files** conterrà un documento per ogni file (originale), memorizzandone i metadati. Il valore del campo **_id** di ogni documento in questa collezione è riportato in tutti i suoi chunk (campo **files_id**) che, grazie agli indici creati in modo automatico, ne permette un efficiente recupero.

La Figura 6.8 mostra alcuni documenti contenuti nella collezione fs.chunks del database *homo_sapiens*.

I tre documenti sono parti della suddivisione della sequenza genica del primo cromosoma umano. L'identificativo della sequenza è riportato in ogni chunk nel campo *files_id*, mentre il campo *n* indica il numero di sequenza del frammento, che sarà fondamentale per ricomporre il file originale.

Si conclude questa Sezione riportando, in Figura 6.9, una panoramica delle collezioni di una istanza di database MongoDB in seguito ad inserimento dei dati genomici.

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
chromosome	639	2.0 KB	1.2 MB	3	116.0 KB
feature	386,562	946.9 B	349.1 MB	3	7.4 MB
fs.chunks	12,895	247.9 KB	3.0 GB	2	320.0 KB
fs.files	639	140.2 B	87.5 KB	2	80.0 KB
links	539,719	156.0 B	80.3 MB	3	10.8 MB

Figura 6.9: Panoramica MongoDB Compass di tutte le collezioni del database.

```
def __init__(self, db, requirement=None, exclude=None):

    self.db=db

    if requirement == None:
        cursor = db.chromosome.find({'sequence_role':'chromosome'},
                                    {'annotations':0, '_id':0})
    else:
        cursor = db.chromosome.find(requirement, exclude)

    self.chrs = pd.DataFrame(list(cursor))
    self.chrs = self.chrs.set_index('id')
```

Figura 6.10: Codice per l'inizializzazione della classe *Genome* (Prototipo 2).

6.3 Integrazione del database e ottimizzazione del codice

La prima e fondamentale operazione che ha permesso di migliorare le prestazione del codice è stata l'integrazione del database (Sottosezione 6.3.1). Oltre a velocizzare i tempi di esecuzione grazie all'uso di interrogazioni mirate sui dati indicizzati, l'utilizzo del database ha permesso di diminuire l'occupazione di memoria.

In seguito si è analizzato il codice al file di applicare ulteriori ottimizzazioni e portare il tempo di esecuzione ad un valore accettabile per una applicazione web (Sottosezione 6.3.2).

6.3.1 Integrazione del database

Il Prototipo 2 è stato, quindi, realizzato integrando MongoDB al codice del Prototipo 1. L'integrazione del database è stata resa possibile dalla stessa libreria utilizzata per l'inserimento dei dati, PyMongo, e le modifiche al codice sono state minimali.

La quasi totalità delle modifiche hanno riguardato la classe *Genome*. Il metodo di inizializzazione è completamente adattato all'uso del database. Come mostrato nel codice in Figura 6.10, il nuovo codice risulta molto più semplice: tramite una singola interrogazione alla base di dati si ottengono tutte le informazioni riguardanti i cromosomi da disegnare nel grafico. I campi *annotations* e *_id* vengono esclusi in quanto attualmente non utili ai fini rappresentativi, mentre i rimanenti dati (nome, lunghezza, ruolo e descrizione) vengono, come nel

Prototipo 1, memorizzati in un DataFrame, indicizzati secondo l'identificativo delle sequenze.

Altre modifiche sono state effettuate ai metodi che gestiscono i dati (link e feature), sostituendo la ricerca delle informazioni all'interno delle variabili (Prototipo 1) con semplici interrogazioni al database (Prototipo 2). In particolare, le query aggiunte al codice sono quelle analizzate nella Sottosezione 6.2.2. La loro esecuzione è resa efficiente dagli indici costruiti sulle varie costruzioni, come mostrano i risultati riportati in Figura 6.11.

Collezione interrogata (criteri di ricerca)	Esecuzione query (secondi)	n° documenti recuperati
chromosome (sequence_role)	0,000183	24
feature (chromosome, type)	0,000094	2557
links (start.id)	0	14095

Figura 6.11: Tempi di esecuzione delle query più frequenti.

Anche la gestione delle feature risulta essere più semplice ed efficiente rispetto a quanto era necessario fare nel Prototipo 1. Facendo riferimento al codice in Figura 6.5, infatti, con le feature memorizzate in un dizionario Python, la ricerca di un particolare tipo (ed eventuale filamento) necessitava la lettura di tutte le feature del cromosoma in questione. Con l'integrazione del database è sufficiente una query mirata, come riporta il metodo in Figura 6.12.

