UNIVERSITÀ
DI PARMA

Dipartimento di Scienze Matematiche, Fisiche e Informatiche

Corso di Laurea Triennale in Informatica

Un approccio scalabile per il calcolo della omologia di sequenza tra geni

Candidato:
Giandonato Inverso

Relatore:
Ch.mo Prof. Vincenzo Bonnici

Anno Accademico 2021–2022

Indice

Introduzione

Il concetto di bioinformatica si può descrivere come la convergenza di due rivoluzioni tecnologiche: la crescita esplosiva delle biotecnologie, uguagliata da quella delle tecnologie informatiche [Boguski, 1998].

Di particolare interesse è l'insieme delle informazioni che una cellula passa alle sue progenie. Tale insieme viene chiamato genoma e spesso, ma non sempre, coincide con l'informazione contenuta nelle molecole di DNA (acido desossiribonucleico).

Negli ultimi decenni, sulla base di queste due rivoluzioni tecnologiche, è stato introdotto un nuovo paradigma di rappresentazione e analisi del genoma: l'*approccio orientato al gene*.

Tale approccio consiste nel confrontare tra loro sequenze di geni quantificandone la reciproca "somiglianza". Sulla base delle relazioni di somiglianza è possibile inferire una "omologia".

Le relazioni di omologia costituiscono il *contenuto pangenomico*, la cui scoperta, appunto, consiste proprio nella ricerca dell'omologia genetica tra sequenze di geni, al fine di raggrupparli in famiglie.

Le analisi del pan-genoma hanno trovato molte applicazioni negli studi clinici. Gli studi pangenomici sono utili, ad esempio, per identificare i bersagli farmacologici dei vaccini e degli antibatterici [Serruto et al., 2009] e per studiare gli agenti patogeni nelle malattie epidemiche [Holt et al., 2008].

L'obiettivo di questa tesi è, pertanto, ricercare e implementare una nuova metodologia scalabile per calcolare l'omologia di sequenza tra geni.

Questo elaborato deriva dalle attività di tirocinio svolte presso l'Università di Parma insieme al tutor prof. Bonnici Vincenzo

Nel capitolo 1 sono presenti le definizioni relative alla terminologia scientifica utilizzata nella tesi, nonchè concetti essenziali di biologia molecolare ed evolutiva. Nel capitolo 2 viene esposto il background scientifico relativo

alle analisi pangenomiche moderne. Nel capitolo 3 viene descritto lo stato dell'arte, in particolare viene descritta la metodologia per la rilevazione dell'omologia genica su cui è basata questa tesi, analizzando i punti di forza nonchè alcune criticità. Nel capitolo 4 viene descritta una nuova metodologia per calcolare l'omologia di sequenza tra geni. Infine, nel capitolo 5 vengono discussi i test sperimentali effettuati e i risultati ottenuti.

Capitolo 1

Definizioni

Definizione 1.0.1. L' *atomo* è l'unità più piccola e indivisibile della materia. È composto da un nucleo, al cui interno si trovano protoni e neutroni, e da una nube di elettroni [Silvestroni and Pasquali, 1996].

Definizione 1.0.2. Un *legame chimico* è una una forza di natura elettrostatica che tiene uniti più atomi in una specie chimica (legami forti, o primari o intramolecolari) o più molecole in una sostanza allo stato condensato [Silvestroni and Pasquali, 1996].

Definizione 1.0.3. Un *legame covalente* è un legame chimico in cui due atomi mettono in comune delle coppie di elettroni [Silvestroni and Pasquali, 1996].

Definizione 1.0.4. Una *molecola* è un'entità elettricamente neutra composta da due o più atomi uniti da un legame covalente [Silvestroni and Pasquali, 1996].

Definizione 1.0.5. La *massa molecolare* o *peso molecolare* è il rapporto tra la massa di una data quantità di quella sostanza e il numero di moli della stessa quantità di quella sostanza [Silvestroni and Pasquali, 1996].

Definizione 1.0.6. Una *macromolecola* è una molecola di dimensioni molto grandi e di peso molecolare molto elevato [McNaught et al., 1997].

Definizione 1.0.7. Un *monomero* è molecola in grado di combinarsi con due, tre o molte molecole identiche per formare composti a più elevato peso molecolare [Silvestroni and Pasquali, 1996].

Definizione 1.0.8. Un *polimero* è una macromolecola costituita da una catena di monomeri mediante la ripetizione dello stesso tipo di legame covalente.

Definizione 1.0.9. Un *nucleotide* è un simbolo appartenente all'alfabeto {A, C, G, T}.

Definizione 1.0.10. Un *amminoacido* è un simbolo appartenente all'alfabeto {F, L, I, M, V, S, P, T, A, Y, *, H, Q, N, K, D, E, C, W, R, G}.

Ogni amminoacido è ottenuto attraverso una traduzione di una sequenza di 3 nucleotidi (chiamata tripletta):

Amminoacido	Tripletta nucleotidi
F	TTT
L	TTA
I	ATT
M	ATG
V	GTT
S	TCT
P	CCT
T	ACT
A	GCT
Y	TAT
*	TAA
H	CAT
Q	CAA
N	AAT
K	AAA
D	GAT
E	GAA
C	TGT
W	TGG
R	CGT
G	GGG

Tabella 1.1: Esempio di tabella relativa alla traduzione di nucleotidi in amminoacidi per il dominio dei procarioti

Definizione 1.0.11. Un *gene* o *sequenza biologica* è una stringa s sull'alfabeto di amminoacidi Γ, $s = a_1 a_2 \dots a_h$, con $a_i \in \Gamma$ tale che $1 \leq i \leq h$.

Definizione 1.0.12. L'insieme dei k-meri di una stringa s è composto da tutte le sottostringhe possibili di s di lunghezza k. Una sequenza s, che ha lunghezza $|s|$, contiene $|s| - k + 1$ k-meri diversi. Un k-mero w si può

presentare diverse volte in s. Il numero di volte in cui w si presenta in s si chiama *molteplicità* di w in s e si denota con $c_s(w)$.

Definizione 1.0.13. Il k-dizionario della stringa s è un insieme composto da tutti i k-meri distinti ottenuti a partire da s:

$$D_k(s) = s[i..i+k] : 1 \leq i \leq |s| - k,$$

dove $s[i..i+k]$ è la sottostringa di s che inizia alla posizione i e termina dopo k caratteri.

Definizione 1.0.14. L'*indice di Jaccard*, oppure *indice di similarità di Jaccard*, è un indice statistico che permette di misurare quanto sono simili due insieme campionari ed è dato dal rapporto tra la dimensione dell'intersezione e la dimensione dell'unione degli indici campionari.

Definizione 1.0.15. L'*indice di Jaccard generalizzato*, oppure *indice di similarità di Jaccard generalizzata*, è una generalizzazione dell'*indice di Jaccard*. Essa permette di calcolare l'*indice di Jaccard* tra due variabili piuttosto che tra due insiemi.

Definizione 1.0.16. Data una popolazione di n *individui*, un *genoma* è l'insieme di geni dell'i-esimo individuo e si denota con:

$$G^i = \{s_1, s_2, \ldots, s_m\}$$

Definizione 1.0.17. La *lunghezza genetica* di G^i è data dalla somma delle lunghezze di tutti i geni del genoma e si denota con $\langle G^i \rangle$.

Definizione 1.0.18. Un *grafo G* è una coppia *(V,E)* dove dove*V(G)* è l'insieme dei *vertici* (o nodi) ed *E(G)* è l'insieme degli *archi* (che può anche essere vuoto).

Definizione 1.0.19. In un *grafo G* una *relazione di adiacenza* è un collegamento fra due nodi.

Definizione 1.0.20. Un *grafo G* si dice *indiretto* quando la relazione di adiacenza è simmetrica. L'arco che connette il nodo u al nodo v è lo stesso che connette il v al nodo u. Pertanto, l'ordine dei vertici nella coppia che compone l'arco non ha importanza.

Definizione 1.0.21. Un *grafo G* si dice *diretto* (o orientato) se gli elementi di *E(G)* sono coppie *ordinate* e, di conseguenza, gli archi sono associati ad una direzione.

Definizione 1.0.22. Un *grafo pesato* G è un grafo dove ogni elemento dell'insieme *E(G)* ha associato un valore numerico.

Definizione 1.0.23. In un *grafo* G, si definisce *percorso di lunghezza k in G* una sequenza di vertici collegata da una sequenza di archi.

Definizione 1.0.24. In un *grafo* G, un *cammino* che connette il nodo u al nodo v è un *percorso* con i lati a due a due distinti tra loro.

Definizione 1.0.25. In un *grafo* G, si definisce *cammino minimo* che connette il nodo u al nodo v, il *cammino* di lunghezza minima che connette il nodo u al nodo v. Nei grafi non pesati si conta il numero di archi attraversati durante il percorso, mentre nei grafi pesati si sommano i pesi degli archi attraversati.

Definizione 1.0.26. Un *sottografo* di un grafo G è un grafo *G'(V',E')* composto da un sottinsieme dei nodi e degli archi di G.

Definizione 1.0.27. Una *componente connessa* di un grafo G è un *sottografo di G* avente tutti nodi connessi tra loro e che non può essere esteso, in quanto non esistono ulteriori nodi in G che siano connessi ai nodi di G'.