Tutte i rimanenti metodi per la creazione del grafico non hanno subito cambiamenti.

```
def get_chrFeatures (self, chrid, feat_type='CDS', strand=None):

    if strand == 'forward' or strand == 'reverse':
        strand = 1 if strand == 'forward' else -1
        cursor = self.db.feature.find({'chr_id':chrid, 'type':feat_type, 'strand':strand})
    else:
        cursor = self.db.feature.find({'chr_id':chrid, 'type':feat_type})
    return cursor
```

Figura 6.12: Metodo per il recupero delle feature (Prototipo 2).

6.3.2 Ottimizzazione delle funzioni grafiche

L'uso del database ha portato il tempo di esecuzione di un generico grafico del genoma umano da circa 40 minuti ad una media di 10 minuti. Nonostante l'evidente incremento delle performance, i tempi di esecuzione del Prototipo 2 non lo rendono ancora adatto ad una applicazione web.

Si è quindi eseguito il profiling del codice al fine di individuare le funzioni più lente e potenzialmente ottimizzabili. La tabella in Figura 6.13 mostra i tempi di esecuzione delle principali funzioni del codice che ha generato il grafico di esempio.

Funzione	Tempo (secondi)
Creazione classe Genome	0,015991
Creazione classe Circos	0,132951
Visualizzazione cromosomi	0,239013
Visualizzazione feature	189,1152
Calcolo feature density	1,328091
Visualizzazione feature density	2,261887
Salvataggio	159

Figura 6.13: Tempi di esecuzione del Prototipo 2.

Sono facilmente individuabili le due operazioni che richiedono la maggioranza del tempo di esecuzione: l'**aggiunta delle feature** e il **salvataggio**.

Nell'ottica di una applicazione web, il tempo impiegato per il salvataggio dell'immagine è trascurabile rispetto all'operazione di aggiunta delle feature. Il tempo di salvataggio, infatti, non è sperimentato dall'utente nel momento della generazione dell'immagine, ma solo se decide di scaricare il grafico creato.

Il metodo che si cerca di ottimizzare è l'aggiunta delle feature.

La lentezza dell'operazione è dovuta all'elevato numero di chiamate alla funzione grafica Matplotlib che si utilizza (Figura 6.5). Per ogni sequenza caratteristica recuperata si calcolano le coordinate in cui porre la feature e si invoca il metodo *Axes.bar()* per la sua rappresentazione. Per visualizzare i geni contenuti nel filamento reverse di ogni cromosoma umano, queste operazioni dovranno essere ripetute per un totale di 26800 volte.

```
def draw_features(self, r=None, feat_type='gene', strand=None, height=0.5,
                  color=None, cmap=None, chrid_color=None):

        if r == None:
            r = self.compute_radius(height)

        if color != None and cmap == None:
            col = color
        elif color == None and cmap != None:
            cm = plt.cm.get_cmap(cmap)
            num = cm.N if cm.N < 256 else 20
            col = cm(np.linspace(0, 1, num))
        else:
            col = self.def_colors

        genColors = self.gen_colors(col)

        points = []
        colors = []
        widths = []
        line_points = []
        line_colors = []
        line_widths = []

        for chrid in self.genome.chrs.index:
            cursor = self.genome.get_chrFeatures(chrid, feat_type=feat_type,
                                                 strand=strand)

            for feat in cursor:
                c = next(genColors)
                s = self.get_theta(chrid, feat['start'])
                e = self.get_theta(chrid, feat['end'])
                if (e - s) > 5.0e-3:
                    points.append(s)
                    widths.append((e - s))
                    colors.append(c)
                else:
                    line_points.append([(s, r), (s, r + height)])
                    line_widths.append((e - s))
                    line_colors.append(c)
        try:
            line_widths = line_widths / max(line_widths)
        except ValueError:
            pass

        line_segments = LineCollection(line_points, linewidths=[l + 0.4 for l in line_widths],
                                       colors=line_colors)
        self.ax.add_collection(line_segments)
        self.ax.bar(points, height, bottom=r, width=widths, color=colors,
                    align='edge', linewidth=0)
```

Figura 6.14: Metodo per la renderizzazione delle feature (Prototipo 2).

Una soluzione al problema è stato trovato osservando l'aspetto finale del grafico: maggiore è il numero delle feature da rappresentare, minore sarà la loro dimensione nel grafico, arrivando ad essere rappresentati da sottili linee. Una volta calcolati i punti d'inizio e di fine in cui le sequenze saranno disegnate nella circonferenza, è possibile individuare una dimensione di soglia sotto la quale adottare una funzione grafica differente.

In seguito ad uno studio più approfondito dei metodi disponibili a questo scopo si è individuata la classe **LineCollection** di Matplotlib, appartenente al modulo *collections*: un insieme di classi per disegnare in modo efficiente grandi quantità di oggetti (linee, poligoni, eccetera).

La Figura 6.14 mostra il codice ottimizzato per disegnare le feature. I dati vengono suddivisi in due insiemi, in base alla lunghezza del tratto di circonfe-

renza in cui dovranno essere rappresentati. I segmenti dalle dimensioni minori saranno renderizzati grazie alla classe **LineCollection** e quindi aggiunti al grafico tramite la funzione *Axes.add_collection*; le feature con dimensione superiore alla soglia verranno aggiunte al grafico per mezzo della funzione *Axes.bar* usata nel Prototipo 1.

Grazie a questa modifica, il tempo di esecuzione del codice scende, complessivamente, a meno di un minuto. L'aggiunta delle feature passa dai circa 200 secondi precedenti a **20 secondi**, e anche il tempo di salvataggio diminuisce in seguito alla modifica, arrivando a **14 secondi**.

6.4 Implementazione dell'applicazione web

Il codice sviluppato nel secondo Prototipo del progetto ha permesso la creazione di grafici di **grandi genomi** in un **tempo breve**, accettabile per le necessità di una applicazione web.

La terza fase di implementazione ha quindi visto la creazione dell'applicazione web InstaCircos: una piattaforma semplice da usare e veloce anche nella gestione di grossi genomi.

6.4.1 Struttura e creazione dell'applicazione Dash

Il codice principale dell'applicazione web, in cui è presente la struttura e il funzionamento della pagina, è contenuto nel file *app.py*. Altri file, contenenti codice dalle specifiche funzioni, sono organizzati in due differenti cartelle, come mostra la Figura 6.15.

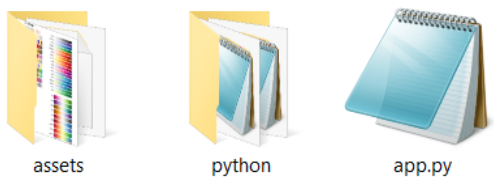

Figura 6.15: Struttura dell'applicazione web

In particolare, la cartella *assets* contiene il **file CSS** per la definizione dello stile di alcune componenti della grafica della pagina web. Dash, infatti, individua automaticamente eventuali file CSS e JavaScript se posti in una cartella denominata in tal modo.

La cartella *python*, invece, contiene i seguenti tre file:

- *connection.py* contiene il codice per la gestione dell'integrazione di **MongoDB**: creazione della connessione, recupero delle collezioni e chiusura della connessione;
- *data.py* è il file contenente la **gestione dei dati genomici** così come faceve la classe *Genome* dei prototipi precedenti;
- *chart.py* contiene il codice per la manipolazione dei dati e il loro utilizzo per **la creazione dei grafici**.

Si sottolinea come le funzioni del file *data.py* siano fondamentalmente uguali ai metodi della classe *Genome*. Il recupero dei dati dall'istanza di database MongoDB selezionata rimane, infatti, invariato.

La creazione dei grafici, tuttavia, differisce rispetto al codice della classe *Circos* del Prototipo 1 e 2. La libreria Matplotlib, infatti, non permette la creazione di grafici interattivi ed è quindi stata sostituita dall'uso della suite **Dash Bio**, il cui componente *Circos* rispetta i requisiti per la rappresentazione dei dati genomici nell'applicazione web.

Maggiori dettagli su questa parte di codice verranno mostrati nella Sottosezione 6.4.3.

6.4.2 Implementazione delle funzionalità

Prima di esporre i dettagli dell'implementazione dell'applicazione web si riassume brevemente come è stato possibile implementare le funzionalità individuate nella Sezione 5.2.

- La visualizzazione dei dati in forma circolare, l'interattività del grafico e la possibilità di scaricare il tracciato creato sono state implementate grazie all'uso di Dash Bio. Dash gestisce, infatti, l'interattività di base (zoom, informazioni a seguito di mouse hover) e il download dell'immagine;

- La selezione di differenti genomi, anche di grandi dimensioni, è stata implementata grazie all'integrazione del database MongoDB;
- La creazione dinamica di nuovi grafici e la visualizzazione di informazioni riassuntive sono state aggiunte grazie all'implementazione di funzioni callback dedicate a tali scopi.

6.4.3 Creazione dei grafici interattivi

Come specificato precedentemente, la creazione dei grafici interattivi per l'applicazione web avverrà grazie alla componente **dash_bio.Circos**, che rappresenta ideogrammi circolari.