Definizione 1.0.28. Nella *teoria dei grafi*, gli *indicatori di centralità* assegnano un valore numerico ai nodi all'interno di un grafo corrispondente alla loro posizione nella rete.

Definizione 1.0.29. Nella *teoria dei grafi*, la *betweenness centrality* è una misura della centralità in un grafo basato su *cammini minimi*.

Definizione 1.0.30. Nella *teoria dei grafi*, la *betweenness centrality di un nodo* u è il numero di cammini minimi che attraversano il nodo u.

Definizione 1.0.31. Nella *teoria dei grafi*, la *betweenness centrality di un arco* y è il numero di cammini minimi tra coppie di vertici che passano da tale arco.

Definizione 1.0.32. Un *processo* racchiude una sequenza di attività ed è mappato in memoria RAM e controllato dal processore attraverso il sistema operativo.

Definizione 1.0.33. Un *thread* è l'unità di elaborazione più piccola all'interno di un sistema operativo. Esso può esistere solo all'interno di un *processo* infatti, quando un processo termina, terminano anche i thread.

Definizione 1.0.34. In informatica, con il termine *High-performance computing* o *HPC* si identifica un campo multidisciplinare che include l'amministrazione dei sistemi (comprese le conoscenze di rete e sicurezza), l'elettronica digitale, l'architettura dei computer, i linguaggi di programmazione e la programmazione parallela [Brazell and Bettersworth, 2008].

Definizione 1.0.35. Nell'ambito dell'*High performace computing*, le *architetture parallele* sono sistemi in grado di eseguire più istruzioni diverse su più dati in parallelo. Tali architetture vengono classificate attraverso la *tassonomia di Flynn*.

Definizione 1.0.36. La *tassonomia di Flynn* suddivide le architetture dei calcolatori in quattro categorie, a seconda di due dimensioni indipendenti quali il flusso di istruzioni e il flusso di dati, che possono essere singole o multiple.

Definizione 1.0.37. *MIMD* (*Multiple Instruction stream Multiple Data stream*) è una delle quattro categorie delle architetture degli elaboratori appartenente alla *tassonomia di Flynn*. È un tipo di *architettura parallela* in cui ogni unità di calcolo può eseguire un differente flusso di istruzioni e, ogni flusso di istruzioni lavora su un differente flusso di dati.

Definizione 1.0.38. Nell'ambito delle *architetture parallele* è possibile distinguere *architetture a memoria distribuita*, nelle quali le unità di elaborazione accedono esclusivamente ad una memoria locale che non fa parte dello spazio di indirizzamento delle altre unità di calcolo, e *architetture a memoria condivisa*, nelle quali le unità di elaborazione accedono alla memoria come spazio di indirizzamento globale condiviso tra tutte le unità di elaborazione.

Definizione 1.0.39. La *programmazione parallela* è la tecnica di programmazione necessaria per la *scomposizione del carico computazionale* in task da distribuire ed eseguire sui diversi livelli di parallelismo dei sistemi HPC.

Definizione 1.0.40. Nell'ambito della *programmazione parallela*, la *decomposizione a livello di dominio* è una tipologia di *scomposizione del carico computazionale* che consiste nella suddivisione di dati, organizzati in strutture regolari, in strutture più piccole di dimensione uguale e assegnati a diverse unità computazionali.

Capitolo 2

Background

Un organismo vivente è un'entità soggetta alle leggi naturali, le stesse che controllano il resto del mondo fisico, ma tutti gli organismi viventi, comprese le loro parti, vengono controllati anche da una seconda fonte di causalità: i programmi genetici. L'assenza o la presenza di programmi genetici indica il confine netto tra l'inanimato e il mondo vivente [Ereshefsky, 1990]

È questa la definizione di organismo vivente che dà il biologo Ernst Walter Mayr nella sua pubblicazione scientifica, *Toward a new philosophy of biology : observations of an evolutionist*. Secondo Mayr, appunto, la differenza tra il mondo inanimato e gli esseri viventi è data proprio dall'esistenza dei *programmi genetici*. Infatti, tutti gli organismi viventi possiedono al loro interno le informazioni che definiscono la loro struttura e funzione.

Nella maggior parte di essi, l'informazione è memorizzata nel DNA, il quale contiene tutte le informazioni genetiche necessarie per la produzione di molecole essenziali per lo sviluppo degli esseri viventi.

In chimica, il DNA è un polimero a doppia catena i cui monomeri sono chiamati nucleotidi. Esistono 4 diversi nucleotidi, rappresentati con 4 lettere: A, C, G, T che derivano dalle iniziali delle basi azotate che li costituiscono (adenina, citosina, guanina, timina).

Il DNA risiede nel nucleo cellulare, il quale si occupa di conservare le informazioni genetiche. Esso è un filamento che può essere molto lungo, infatti, per essere contenuto nel nucleo, deve essere *spiralizzato* in diversi livelli e il grado di spiralizzazione massimo si chiama *cromosoma*.

Il DNA fu isolato per la prima volta nel 1869 dal biochimico Friedrich Miescher e, all'incirca negli stessi anni, il biologo Gregor Johann Mendel elaborò la *teoria sull'eredità biologica*, basata sui suoi famosi esperimenti tra piante di pisello odoroso. Mendel condusse gli esperimenti su questo tipo di pianta perché si riproduce per autoimpollinazione e scoprì i *caratteri*

dominanti, che rimanevano inalterati nelle nuove generazioni, e i *caratteri recessivi* che si ripresentavano solo in alcune generazioni.

Gregor Johann Mendel è considerato, quindi, il padre della *genetica*, una branca della biologia molecolare che studia i *geni*, cioè determinati caratteri di un individuo che vengono trasmessi alle generazioni successive. Un *genoma* è, quindi, un insieme di geni di un particolare individuo.

La *genetica* si occupa di studiare anche l'*ereditarietà genetica* degli esseri viventi, cioè la trasmissione, da una generazione alle successive, dei caratteri contenuti nel genoma nonchè la *variabilità genetica*, cioè versioni diverse di uno stesso organismo.

2.1 Variabilità genetica

Data una specie di un organismo vivente, determinati individui appartenenti a tale specie non avranno tutti lo stesso genoma, perché ogni individuo è frutto di una combinazione dei geni materni e paterni.

Secondo il concetto di *selezione naturale*, introdotto dal celebre biologo *Charles Darwin*, in base alla *diversità genetica* di individui di una data specie, si ha un progressivo aumento di coloro che hanno caratteristiche favorevoli ad una particolare capacità di adattamento e sopravvivenza per l'ambiente in cui vivono.

La diversità genetica si verifica, ad esempio, a causa delle *mutazioni genetiche*, cioè a modifiche nel DNA di un individuo dovute ad agenti esterni oppure a fenomeni naturali.

Le mutazioni genetiche all'interno di un gene più frequenti sono di tipo:

- *Delezione:* perdita di uno o più nucleotidi;
- *Inserzione:* aggiunta di uno o più nucleotidi;
- *Sostituzione:* scambio di nucleotidi;
- *Duplicazione:* moltiplicazione di nucleotidi.

All'interno delle cellule l'informazione genetica viene utilizzata per la produzione di *proteine* attraverso un processo biochimico chiamato *sintesi proteica*. In particolare, la catena di nucleotidi viene tradotta a blocchi di triplette in determinati *amminoacidi*, i quali poi formeranno le proteine. Le mutazioni, quindi, sono la causa di traduzioni differenti che, producendo una proteina diversa, possono renderla non funzionante o dannosa.

2.2 Genomica

La genomica è una branca della genetica ed è la disciplina che si occupa della struttura, sequenza, funzione ed evoluzione del genoma, vale a dire di tutta l'informazione genetica contenuta nel DNA presente nelle cellule di una particolare specie [Cavallaro,].

Tale disciplina nasce nei primi anni '80 a seguito dei primi studi sul sequenziamento di genomi, cioè la determinazione dell'ordine dei nucleotidi di una molecola di DNA, e a seguito del primo sequenziamento relativo al *virus fago*.

Nel 1995 ci furono i primi sequenziamenti di genomi relativi ad organismi veri e propri, cioè dei batteri *Haemophilus influenzae* e *Mycoplasma genitalium*. Tuttavia, già nel 1986 lo United States Department of Energy, congiuntamente al National Institutes of Health, istituì un grande progetto in questo campo chiamato *Human Genome Project*, con l'obiettivo di sequenziare ed identificare i geni che compongono il genoma umano.

Il progetto venne ufficialmente lanciato il 1 Ottobre 1990 e il 26 Giugno 2000 venne presentata una prima bozza comprendente "solo" il 90% dell'intero genoma umano, in quanto la restante parte era ancora indecifrabile per le macchine del tempo, dato che era composta da sequenze ripetute molto lunghe che non consentivano alle stesse di individuarne l'inizio e la fine. Il progetto è stato completato il 20 giugno 2003 dal Genome Bioinformatics Group.

In base ai concetti di *variabilità genetica*, nel 2019 il National Human Genome Research Institute ha avviato il *Human Pangenome Project* per "*creare un genoma di riferimento umano più sofisticato e completo con una rappresentazione grafica della diversità genomica globale*".

Infatti, sulla base degli studi riguardanti il sequenziamento dei genomi, nel corso degli anni è emerso un nuovo approccio in questo ambito, il quale permette di studiare e rappresentare la *variabilità genetica* di una specie, la *pangenomica*.

2.3 Pangenomica

Nel 2005 il professore di microbiologia e immunologia Herve Tettelin, insieme ai suoi collaboratori, confrontò sei genomi del batterio *Streptococcus agalactiae* scoprendo che, nonostante ci fossero molti geni presenti in tutti i genomi, si presentavano tantissimi geni "unici" nei singoli genomi.

Quindi è nella pubblicazione scientifica *Genome analysis of multiple pathogenic isolates of Streptococcus agalactiae: implications for the microbial "pan-genome"* [Tettelin et al., 2005] che compare, per la prima volta, il termine *pan-genoma*, coniato proprio dagli stessi autori.

Il termine *pan-genoma* è costituito da un prefisso "pan" che deriva dal greco e significa "tutto", seguito dalla parola *genoma* per indicare l'insieme dei geni acquisiti da una specie [Tettelin et al., 2005].

Un *pan-genoma* è, quindi, una struttura astratta definita su un insieme di geni che discendono tutti da un antenato comune [Tettelin et al., 2005, Bonnici et al., 2018].

Infatti, osservando i genomi di individui di una data specie, è possibile inviduare un insieme di *geni core*, formato da geni comuni a tutti gli individui, i quali sono coinvolti in funzionalità essenziali per la vita. Un insieme di *geni accessori*, formato da geni presenti nella maggior parte degli individui, i quali rappresentano caratteristiche variabili e un insieme di *geni singleton*, formato da geni che sono univoci per ogni individuo, che rappresentano alcune funzionalità specifiche del genoma.

L'unione fra i *geni core* e i *geni accessori* di tutti gli organismi, sequenziati all'interno di una determinata specie, prende il nome di pan-genoma [Mira et al., 2010].

Nello stesso anno il ricercatore di microbiologia Duccio Medini, insieme ai suoi collaboratori, scoprì che, aumentando il numero di genomi presi in considerazione in uno studio pangenomico, il numero di nuovi geni scoperti diminuisce in modo asintotico [Medini et al., 2005].

Furono introdotti quindi i concetti di *pan-genoma aperto e chiuso*.

Un *pan-genoma* si dice *aperto* se, sulla base di un numero crescente di genomi, il numero di nuovi geni scoperti aumenta in maniera non asintotica rispetto al numero totale di geni considerato.

Viceversa, un *pan-genoma* si dice *chiuso* se presenta un numero maggiore di *geni core* rispetto ai *geni accessori* e/o *geni singleton*. Un pan-genoma *aperto* presenterà molti più *geni accessori*.

L'obiettivo di uno strumento pangenomico è la *scoperta del contenuto pangenomico* e si basa sull'identificazione di gruppi di *geni omologhi*.

Due geni sono *omologhi* se condividono un gene ancestrale comune. I geni omologhi possono essere distinti in due tipologie:

- *Paraloghi*, quando l'omologia si presenta all'interno dello stesso genoma;
- *Ortologhi*, quando l'omologia si presenta tra genomi diversi.

Questa differenza dipende dai diversi meccanismi coinvolti nella trasmissione genica. In particolare la *paralogia* è legata alla duplicazione della sequenza all'interno dello stesso genoma, mentre l'*ortologia* può essere associata a:

- *Trasmissione verticale:* accade tra i genomi della stessa discendenza e coinvolge la maggior parte dei contenuti genetici;
- *Trasmissione orizzontale:* accade tra genomi di organismi di diverse discendenze che coinvolgono uno o pochi geni.

L'identificazione di gruppi di *geni omologhi* deve tener conto della *distanza filogenetica* tra i genomi presi in considerazione, in quanto i fenomeni di *trasmissione orizzontale e verticale*, appena descritti, introducono delle *alterazioni* nelle sequenze che possono determinare una diversa interpretazione del confronto tra genomi durante l'identificazione.

In generale, i *geni core*, essendo essenziali per la vita della specie, sono spesso sottoposti a una forte selezione evolutiva, quindi le loro sequenze vengono quasi sempre trasmesse senza alcuna alterazione; mentre i *geni accessori* sono più soggetti ad alterazionni e la loro somiglianza tende a diminuire secondo la loro distanza filogenetica [Tettelin et al., 2005].

Durante le analisi è necessario, quindi, introdurre delle soglie ragionevoli ed adattive per identificare i gruppi di *geni omologhi*.

Degno di nota è il distinguo tra rilevazione di *omologia di sequenza* e rilevazione di *omologia di funzione*.

Nel primo caso, gli strumenti pangenomici consentono di determinare famiglie geniche sulla base dei risultati analitici relativi alle sequenze genetiche. Nel secondo caso, gli strumenti pangenomici determinano famiglie geniche i cui componenti conservano la loro funzione ancestrale comune.

Tali geni sono originati da un gene ancestrale comune e, seppur diversi tra loro, codificano la stessa proteina oppure proteine con funzioni molto simili tra loro.

2.4 Metodi per la scoperta del contenuto pangenomico

Nell'ambito del *machine learning*, una branca dell'intelligenza artificiale, la tecnica dell'*apprendimento non supervisionato* permette di riorganizzare in

classi una serie di dati presi in input sulla base di *caratteristiche comuni* e in assenza di un *training set*, cioè informazioni circa le classi da scoprire.

Gli *algoritmi di clustering* fanno parte delle tecniche dell'*apprendimento non supervisionato* e si basano su misure relative ad una *somiglianza* tra gli elementi.

In particolare, nella scoperta del contenuto pangenomico è presente una "fase di clustering" nella quale l'omologia viene ricercata principalmente in termini di somiglianza tra sequenze genetiche, al fine di raggruppare i geni in *cluster genici*. Da notare che, in ambito biologico, non si usano i classici algoritmi di clustering bensì delle varianti sviluppate ad hoc per identificare *cluster genetici*.

In particolare, per ogni gene bisogna trovare i geni più simili all'interno degli altri genomi. Tale somiglianza può essere calcolata mediante approcci che fanno uso di *allineamento tra le sequenze* o tramite approcci *senza allineamento*.

2.4.1 Approcci con allineamento

Gli approcci con allineamento di sequenze si occupano di quantificare la somiglianza tra sequenze, allineandole in tutti i possibili allineamenti e misurando, a parità di posizione, la percentuale di nucleotidi uguali. Tra tutti i possibili allineamenti viene scelto quello che produce il valore di similarità più alto.

2.4.2 Approcci senza allineamento

Gli approcci senza allineamento di sequenze si basano sulla quantificazione della *composizione genomica di k-meri*, che consiste nel ricercare ed enumerare, per ogni sequenza, tutte le possibili sottostringhe lunghe k. Tra due sequenze, più è alto il numero di k-meri in comune e più sono simili tra di loro.

Una volta scelto un metodo di somiglianza, l'eventuale omologia può essere inferita in diversi modi. È possibile, ad esempio, interrogare database di sequenze genetiche al fine di trovare sequenze simili a quelle di input, oppure è possibile lavorare con coppie di genomi al fine di trovare, per ogni gene di un genoma, il gene più simile nell'altro genoma [Bonnici et al., 2018].

Tale ricerca può essere unidirezionale o anche bidirezionale (condizione più forte) ed avviene sempre rispettando una soglia minima di accettazione della similarità, determinata con diverse strategie a seconda dello strumento considerato.

2.4.3 Community detection

Ottenuti gruppi omogenei di dati in cluster (famiglie di geni), è possibile assimilarli ad una *community structure* [Leskovec et al., 2009] in quanto i geni rappresentano gli invidui (nodi) e i vari cluster rappresentano gli archi che collegano più individui accumunati da caratteristiche simili (omologia di sequenza).

Nell'ambito delle reti sociali è possibile applicare gli algoritmi di *community detection*, che similmente agli *algoritmi di clustering*, permettono di partizionare grandi insiemi di nodi di un grafo in gruppi omogenei.

Alcune metodologie, infatti, per perfezionare le relazioni di omologia, utilizzano algoritmi di community detection sui risultati ottenuti dalla fase di clustering.

Nella scoperta del contenuto pangenomico, gli algoritmi di community detection, consentono di analizzare e valutare come gruppi di geni vengono raggruppati, inoltre consentono di determinare se gruppi di geni tendono a rafforzarsi o a rompersi [Bonnici et al., 2018].

La figura 2.1 rappresenta un esempio di un'architettura di un workflow pangenomico.