Per creare un grafico circolare è quindi necessario definire due principali componenti: il **layout** dell'ideogramma (i cromosomi da rappresentare) e le **tracks** (tracce di dati aggiuntivi in vari formati grafici).

Layout e tracks richiedono dati in un determinato formato: una lista di dizionari Python dalle specifiche chiavi. Questo formato è facilmente ottenibile da un DataFrame Pandas tramite la semplice ridefinizione delle colonne e l'uso della funzione *pandas.DataFrame.to_dict*.

La Figura 6.16 mostra la funzione per la creazione dei grafici tramite la componente *dash_bio.Circos*.

La variabile *df_chart* contiene un DataFrame di dati genomici le cui colonne memorizzano identificativo, nome, lunghezza e colore di ogni cromosoma. Questi dati sono trasformati in una lista di dizionari le cui coppie chiave-valore riportano il nome della colonna e il suo valore, ad esempio:

```
[ {'id': 'NC_001133.9', 'label': 'I', 'len': 230218,
'color': '#de8639'}, {'id': 'NC_001134.8', 'label': 'II',
'len': 813184, 'color': '#e09b41'}, {'id': 'NC_001135.5',
'label': 'III', 'len': 316620, 'color': '#e8ae49'} ]
```

Gli altri parametri specificati per la creazione del grafico sono: *id*, l'identificativo del componente grafico per la gestione dell'interattività tramite le callback; *size*, la dimensione del grafico; *selectEvent*, un dizionario le cui chiavi numeriche

```python
def draw_chr(df_chart, size, r=None, height=20, lbl_offset=7, lbl=True, chr_ticks=True):

    den = Chart.compute_gap(min(df_chart['len'])) if chr_ticks else 1
    rad = r if r is not None else (size / 2) - 70 - height
    g2 = dashbio.Circos(
        layout=df_chart.to_dict(orient="records"),
        size=size,
        id='main-circos',
        enableZoomPan=True,
        selectEvent={'0': 'both'},
        config={
            'innerRadius': int(rad),
            'outerRadius': int(rad + height),
            'ticks': {'display': True,
                      'labelDenominator': den},
            'labels': {
                'position': 'center',
                'display': lbl,
                'size': 12,
                'color': '#fff',
                'radialOffset': lbl_offset,
            },
        },
        tracks=[],
    )
    return g2
```

Figura 6.16: Creazione del grafico circolare (InstaCircos).

corrispondono alle tracce aggiunte al grafico e ne specificano l'evento (*hover, click, both*) che permette di ottenere informazioni sui dati rappresentati.

Il parametro *tracks* andrà a contenere i dati necessari all'aggiunta delle tracce.

La figura 6.17, ad esempio, riporta il codice per l'aggiunta delle tracce di densità. In primo luogo è necessario recuperare le feature e calcolarne la densità (funzione *get_all_FeatureDensity* contenuta nel file *data.py*). Viene quindi fatto un controllo sui dati: se non è stata recuperata alcuna feature si aggiungono ai dati della traccia delle informazioni incomplete, che quindi non verranno visualizzate, ma che saranno utili al rendering del pannello riassuntivo. La traccia viene quindi creata come un dizionario Python (variabile *track*), specificando i dati da rappresentare, lo stile grafico da adottare (parametro *type*) e specifiche di rappresentazione (parametro *config*), che possono variare in base al tipo di grafico scelto. Infine, la traccia viene aggiunta all'istanza di *dash_bio.Circos*.

Similmente avverrà per l'aggiunta al grafico delle tracce riportanti le posizioni delle feature e i link dei reverse complement.

```python
def add_feature_density(graph, db, df_chart, window, r_inner, r_outer, feat_type='CDS', strand=None,
                        block_type='HEATMAP', color='Blues', log=False, tooltip=None, block_id=None):

    if tooltip is None:
        tooltip = {'name': 'value'}
    position = True if block_type == 'SCATTER' or block_type == 'LINE' else False
    data = Genome.get_all_FeatureDensity(db, df_chart, window, feat_type=feat_type, strand=strand,
                                         position=position)
    num = len(graph.tracks)
    if tooltip is None:
        tooltip = {'name': 'value'}
    if len(data) == 0:
        if strand is None:
            strand_info = 'both'
        else:
            strand_info = strand
        data.append({'type': feat_type, 'strand': strand_info, 'first_id': list(df_chart.id)[0]})

    if block_type.upper() == 'LINE':
        color = plt.cm.get_cmap(color)
        color = color([0.8])

    track = {'data': data,
             'type': block_type,
             'config': {
                 'innerRadius': r_inner,
                 'outerRadius': r_outer,
                 'color': color,
                 'tooltipContent': tooltip,
             }}
    if block_id is not None:
        track['id'] = block_id

    if block_type.upper() == 'HEATMAP':
        track['config']['logScale'] = log

    if block_type.upper() == 'SCATTER':
        track['config']['shape'] = 'circle'
        track['config']['size'] = 14
        track['config']['strokeWidth'] = 0.2,
        track['config']['strokeColor'] = 'grey'

    graph.tracks.append(track)

    if block_type == 'LINE':
        graph.selectEvent[num] = None
    else:
        graph.selectEvent[num] = 'both'
```

Figura 6.17: Codice per l'aggiunta di tracce di densità (InstaCircos).

Riguardo a questi ultimi dati, InstaCircos mette a disposizione la funzionalità di rappresentare i soli **link** che si trovano **all'interno di caratteristiche sequenze** del genoma, e, quindi, di ottenere informazioni sulle feature in questione. Al fine di filtrare i reverse complement, delle analisi preliminari dovranno essere svolte prima dell'effettiva aggiunta al grafico.

In particolare, si recuperano dal database i link e le feature di interesse, estraendo, di queste ultime, le coppie di coordinate inizio-fine. Per ogni reverse complement si ricerca la presenza di eventuali feature che ne contengano completamente la regione di inizio e di fine del link. In caso positivo, le informazioni sulla coppia di sequenze vengono memorizzate insieme ai dati del link stesso, per poterli mostrare nel pannello informativo.

6.4.4 Contenitori nascosti

Prima di trattare le funzioni callback per l'implementazione dell'interattività della pagina web, si descrive brevemente l'uso dei contenitori (*div*) nascosti.

Le applicazioni Dash prevedono l'esistenza di una istanza di applicazione, completa di layout, e di funzioni callback per gestirne l'interazione. Queste funzioni modificano degli elementi html presenti nel layout dell'applicazione in base a determinati input ed eventi.

Può essere necessario, nella scrittura di queste funzioni, passare od ottenere dati da altre callback. Dash, tuttavia, essendo pensato e sviluppato per funzionare in ambienti multi-user, sconsiglia fortemente l'uso di variabili globali, che potrebbero portare ad un malfunzionamento dell'applicazione.

Per risolvere il problema della condivisione dei dati tra funzioni callback è possibile adottare, come è stato fatto per il progetto InstaCircos, il meccanismo dei **contenitori nascosti** (*hidden div*). Il suo funzionamento prevede la creazione di una funzione callback per processare e manipolare i dati di interesse, che verranno poi serializzati in JSON e passati in output al contenitore nascosto. In quanto tale, il componente *div* non sarà visibile dall'utente dell'applicazione, e al tempo stesso rimane accessibile ad altre funzioni callback.

Nell'applicazione web InstaCircos sono stati usati due contenitori nascosti per la memorizzazione dei dati genomici della specie in visualizzazione.

6.4.5 Callback per l'interattività

Numerose solo le funzioni callback implementate per rendere interattiva l'applicazione web InstaCircos.

Oltre alle funzioni per l'aggiornamento dei contenitori nascosti sono presenti callback per la creazione del pannello riassuntivo; la visualizzazione dei dati in seguito ad eventi di *mouse hover* sui componenti del grafico; l'aggiornamento della scheda riportante maggiori informazioni sui link; la modifica del grafico stesso.