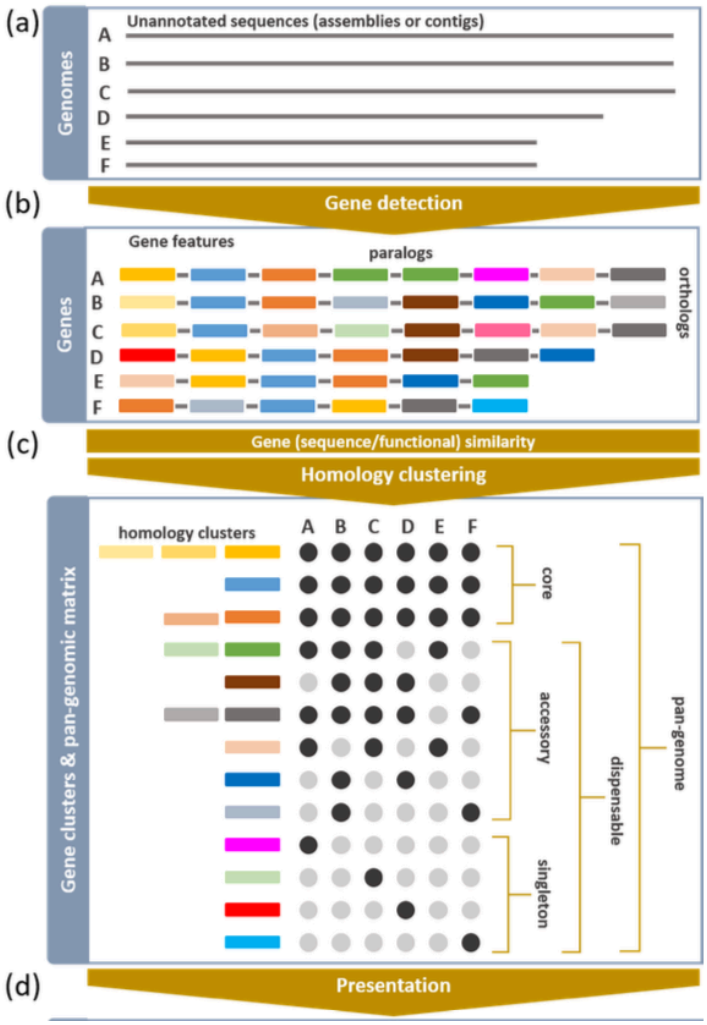

Figura 2.1: Architettura di un workflow pangenomico, *PanDelos: a dictionary-based method for pan-genome content discovery, 2018*

Capitolo 3

Stato dell'arte

3.1 PanDelos

PanDelos è una metodologia per la *scoperta del contenuto pangenomico* in organismi filogeneticamente distanti senza l'ausilio di parametri forniti dall'esterno, in quanto vengono dedotti dalle sequenze di input. Inoltre, per determinare la somiglianza tra le sequenze, utilizza un *approccio senza allineamento* che si basa sulla molteplicità di k-meri.

PanDelos, attraverso confronti tra coppie di genomi, è in grado di determinare i geni ortologhi e per ogni genoma, la similarità più alta (tra tutti i geni che lo compongono), viene utilizzata come soglia per scoprire i geni paraloghi all'interno dello stesso genoma.

Le relazioni di omologia e i geni singleton sono memorizzati in un grafo non orientato pesato, il quale viene processato da un algoritmo di community detection al fine di eliminare eventuali inconsistenze nelle famiglie di geni individuate [Bonnici et al., 2018].

3.2 Metodologia

Nell'articolo scientifico ufficiale *PanDelos: a dictionary-based method for pangenome content discovery* [Bonnici et al., 2018] sono dettagliatamente descritte tutte le fasi principali di PanDelos. Di seguito un estratto suddiviso in cinque macro-fasi:

- Costruzione dei k-dizionari;
- Selezione delle coppie di geni candidate ad essere omologhe;
- Calcolo della similarità di sequenza;

- Estrazione delle coppie di geni, calcolo della similarità bidirezionale;
- Estrazione dei paraloghi;
- Perfezionamento delle relazioni di omologia.

Nelle prime fasi vengono confrontate coppie di geni al fine di ottenere i migliori risultati in termini di somiglianza, mentre nelle ultime si ha un processamento dei cluster genici prodotti al fine di recuperare le famiglie di geni.

3.2.1 Costruzione dei k-dizionari

Nella fase iniziale PanDelos calcola il valore k al fine di comporre dizionari di k-meri per ogni gene. Se le sequenze in input sono nucleotidi, allora $k = log_4|G|$, dove 4 è la cardinalità dei nucleotidi e $|G|$ è la somma delle lunghezze di tutti i geni.

Questo valore ha dimostrato di rivelare leggi strutturali che emergono dalla massima differenza entropica tra genomi reali con genomi casuali della stessa lunghezza [Bonnici et al., 2018].

In caso di sequenze amminoacidiche, l'alfabeto da considerare è Γ, quindi il valore k sarà dato da: $k = log_{|\Gamma|} \sum_{i=1}^{n} \langle G^i \rangle$. dove $|\Gamma|$ è la cardinalità dell'alfabeto.

3.2.2 Selezione delle coppie di geni candidate ad essere omologhe

Attraverso confronti tra coppie di genomi, coppie di geni che rispettano una soglia percentuale minima di intersezione tra le rispettive coppie di dizionari, diventano coppie di geni candidati ad essere omologhi.

Formalmente, dati due geni s e t, tale che $s \in G^i$ e $t \in G^j$:

$$\widehat{D_k}(s,t) = D_k(s \cap t) = D_k(s) \cap D_k(t) > soglia$$

$$D_k(s) \cap D_k(t) = p_k(s \to t) \cup p_k(t \to s)$$

$$p_k(s \to t) = \frac{\sum_{w \in D_k(s, \cap t)} c_s(w)}{|s|-k+1}$$

$$p_k(t \to s) = \frac{\sum_{w \in D_k(t, \cap s)} c_t(w)}{|t|-k+1}$$

In particolare 2 geni sono omologhi se entrambe le percentuali $p_k(s \to t)$ e $p_k(t \to s)$ sono maggiori di una determinata soglia. Il valore di quest'ultima è quantificato, non empiricamente, in $2/k$.

Una riduzione del dominio di questo tipo consente di ottenere solo coppie di geni candidate ad essere omologhi con un punteggio di similarità non troppo basso. Per le motivazioni sulla scelta di tale soglia si faccia riferimento all'articolo scientifico ufficiale.

3.2.3 Calcolo della similarità di sequenza

Ottenute tutte le coppie di geni candidati ad essere omologhi, attraverso dei confronti tra coppie di genomi, PanDelos calcola il valore di similarità tra tali coppie che appartengono alle coppie di genomi considerate.

Tale valore è determinato attraverso il calcolo della *similarità di Jaccard generalizzata*, una generalizzazione dell'*indice di similarità di Jaccard*, il quale permette di calcolare la similarità di insiemi campionari.

Il calcolo di questo indice è basato su operazioni in ambito insiemistico e, poichè nella teoria degli insiemi non sono ammessi *duplicati* di un dato elemento, si rende necessario l'utilizzo della generalizzazione dell'*indice di similarità di Jaccard*, la quale si basa su operazioni in ambito multi-insiemistico. In particolare, l'*indice di similarità di Jaccard generalizzata* ammette nel calcolo il valore minimo e il valore massimo della molteplicità di ogni k-mero, una duplicazione dello stesso tipo di elemento.

A tal proposito, l'indice è calcolato come il rapporto tra la sommatoria della minima molteplicità per un dato k-mero in $s \in G^i$ e $t \in G^j$ e la sommatoria della massima molteplicità per un dato k-mero in $s \in G^i$ e $t \in G^j$:

$$J_k(s,t) = \frac{\sum_{w \in D_k(s \cup t)} min(c_s(w), c_t(w))}{\sum_{w \in D_k(s \cup t)} max(c_s(w), c_t(w))}$$

dove $c_s(w)$ e $c_t(w)$ sono, rispettivamente, le molteplicità del k-mero w in s e t secondo la definizione data al capitolo 1

3.2.4 Estrazione delle coppie di geni, calcolo della similarità bidirezionale

Dopo aver calcolato tutti i valori di similarità nell'insieme composto dalle coppie di geni condiderate, all'interno dello stesso contesto, PanDelos determina il sottoinsieme dei geni ortologhi attraverso l'individuazione dei geni *Best Hits* (di seguito BH) e il sottoinsieme dei geni *Bidirectional Best Hits*

(di seguito BBH).

Dato un gene $s \in G^i$, l'insieme dei geni BH di s all'interno del genoma G^i è composto da tutti i geni in G^j che hanno la stessa massima similarità di Jaccard generalizzata con s:

$$BH(s, G^j) = \{t \in G^j : J_k(s,t) = \max_{v \in \mathbb{G}^j} J_k(s,v)\}.$$

Dato un gene $s \in G^i$, l'insieme dei geni BBH di s all'interno del genoma G^i è composto da tutti i geni in G^i che hanno la stessa massima similarità di jaccard generalizzata con s e che contengono, nei loro rispettivi insiemi di geni *best hits*, il gene s:

$$BBH(s, G^j) = \{t \in G^j : t \in BH(s, G^j) \wedge s \in BH(t, G^i)\}$$

Due geni sono ortologhi se sono bidirectional best hits tra di loro.

3.2.5 Estrazione dei paraloghi

Per ogni genoma viene preso in considerazione il valore di similarità minimo tra i geni appartenenti ad esso e che sono stati individuati come BBH; quindi si utilizza come valore minimo per il calcolo della similarità tra coppie di geni all'interno dello stesso genoma (paralogia).

Le coppie di geni che rispettano questa soglia formano l'insieme dei geni paraloghi, dato che la loro somiglianza è *forte* almeno quanto la somiglianza minima tra coppie di ortologhi in cui un gene appartiene al genoma che si sta analizzando [Bonnici et al., 2018].

Le relazioni di omologia (geni ortologhi e paraloghi) sono memorizzate in un grafo non diretto pesato in cui i nodi sono riferimenti univoci dei geni, gli archi sono le relazioni di omologia che coinvolgono due geni e il peso è il valore di similarità di Jaccard generalizzata calcolato.

3.2.6 Perfezionamento delle relazioni di omologia

All'interno del grafo sono presenti diversi cluster che rappresentano potenziali famiglie di geni.

Ogni cluster è una *componente connessa* formata da numerosi archi tra i nodi, in quanto i geni sono molto simili tra di loro.