Si evidenza che i componenti html presenti nel layout dell'applicazione possono possedere un'unica funzione callback che li modifichi.

```python
@app.callback(
    Output('links-info-div', 'children'),
    [   Input('chart-df', 'children'),
        Input('main-circos', 'tracks'),
        Input('circos-hold', 'children'),
        Input('create-btn', 'n_clicks'),
    ],
    [
        State('genome-df', 'children'),
    ]
)
def render_links_tab(_, tracks, __, ___, genome):
    genome_df = pd.read_json(genome)
    content = []

    if tracks is not None and len(tracks) != 0:
        for i, t in enumerate(tracks):
            if t['type'] == 'CHORDS':
                try:
                    feat_type = t['feat-type']
                except KeyError:
                    feat_type = None
                try:
                    tot_links = t['tot-links']
                except KeyError:
                    tot_links = None
                if feat_type is not None:
                    content.append(html.H6(f'Chromosome {genome_df.name[t["data"][0]["source"]["id"]]}'))
                    rows = [html.Tr([html.Td(f'n° of links '),
                                     html.Td('{:,.0f}'.format(t['tot-links']), style={'fontWeight': 'bold',
                                                                                      'fontSize': 'large'})]),
                            html.Tr(
                                [
                                    html.Td(
                                        f'n° of links inside {t["feat-type"]} regions ({t["feat-strand"]}),'
                                        f' length > {t["len-threshold"]}'),
                                    html.Td('{:,.0f}'.format(t['links-inside-threshold']), style={'fontWeight': 'bold',
                                                                                                  'fontSize': 'large'})
                                ]
                            ),
                            html.Tr(
                                [
                                    html.Td(f'feature pairs of drawn links:'),
                                ]
                            ),
                            html.Tr(
                                [
                                    html.Td(f'{t["pairs"]}'),
                                ]
                            ),
                            ]
                    content.append(dbc.Table(html.Tbody(rows), striped=False, className='table-card'))
                    content.append(html.Br())
                elif tot_links is not None:
                    content.append(dbc.Table(html.Tbody(html.Tr(
                        [
                            html.Td(f'n° of links with source {genome_df.name[t["data"][0]["source"]["id"]]}'),
                            html.Td('{:,.0f}'.format(tot_links), style={'fontWeight': 'bold',
                                                                        'fontSize': 'large'})
                        ]
                    ), )))

    if len(content) == 0:
        content.append(html.H6('No information available'))

    panel = dbc.CardBody(html.Div(content))
    return panel
```

Figura 6.18: Codice per l'aggiornamento del pannello informativo sui link (InstaCircos).

InstaCircos permette, attualmente, di modificare la dimensione del grafico, rimuovere tracce presenti e creare un grafico completamente nuovo. Queste modifiche sono scatenate da differenti elementi html, ed è, quindi, stato necessario gestire tutti i differenti input in una sola funzione, sfruttando i valori di ingresso per identificare l'operazione da svolgere per modificare il grafico.

A scopo illustrativo si riporta, in Figura 6.18 (pagina precedente), il codice della funzione callback per il rendering del pannello informativo sui link eventualmente rappresentati nel grafico (scheda *Links info* visibile nel pannello di sinistra in Figura 5.2).

Gli input che scatenano l'esecuzione della funzione riguardano eventi che causano la modifica del grafico. Tra le tracce del grafico in visualizzazione si individuano quelle che rappresentano i link (tipo di grafico "CHORDS") e se ne estraggono informazioni.

Se i link sono filtrati in base alla loro posizione all'interno delle feature, allora possiedono un valore "feat-type". In tal caso verranno mostrati: il numero dei link recuperati, il numero dei link che si trovano all'interno delle feature selezionate e tutte le coppie di feature in questione. In caso contrario, se sono effettivamente presenti dei link nel grafico che si sta visualizzando, si mostra solamente il numero dei link rappresentati. Se nessun reverse complement è stato trovato, il pennello mostrerà la scritta "*No information available*".

6.5 Valutazione delle prestazioni

Avendo analizzato tutte le tre fasi di sviluppo del progetto InstaCircos è ora possibile riassumerne le prestazioni in termini di tempo di esecuzione e occupazione di memoria (server-side). Le analisi sono state svolte per la creazione di grafici di due differenti genomi: il piccolo genoma del Pyrococcus Abyssi (file GBFF di 4.44MB) e quello dell'Homo Sapiens (file GBFF di 4.26GB, link esclusi).

Oltre a mostrare le performance delle varie implementazioni, per dare risalto alle scelte implementate, si effettuerà un confronto con il pacchetto Python pyCircos .

I testi sono stati effettuati su un notebook dalle seguenti caratteristiche: x64-based CPU @2.70GHz, 12GB RAM, 1TB hard drive.

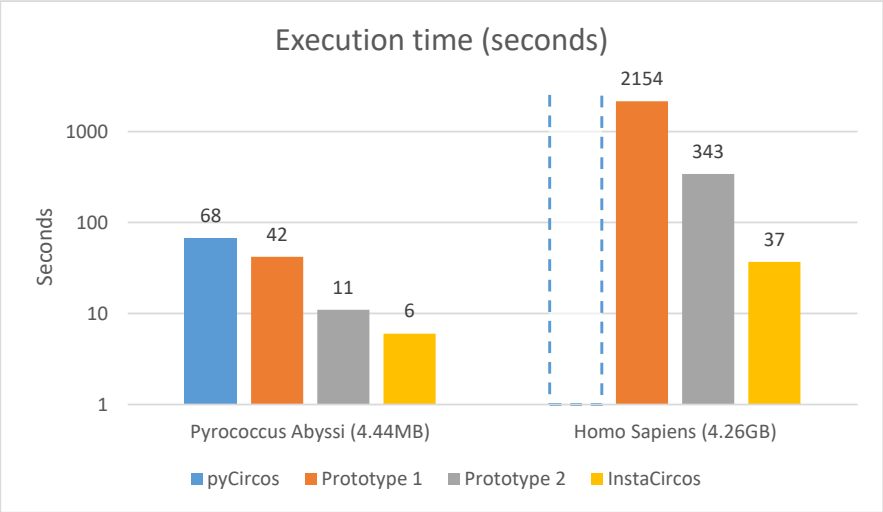

Figura 6.19: Confronto dei tempi di esecuzione per il genoma piccolo (sinistra) e per il genoma grande (destra).

Il grafico in Figura 6.19 mostra il confronto delle prestazioni. in tempo di esecuzione, per la generazione del grafico del genoma piccolo, sulla sinistra, e per quello del genoma grande (sulla destra).

Per la creazione del grafico dell'organismo Pyrococcus Abyssi, i tempi di esecuzione dei due prototipi e di InstaCircos (implementazione finale) sono rispettivamente 42, 11 e 6 secondi, rispetto ai 68 secondi impiegati da pyCircos.

Generare lo stesso tipo di grafico per il genoma umano non è possibile con pyCircos, a causa di un errore di memoria. I tempi di esecuzione delle tre implementazioni del progetto InstaCircos sono, invece, i seguenti: 2154 secondi per il Prototipo 1, 343 secondi per il Prototipo 2 e 37 secondi per l'implementazione finale.

Riguardo all'occupazione di memoria server-side, visibile in Figura 6.20, l'esecuzione di pyCircos sul genoma Pyrococcus Abyssi richiede 172MB. Il Prototipo 1 del progetto InstaCircos li riduce a 39MB, diminuiti ulteriormente a 25MB e 14MB rispettivamente per il Prototipo 2 e per l'implementazione finale, grazie all'uso del database.

La differenza di performance di InstaCircos e pyCircos sono ancora più visibili nella generazione del grafico del genoma umano: pyCircos finisce la memo-

ria disponibile generando un errore, mentre InstaCircos riduce l'occupazione da 6,43GB e 1,3GB dei primi due prototipi a soli 55MB della versione finale.

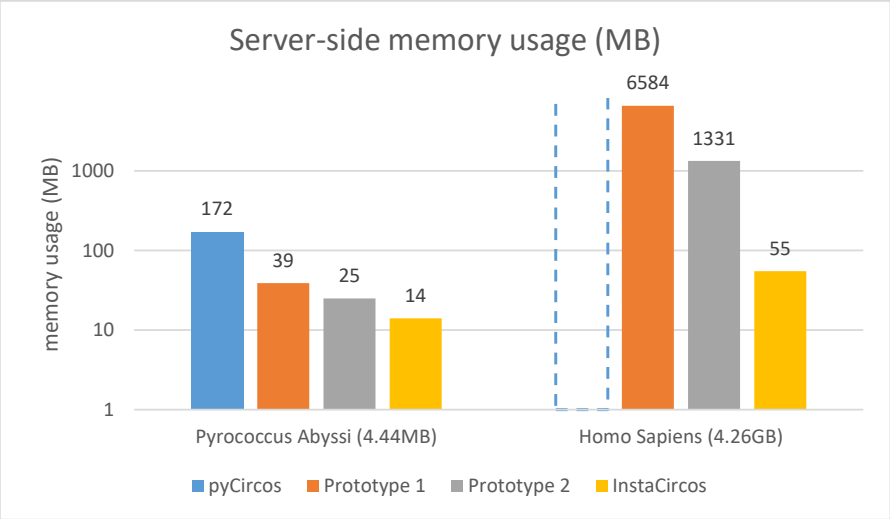

Figura 6.20: Confronto dell'occupazione di memoria server-side per il genoma piccolo (sinistra) e per il genoma grande (destra).

Conclusioni e sviluppi futuri

La collaborazione tra l'Università di Modena e Reggio Emilia e l'Università di Padova ha permesso la creazione del progetto InstaCircos: una applicazione web per la visualizzazione di dati genomici in forma circolare, che supera efficacemente le limitazioni degli strumenti attualmente disponibili.

InstaCircos è facile da usare, non necessitando alcuna installazione o elaborazione preliminare; efficiente nella generazione dei grafici di genomi di grandi dimensioni, grazie all'uso di un database; interattivo nella visualizzazione dei grafici e nell'uso della piattaforma, fornendo gli strumenti per generare nuovi grafici in modo dinamico.