In questo caso, una componente connessa è *inconsistente* se nelle relazioni di omologia che essa costituisce, compaiono due geni appartenenti allo stesso genoma, i quali all'interno della stessa non sono presenti anche come paraloghi (quindi non sono collegati da un arco) [Bonnici et al., 2018].

Come spiegato nella fase precedente, essendo che le soglie di individuazione dei geni paraloghi sono molto basse, c'è una probabilità molto alta che tali geni non siano paraloghi e ciò significa che la componente connessa rappresenta un'unica famiglia di geni ma che in realtà, essendo inconsistente, racchiude due famiglie di geni separate.

Visivamente essa presenterà due zone con tanti archi e una zona in cui ci sono pochi archi tra i nodi e per avere esclusivamente componenti connesse consistenti, PanDelos impiega un algoritmo di *community detection* che elimina progressivamente gli archi, generando componenti connesse più piccole. In particolare, l'algoritmo utilizzato da PanDelos è l'algoritmo di *Girwan-Newman*.

L'algoritmo di *Girwan-Newman* è un algoritmo iterativo che si occupa di calcolare la *betweenness centrality* degli archi e di eliminare progressivamente quelli con il valore più alto.

PanDelos dopo aver identificato una componente connessa *inconsistente* utilizza l'algoritmo di *Girwan-Newman* per ridurla e ad ogni iterazione effettua un nuovo controllo di consistenza.

3.3 Test sperimentali e risultati

Negli articoli scientifici relativi a PanDelos:

- *PanDelos: a dictionary-based method for pan-genome content discovery, 2018*
- *Challenges in gene-oriented approaches for pangenome content discovery, 2020*

sono riportati tutti i dettagli sui test sperimentali condotti al fine di dimostrare la validità dei risultati prodotti.

PanDelos è stato testato con genomi reali e *genomi sintetici*, cioè genomi generati da un software che, partendo da un genoma *root* simula la sua evoluzione in n genomi, introducendo opzionalmente *inserzioni*, *delezioni* e *duplicazioni* nelle sequenze.

L'obiettivo dei test condotti è di misurare la correttezza di uno strumento pangenomico al variare della *distanza filogenetica*.

PanDelos è stato inizialmente testato con quattro specie di batteri, i cui genomi reali sono caratterizzati da una diversa distanza filogenetica:

- *Salmonella enterica*: 7 genomi;
- *Xanthomonas campestris*: 14 genomi;
- *Escherichia coli*: 10 genomi;
- *Mycoplasma genitalium*: 64 genomi.

Da notare che i batteri *Salmonella enterica* ed *Escherichia coli* sono noti per avere molti *geni core*, mentre il batterio *Mycoplasma genitalium* è noto per avere tanti *geni singleton* [Bonnici et al., 2018].

I risultati sulle famiglie di geni individuate rispettano le caratteristiche delle specie batteriche. In particolare le quantità di *geni core*, individuate nelle specie *Salmonella enterica* e *Escherichia coli*, è alto e dello stesso ordine di grandezza rispetto ad altri software noti. Analogamente, il numero di *geni core* per la specie *Mycoplasma genitalium* è basso.

Dal momento che, con i genomi reali non si conosce l'esatta distanza filogenetica, sono stati generati dei dataset sintetici con *ALF* [Dalquen et al., 2012], *lo strumento più completo e affidabile per la creazione di pseudo genomi batterici* [Bonnici et al., 2020], partendo dal genoma *Mycoplasma genitalium G37* come punto di origine per l'evoluzione.

Al momento della generazione dei genomi sintetici è possibile scegliere l'esatta distanza filogenetica, potendo comprendere dai risultati dello strumento pangenomico quanto siano corrette le quantità di famiglie di geni individuate.

Anche in questo tipo di test, PanDelos ha prodotto risultati eccellenti.

In generale, come descritto nel suo articolo scientifico ufficiale, i risultati e le performance di PanDelos sono state comparate con altri due software per la scoperta del contenuto pangenomico: *Roary* [Page et al., 2015] e *EDGAR* [Blom et al., 2016], ed è stato dimostrato che i risultati prodotti da PanDelos sono coerenti con gli altri software ed ottenuti in minor tempo. Inoltre, a differenza degli altri due software, avendo genomi con *distanze filogenetiche* alte, esso è in grado di rilevare un numero maggiore di *geni core* [Bonnici et al., 2018].

3.4 Strutture dati

PanDelos analizza coppie di genomi mediante analisi sulle molteplicità dei k-meri nelle sequenze appartenenti ad essi.

Per analizzare tali molteplicità è necessario, innanzitutto, determinare in quali geni compare un dato k-mero.

È un classico problema di ricerca afferente alla categoria dei problemi di *document listing*, i quali si occupano di stabilire i documenti che contengono una determinata *word*.

Per risolvere questo problema viene utilizzata una struttura dati per *l'indicizzazione delle stringhe* chiamata *Suffix-Array* (SA) [Manber and Myers, 1993]. In generale, un *Suffix-Array* consente di indicizzare tutti i suffissi di una stringa in ordine lessicografico.

In questo modo, dato un pattern (k-mero) di cui si intende cercare la presenza e/o la molteplicità all'interno di tale stringa, si può adottare un approccio più efficiente della semplice *ricerca posizione per posizione* (della stringa).

In particolare, questa struttura dati consente di effettuare l'operazione attraverso una ricerca dicotomica tra i suffissi ordinati in ordine lessicografico. Si passa, quindi, da una complessità lineare ad una complessità logaritmica.

Nel caso di PanDelos, tale struttura dati è stata utilizzata non tanto per la ricerca di un pattern, quanto per la enumerazione di tutti e soli i k-meri contenuti nella stringa indicizzata.

Per ogni k-mero ne viene data anche la molteplicità e l'insieme delle posizioni in cui esso occorre.

Tale operazione è ottenuta in modo efficiente utilizzando una struttura estesa rispetto al suffix array originale che si chiama *enhanced Suffix-Array* [Brandes, 2001].

In tale struttura dati, presi due suffissi consecutivi in ordine lessicografico, ne viene data la lunghezza del prefisso comune tra di loro più lungo.

Nel paper [Vincenzo Bonnici, 2015] tale struttura dati è stata ulteriormente estesa al fine di lavorare specificatamente su stringhe di tipo DNA. In particolare, la nuova struttura dati si occupa di enumerare solo k-meri nell'alfabeto nucleotidico A, C, G, T.

Infatti, nelle stringhe di DNA è possibile trovare delle occorrenze di caratteri N che indicano l'impossibilità di risalire allo specifico nucleotide per quella determinata posizione.

Nella figura 3.1 sono presenti degli esempi grafici delle principali strutture dati di PanDelos. In particolare, gli esempi grafici prendono in considerazione due genomi composti solo da due geni, per i quali vengono estratte tutte le sottostringhe di lunghezza $k = 1$ e $k = 2$.

La figura *a* mostra un *enhanced Suffix-Array* per una delle sequenze. Le prime due colonne contengono l'elenco dei 1-mero e 2-meri; la terza co-

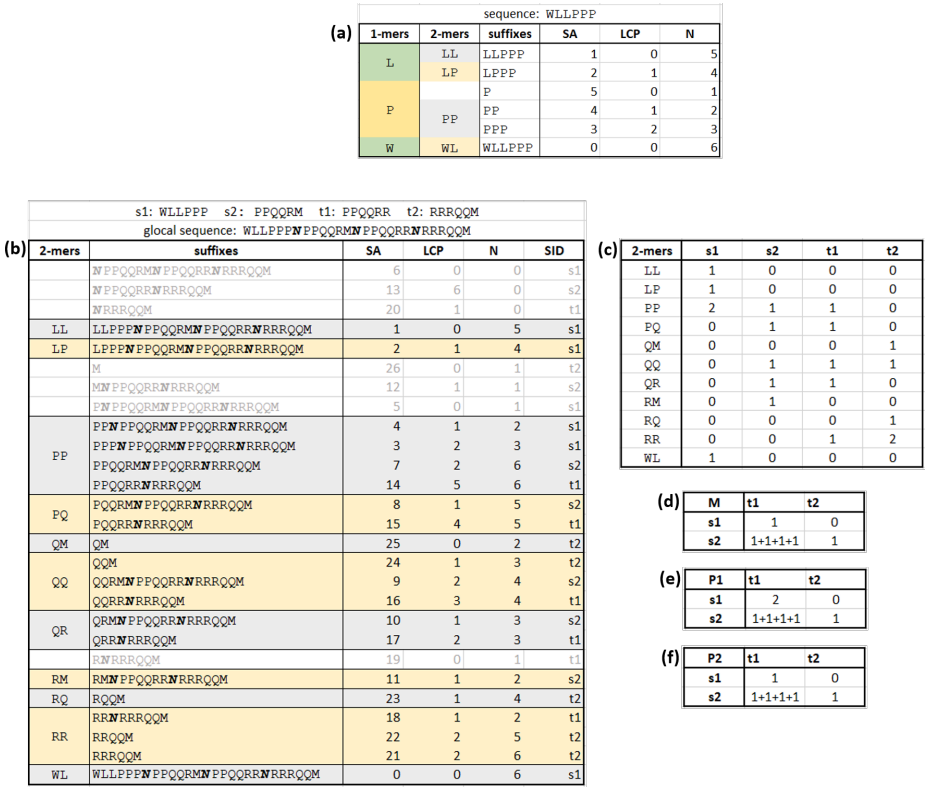

Figura 3.1: Strutture dati di PanDelos, [Bonnici et al., 2018]

lonna contiene tutti i suffissi dei 2-meri in ordine lessicografico; la quarta colonna contiene l'indice di partenza del suffisso; la quinta colonna contiene la lunghezza *LCP*, cioè la lunghezza del prefisso in comune con il suffisso precedente; l'ultima colonna contiene l'indice in cui è presente il carattere delimitatore *N*, che nel caso di una sola sequenza, corrisponde al carattere terminatore di una stringa.

La figura *b* mostra un *enhanced Suffix-Array*, il quale permette di indicizzare i 2-meri di una *global sequence*, data dalla concatenazione di tutte le sequenze dei geni della coppia di genomi, separate dal carattere delimitatore *N*.

Sono presenti, quindi, tutti i suffissi in ordine lessicografico, la lunghezza *LCP* e l'indice *n* che indica la posizione in cui è presente il carattere delimitatore *N*, il quale, in questo caso, serve ad escludere dall'operazione di enumerazione dei *k*-meri, coloro che lo coinvolgono in quanto non ammissibili perché contenenti un carattere che separa due sequenze nella concatenazione.

Inoltre, è presente il *SID*, cioè una relazione binaria tra un suffisso e il gene di appartenenza.

La matrice *c* è un esempio di enumerazione di tutti i 2-meri in tutte le sequenze.

Le matrici *d*, *e*, *f* sono matrici di ordine quadratico le cui righe e colonne corrispondono ad identificatori univoci dei geni appartenenti ai due genomi e sono utilizzate per calcolare e memorizzare i valori di similarità di Jaccard generalizzata tra coppie di geni.

3.4.1 Criticità

Le strutture dati, brevemente descritte, vengono create (e poi distrutte) ad ogni confronto tra coppie di genomi.

Dagli esperimenti effettuati durante il tirocinio è emerso che la quantità di memoria RAM richiesta è piuttosto alta in proporzione alla dimensione del dataset.

In particolare, un *enhanced Suffix-Array* necessita di 4 valori interi (4 * 4 byte) per ogni suffisso. Tuttavia, la maggior parte della quota di memoria RAM richiesta è riferita alle tre matrici *d*, *e*, *f*, in quanto ognuna necessita di uno spazio pari al prodotto della cardinalità dei genomi moltiplicato per 4 byte.

Ne consegue che, un eventuale tentativo di parallelizzazione dell'algoritmo di PanDelos genererebbe un software ad elevatissimo uso di memoria RAM, poichè le unità di calcolo dovrebbero costruire simultaneamente tali strutture dati per ogni confronto tra coppie di genomi.

Capitolo 4

Metodologia

Considerate le criticità di PanDelos analizzate nel capitolo precedente, l'obiettivo di questa tesi è ricercare e implementare una nuova metodologia scalabile per calcolare l'omologia di sequenza tra geni. Tale metodologia dovrà sfruttuare al meglio le risorse hardware disponibili al fine di produrre i risultati in tempi più brevi rispetto a quelli ottenuti da PanDelos a partire dagli stessi dataset; mantenendo, possibilmente, la stessa *consistenza* dei risultati prodotti da PanDelos, già dimostrata negli articoli scientifici citati.

4.1 Specifiche del software

Il software è stato sviluppato con il linguaggio di programmazione *C++* ed è stato rilasciato con una licenza *open source* e pubblicato su *GitHub* al seguente link: https://github.com/vbonnici/PanDelos-plusplus corredato di adeguata documentazione.

Il software soddisfa i seguenti requisti:

- È in grado di leggere file *FASTA* e costruire le strutture dati necessarie.
- È in grado di calcolare il valore k corretto per l'estrazione dei k-meri.
- È in grado di costruire i dizionari dei genomi e identificare le coppie di geni candidate ad essere omologhe memorizzandole, successivamente, all'interno di un file.
- Utilizza librerie esterne per il calcolo parallelo, affinché le fasi principali della metodologia siano eseguite con il massimo grado di parallelismo possibile.

- Integra gli script e i software, già sviluppati per PanDelos, che effettuano le operazioni di community detection sul grafo contenente le coppie di geni candidate ad essere omologhe.

4.2 Tecnologie utilizzate

4.2.1 OpenMP

OpenMP (Open Multi-Processing, https://www.openmp.org/) è un'*API* che fornisce un modello di programmazione per lo sviluppo di applicazioni parallele a *memoria condivisa*.

Il tipo di parallelismo offerto da OpenMP è realizzato esclusivamente attraverso l'uso di *thread* ed è *esplicito*, cioè la parallelizzazione è controllata dal programmatore e avviene utilizzando *direttive* all'interno del programma.

In particolare, l'API di OpenMP è stata sviluppata solo per C/C++ e Fortran ed è formata da tre diverse componenti:

- Direttive per il compilatore;
- Libreria di runtime;
- Variabili d'ambiente.

All'interno del software sviluppato sono state utilizzate principalmente le direttive per il compilatore, le quali permettono di effettuare le seguenti operazioni:

- Apertura di *regioni parallele*;
- Distribuzione di funzioni tra thread;
- Distribuzione di iterazioni di un ciclo tra thread;
- Sincronizzazione tra thread;

4.2.2 Standard Template Library

La *Standard Template Library* (STL) del C++ è una raccolta di strutture dati e algoritmi generici di utilità comune, definiti tramite il meccanismo dei template.

Nella STL le strutture dati vengono indicate come *contenitori*, cioè collezioni di elementi di un dato tipo T, ciascuna caratterizzata da un opportuno

criterio di aggregazione dei dati e dalle relative operazioni di selezione dei singoli elementi.

I *contenitori* utilizzati dal software sono i seguenti.

- vector<T>
  - Realizza il tipo di dato astratto *vettore* di elementi di tipo T. L'implementazione fornita dalla STL si basa sull'utilizzo di *array dinamici*. Dunque tutti gli elementi del vettore sono memorizzati in celle di memoria contigue per cui è garantita la possibilità di accesso diretto tramite indice, con prestazioni del tutto analoghe a quelle degli array forniti come primitivi dal linguaggio. Inoltre, la memoria allocata per un oggetto vector<T> è gestita in modo tale che possa essere espansa automaticamente se necessario.

- set<T>
  - Realizza il tipo di dato astratto *insieme*, cioè una collezione di elementi di tipo T, in cui non compaiono duplicati e l'ordine degli elementi non conta.

- tuple< $T1...Tn$ >
  - Permette di costruire collezioni di elementi aventi tipo $T1...Tn$ e di cardinalità n scelta a compile-time.

- pair<T1, T2>
  - È una specializzazione del contenitore tuple< $T1...Tn$ > e consente di costruire collezioni di elementi aventi tipo $T1, T2$ e di cardinalità 2.

- map<T1, T2>
  - Realizza il tipo di dato astratto *tabella* (o relazione binaria), tramite array chiave/valore ordinati rispetto ai valori del campo chiave. La STL permette di costruire anche mappe non ordinate, caratterizzate da un costo computazione costante in fase di inserimento, rispetto ad un costo computazionale logaritmico di quelle ordinate.

4.3 Workflow

Di seguito vengono dati i dettagli metodologici ed implementativi relativi al workflow del software. Da notare che, in corso d'opera è stato necessario sviluppare due diverse metodologie per quanto riguarda la fase di "pre-filtering" al fine di sfruttare al meglio la capacità di calcolo. Tali metodologie sono riportate come: *implementazione del parallelismo nel confronto tra coppie di genomi* e *implementazione del parallelismo nel confronto tra coppie di geni.*

La figura 4.1 è un'illustrazione grafica che sintetizza le implementazioni delle fasi rilevanti della metodologia ideata.