I requisiti e gli obiettivi iniziali sono stati soddisfatti e raggiunti, ma il progetto lascia ampio spazio all'aggiunta di nuove funzionalità, tra cui: l'aumento dei genomi memorizzati nel database per offrire una più vasta scelta di organismi importanti; la possibilità di generare grafici a partire da file forniti dall'utente; l'aggiunta di ulteriori parametri per la personalizzazione dei grafici creati dinamicamente, tra cui la possibilità di rappresentare ulteriori dati genomici quali il GC-content e il GC-skew (indici della quantità delle basi G e C).

Il progetto ha visto la pubblicazione di un articolo, *InstaCircos: a Web Application for Fast and Interactive Circular Visualization of Large Genomic Data (Work in Progress)*, accettato nella conferenza *IV2020: VDSML – Visualization in Data Science and Machine Learning* (7-11 settembre 2020) [1].

Una volta conclusa l'implementazione del progetto, InstaCircos sarà reso pubblicamente disponibile come applicazione web.

Bibliografia

[1] Information Visualisation Society Conference. http://iv.csites.fct.unl.pt/at/.

[2] MongoDB API for Python . https://pymongo.readthedocs.io/en/stable/#overview.

[3] Python Modules for Circos Plot. http://github.com/KimBioInfoStudio/PyCircos/.

[4] Biopython 1.77: https://biopython.org/.

[5] Biopython 1.77 Tutorial and Cookbook: http://biopython.org/DIST/docs/tutorial/Tutorial.html.

[6] BSON (Binary JSON) Serialization: http://bsonspec.org/.

[7] Wei-Hien Cheong, Yung-Chie Tan, Soon-Joo Yap, and Kee Peng Ng. Clico fs: An interactive web-based service of circos. *Bioinformatics (Oxford, England)*, 31, 07 2015.

[8] Kristina Chodorow. *MongoDB: The Definitive Guide, 2nd Edition.* O'Reilly Media, Inc., 2013.

[9] Peter J. A. Cock, Tiago Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and Michiel J. L. de Hoon. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, 03 2009.

[10] Nikos Darzentas. Circoletto: visualizing sequence similarity with Circos. *Bioinformatics*, 26(20):2620–2621, 08 2010.

[11] Dash Documentation & User Guide — Plotly: https://dash.plotly.com/.

[12] Dash Bio 0.4.8 — Dash for Python documentation: https://dash.plotly.com/.

[13] DB-Engines - Knowledge Base of Relational and NoSQL Database Management Systems: https://db-engines.com/en/.

[14] Robert Harris. *Improved Pairwise Alignment of Genomic DNA*. PhD thesis, 01 2007.

[15] Ying Hu, Chunhua Yan, Chih-Hao Hsu, Qing-Rong Chen, Kelvin Niu, George A Komatsoulis, and Daoud Meerzaman. Omiccircos: A simple-to-use r package for the circular visualization of multidimensional omics data. *Cancer Inform*, 13:13–20, 01 2016.

[16] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[17] Plotly Technologies Inc. Collaborative data science, 2015.

[18] Indexes - MongoDB Manual: https://docs.mongodb.com/manual/indexes/.

[19] Martin Krzywinski, Jacqueline Schein, Inanç Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven Jones, and Marco Marra. Circos: an information aesthetic for comparative genomics. *Genome research*, 19:1639–45, 07 2009.

[20] Matplotlib: Python plotting - Matplotlib 3.1.1 documentation: https://matplotlib.org/3.1.1/index.html.

[21] Cydney Nielsen, Mike Cantor, Inna Dubchak, David Gordon, and Ting Wang. Visualizing genomes: Techniques and challenges. *Nature methods*, 7:S5–S15, 03 2010.

[22] What is NumPy? - NumPy v1.19 Manual: https://numpy.org/doc/stable/user/whatisnumpy.html.

[23] Pandas overview - pandas 1.1.0 documentation: https://pandas.pydata.org/docs/getting_started/overview.html.

[24] RefSeq: NCBI Reference Sequence database: https://www.ncbi.nlm.nih.gov/refseq/about/.

[25] RefSeq Genomes Index: ftp://ftp.ncbi.nlm.nih.gov/genomes/refseq.

[26] Wikipedia contributors. Human genome to genes, 2005. [Online; accessed 09-August-2020].

[27] Hongen Zhang, Paul Meltzer, and Sean Davis. Rcircos: An r package for circos 2d track plots. *BMC bioinformatics*, 14:244, 08 2013.