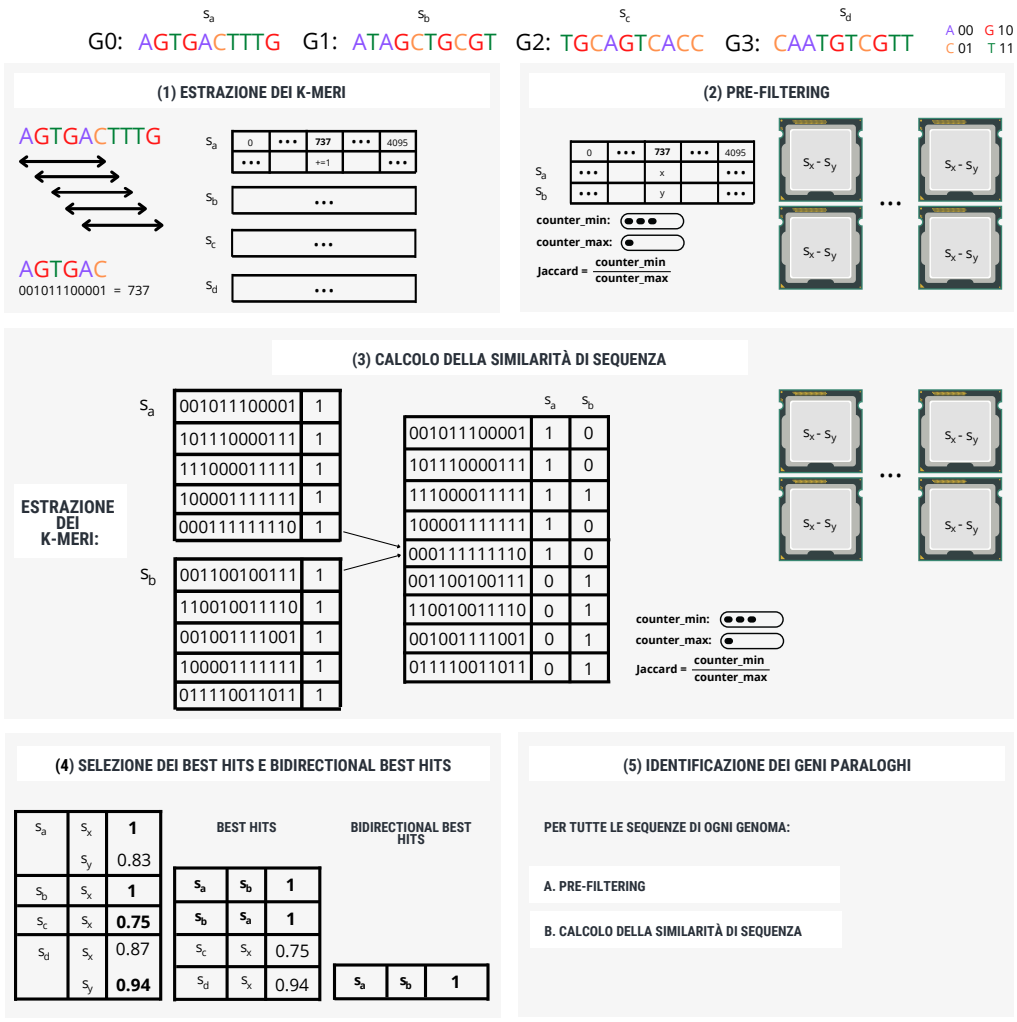

Figura 4.1: Workflow della metodologia ideata

4.3.1 Estrazione dei k-meri

Inizialmente, per ogni sequenza viene calcolata la molteplicità dei k-meri. Il software implementa questa operazione con un metodo *sliding window*, il quale si occupa di estrarre ed enumerare tutti i k-meri. Da notare che in caso di sequenze amminoacidiche, l'amminoacido di lunghezza k estratto viene convertito nella sequenza nucleotidica corrispondente.

4.3.2 Pre-filtering

Lo scopo di questa fase è confrontare coppie di genomi per determinare un primo insieme di coppie di geni candidate ad essere ortologhi, utilizzando due dimensioni prefissate non empiricamente quali, un valore k, per la selezione dei k-meri, e una soglia di similarità di Jaccard generalizzata.

In questo modo il dominio di tutte le sequenze di geni in input viene ridotto ed è possibile effettuare confronti tra coppie di genomi con soglie più alte.

Il confronto tra coppie di genomi consiste nel confrontare, iterativamente, ogni gene appartenente al primo genoma con ogni gene appartenente al secondo genoma.

Il confronto tra coppie di geni consiste nel calcolo della *similarità di Jaccard generalizzata*; cioè per ogni tipologia di k-mero si analizza quante volte esso è contenuto nei due geni. In particolare, nel confronto tra i due valori si incrementano ogni volta due contatori rispettivamente con il valore minimo e il valore massimo.

La *similarità di Jaccard generalizzata* è data proprio dal rapporto tra la somma dei valori minimi e la somma dei valori massimi; tale valore di similarità deve essere maggiore di una soglia prefissata affinchè la coppia di geni presa in considerazione sia inserita in una struttura dati elaborata successivamente, la quale conterrà le coppie di geni candidate ad essere ortologhi.

4.3.3 Calcolo della similarità di sequenza

In questa fase avviene essenzialmente lo stesso tipo di confronto, tuttavia il software opererà solo su coppie di geni che sono abbastanza simili per un valore k prefissato.

In questo confronto si utilizzerà un valore k determinato con gli stessi metodi già descritti nella sezione 3.2.1 e non ci sarà una soglia minima per la similarità di Jaccard generalizzata calcolata in quanto, come già detto, tutte le coppie sono già state filtrate.

4.3.4 Selezione dei Best Hits

Terminate le prime due fasi di identificazione dei geni ortologhi, è necessario identificare le relazioni di *Best Hits* per ogni gene.

In particolare, all'inizio di questa fase, il software opererà su una tabella le cui chiavi sono identificatori di geni e i valori sono tabelle contenenti gli identificatori di altri geni con cui il gene chiave è abbastanza simile.

Quindi ogni gene (chiave) potrà essere abbastanza simile a diversi geni (valori) appartenenti anche ad uno stesso genoma (quest'ultimo è sempre diverso dal genoma del gene chiave in quanto si stanno identificando i geni ortologhi). Lo scopo è di isolare, per ogni genoma, il gene più simile a quello preso in considerazione (chiave) e cioè quello con il valore di similarità più alto. Da notare che possono esserci best hits multipli che hanno lo stesso (massimo) valore di similarità di Jaccard generalizzata.

4.3.5 Selezione dei Bidirectional Best Hits

Dopo aver identificato i geni Best Hits, il software si occupa di identificare i geni con relazione Bidirectional Best Hits.

Per descrivere un esempio di relazione bidirectional best hits si consideri l'esempio seguente.

Data una tabella $BestHits < A, B < C, D >>$, dove:

- A è l'identificatore del gene A;
- B è la tabella dei geni con cui il gene A è in relazione Best Hits;
- C è l'identificatore del gene C;
- D è il valore di similarità di Jaccard generalizzata tra il gene A e il gene C;

Si definisce un *Bidirectional Best Hits* una coppia di geni tale per cui per una chiave A, nella sua mappa B, esiste un gene C che, visto come una delle chiavi della tabella *BestHits*, contiene nella sua mappa B una chiave uguale ad A.

Le coppie di geni individuate alla fine di questa fase sono coppie di geni ortologhi e saranno inserite nel grafo contenente, appunto, le coppie di geni candidate ad essere omologhe.

4.3.6 Identificazione dei geni paraloghi

Il processo di identificazione dei geni paraloghi si basa sugli stessi tipi di confronti appena descritti, ma applicati a coppie di geni appartenenti allo stesso genoma.

In base a quanto esposto nella sezione 3.2.5, per l'identificazione dei geni paraloghi, i geni di ogni genoma analizzato dovranno rispettare una soglia minima di similarità pari al valore più basso di similarità di Jaccard generalizzata riscontrato nella fase di *selezione dei bidirectional best hits*, relativo a tale genoma.

Questa fase si compone di due sottofasi già descritte. In particolare, le sequenze di uno stesso genoma vengono inizialmente filtrate con una fase di *pre-filtering* (cioè con soglie basse e prefissate), successivamente le coppie candidate vengono processate con lo stesso tipo di confronto ma con il valore k calcolato all'inizio e con una soglia minima rappresentata dalla similarità di Jaccard generalizzata, calcolata appositamente per ogni genoma.

Le coppie di geni individuate saranno inserite, quindi, nel grafo contenente tutte le coppie di geni candidate ad essere omologhe.

4.3.7 Perfezionamento delle relazioni di omologia

Nel software sviluppato questa fase è rimasta inalterata e si occupa, come già descritto nel capitolo precedente, di processare il grafo prodotto mediante un algoritmo di community detection, il quale individua le famiglie di geni vere e proprie.

4.4 Implementazione

Rispetto ad alcune fasi del software appena descritte, di seguito sono presenti alcuni commenti relativi a dettagli implementativi rilevanti che le riguardano.

4.4.1 Pre-filtering

Le attività di confronto tra geni sono basate, essenzialmente, sull'analisi del numero di occorrenze di un dato k-mero.

Sapendo che la cardinalità dell'alfabeto nucleotidico è pari a 4 e che in questa fase il valore k è stato fissato non empiricamente a 6, il numero di k-meri possibili è pari a $4^6 = 4096$.

Ogni sequenza di input dovrà essere riconducibile ad una struttura dati nella quale saranno memorizzate le cardinalità di tutti i suoi k-meri. Volendo indicizzare tali k-meri sotto forma di stringhe e supponendo di memorizzare

solo i k-meri effettivamente riscontrati, lo spazio richiesto (per ogni sequenza), nel caso pessimo, sarebbe pari a 4096 k-meri riscontrati * 7 byte per ogni stringa del k-mero = $\sim$ 29 Kb.

Durante lo sviluppo del software è stata implementata un'indicizzazione più efficiente mediante un array di interi di dimensione 4096, nel quale ogni elemento corrisponde ad un k-mero. In questo caso i k-meri riscontrati non vengono inseriti di volta in volta bensì l'array ha la stessa dimensione del numero di tutti i k-meri possibili, quindi lo spazio richiesto è sempre pari a 4096 k-meri riscontrati * 4 byte per ogni valore intero = $\sim$ 16 Kb.

Per realizzare questo tipo di indicizzazione si necessita di una relazione numerica tra un determinato k-mero sotto forma di stringa e un valore intero compreso tra 0 e 4095.

Per ottenere ciò è possibile convertire un determinato k-mero appartenente ai 4096 possibili, in una stringa di bit univoca.

In particolare, conoscendo la cardinalità dell'alfabeto nucleotidico è possibile comporre sequenze di bit associate a stringhe formate a partire da tale alfabeto, utilizzando 2 bit per ogni nucleotide:

- A: 00
- C: 01
- G: 10
- T: 11

In questo modo, ogni nucleotide del k-mero viene concatenato sotto forma di bit attraverso un tipo di *operazione bit a bit* che permette di spostare a sinistra i bit già memorizzati e di confrontare con un'operazione XOR i nuovi bit da aggiungere, generando una nuova sequenza di bit nella quale i nuovi aggiunti si trovano alla fine.

Tale sequenza di bit corrisponde ad un numero intero che rappresenta proprio un indice univoco della struttura dati (tra 0 e 4095) nella quale vengono enumerati i k-meri.

Per il confronto tra coppie di genomi sono state ideate due possibili implementazioni, distinte da una diversa *decomposizione del dominio.*

Una prima implementazione prevede di confrontare coppie di genomi in due *for-loop* annidati, nel quale le iterazioni del ciclo interno vengono eseguite in parallelo.

Una seconda implementazione, invece, prevede di costruire un *vector* contenente identificatori di coppie di geni appartenenti ai due genomi da confrontare. Per il confronto tra questi ultimi, quindi, le attività di analisi del

numero di occorrenze di un dato k-mero, avvengono in iterazioni parallele per ogni coppia di sequenze.

In base ai test sperimentali condotti (e descritti successivamente), all'interno del software è presente solo la seconda implementazione, in quanto la prima non sfrutta al meglio un'eventuale alta disponibilità di unità computazionali perché sensibile al numero di genomi da analizzare e/o alla dimensione di ogni genoma.

In particolare, dato un dataset composto da n genomi, se n^2 (corrispondente al numero di confronti tra coppie genomi) è minore del numero di unità computazionali, allora tra queste ultime ce ne saranno alcune del tutto inutilizzate, dato che l'esecuzione parallela avviene proprio durante i confronti tra coppie di genomi.

Inoltre, i test sperimentali condotti hanno dimostrato che avendo un numero di genomi molto più basso rispetto al numero di unità computazionali, ognuno caratterizzato da una dimensione di geni notevole, i tempi di esecuzione sono fino al 50% più alti rispetto ai tempi di esecuzione di questa stessa fase che adotta la seconda implementazione ideata.

Per entrambe le implementazioni, la libreria OpenMP si occupa di aprire la regione parallela e di distribuire le iterazioni su un numero di thread pari al numero di *core* disponibili.

Dal momento che gli identificatori delle coppie di geni candidate ad essere omologhi vengono memorizzate in un *vector*, bisogna tener conto del fatto che l'operazione di inserimento del contenitore *vector* non è *thread-safe* in quanto è una struttura dati che, in caso di esaurimento dello spazio disponibile, provvede alla riallocazione della memoria e alla copia degli elementi, però quest'ultima operazione è indipendente dalla computazione concorrente dei thread e potrebbe produrre una perdita di dati.

Per ovviare a questo problema, ogni thread inserisce le coppie di geni candidate ad essere omologhi in un *vector* locale e, al termine di ogni confronto tra coppie di genomi, tutti i thread, all'interno di una sezione critica, accodano i dati raccolti all'interno del *vector* globale.

4.4.2 Calcolo della similarità di sequenza

In questa fase, nel confronto tra coppie di genomi si prenderà in considerazione il valore k calcolato. Tale valore può essere più grande o più piccolo del valore k prefissato nella prima fase, ed è giusto ricordare che un dataset di tipo nucleotidico genererà un valore k più grande rispetto ad un dataset amminoacidico, in quanto la cardinalità dell'alfabeto nucleotidico è nettamente inferiore rispetto ad una cardinalità di un alfabeto amminoacidico.

Quindi, in base alla formula descritta nella sezione 3.2.1, il logaritmo al denominatore sarà più piccolo ed è per questo motivo che il valore k sarà più grande.

Avendo un valore k fissato a 6, i k-meri possibili sono sempre 4096 e lo spazio richiesto per ogni sequenza è modesto: $\sim$ 16 Kb

Tuttavia, supponendo di avere un valore k doppio, i k-meri possibili sarebbero 4^{12} e, lo spazio richiesto per enumerare i k-meri di ogni sequenza in un array di interi sarebbe pari a 4^{12} * 4 byte = $\sim$ 67 Mb.

Pertanto, per questa fase, è stato necessario trovare una soluzione in grado di enumerare moltissimi k-meri diversi in una quantità di spazio ragionevole, mantenendo possibilmente, lo stesso metodo di indicizzazione già implementato.

La *Utility library* del C++ offre una classe templatica per memorizzare una sequenza di N bit: std::bitset$< N >$.

Tale classe permette di effettuare le *operazioni bit a bit* più comuni quali AND, OR, NOT, XOR e binary shifting, quindi si presta perfettamente a questo caso d'uso.

A livello implementativo, questa struttura dati è una specializzazione del contenitore *vector* parametrizzato sul tipo *bool* e da 0 a 64 bit memorizzati, sono necessari 8 byte.

Il confronto tra coppie di genomi avviene esclusivamente tra le coppie di geni identificate nella fase precedente e, nello specifico, il confronto tra le tipologie di k-meri consiste nella creazione di una tabella nella quale convergono le enumerazioni dei k-meri di entrambi i geni. Tale struttura dati conterrà occorrenze univoche per un determinato k-mero e le relative quantità presenti nella coppia di geni.

In questo modo è possibile realizzare lo stesso tipo di confronto e conteggio della fase di pre-filtering, esaminando, però, solo le occorrenze dei k-meri effettivamente riscontrati e non quelle relative ai 4^k k-meri possibili.

Anche in questa fase la parallelizzazione è affidata alla libreria OpenMP, che si occupa della suddivisione del lavoro e della sincronizzazione tra i thread. Eventuali perdite di dati vengono evitate, similmente alla fase di pre-filtering, con barriere e sezioni critiche offerte dalla libreria.

4.4.3 Perfezionamento delle relazioni di omologia

Considerato l'obiettivo di sviluppare un software che sfrutti le risorse di calcolo con il massimo grado di parallelismo possibile, l'implementazione di questa

fase è stata leggermente modificata in modo tale da permettere al software di utilizzare in maniera più efficiente le risorse di calcolo durante le operazioni di community detection.

In questa fase, viene utilizzata un'implementazione dell'*algoritmo di Girwan-Newman*, fornita da una libreria Python, per lo studio di grafici e reti che si chiama *NetworkX*. L'algoritmo fornito dalla libreria si occupa di calcolare la *betweenness centrality* degli archi e di eliminare progressivamente coloro che presentano i valori più alti.

Per il calcolo della *betweenness centrality di un arco* in un insieme di nodi, la libreria *NetworkX* utilizza l'algoritmo di *Ulrik Brandes*, esposto nell'articolo scientifico *Ulrik Brandes (2001) A faster algorithm for betweenness centrality, The Journal of Mathematical Sociology*, [Brandes, 2001], dato dalla seguente espressione:

$$c_B(v) = \sum_{s \in S, t \in T} \frac{\sigma(s,t|e)}{\sigma(s,t)}$$

dove:

- S è l'insieme dei nodi di partenza per il calcolo dei cammini minimi;
- T è l'insieme dei nodi di arrivo per il calcolo dei cammini minimi;
- e è l'arco di cui si vuole calcolare la *betweenness centrality*;
- $\sigma(s,t)$ è il valore del cammino minimo tra un nodo di partenza $s \in S$ e un nodo di arrivo $t \in T$;
- $\sigma(s,t|e)$ è il valore del cammino minimo tra un nodo di partenza $s \in S$ e un nodo di arrivo $t \in T$, passante per l'arco e.

A livello implementativo, quindi, questa operazione è una somma dei risultati relativi al calcolo dei *cammini minimi*, i quali possono essere calcolati l'uno indipendentemente dagli altri.

NetworkX riporta all'interno della propria documentazione ufficiale una possibile implementazione parallela per il calcolo della *betweenness centrality* di un arco, la quale consiste nel dividere in *chunk* l'insieme di nodi della componente connessa, consentendo ad unità di calcolo separate di svolgere lo stesso tipo di operazione.

Tale implementazione è stata integrata nello script dello stato dell'arte e utilizzata solo quando la componente connessa da analizzare ha una dimensione superiore a 100, in quanto, a seguito dei test effettuati con una

dimensione inferiore, il tempo di overhead, dovuto al parallelismo, determina un tempo totale superiore all'implementazione sequenziale.

In particolare, l'algoritmo si occupa di dividere i nodi della componente connessa in *chunk* e poi, attraverso un modulo appartenente alla libreria standard di Python, denominato *multiprocessing*, il calcolo della *betweenness centrality* di un determinato arco, relativa ad un chunk di nodi, avviene su più thread. Quando tutti i thread terminano le operazioni di calcolo, avviene un'operazione di riduzione che consiste nel sommare tutti i contributi relativi alle *betweenness centrality* dei chunk alla *betweenness centrality* totale dell'intera componente connessa.

Capitolo 5

Risultati sperimentali

Al fine di ottenere risultati più simili possibile a quelli prodotti da PanDelos, la cui validità è già stata dimostrata in letteratura, nonchè di misurare i tempi di esecuzione e i requisiti di memoria RAM, presi determinati dataset di genomi sintetici generati, il nuovo software sviluppato e PanDelos sono stati testati sperimentalmente con questi ultimi e, successivamente, sono state confrontate le due distribuzioni di famiglie geniche ottenute.

Ciò ha permesso di evidenziare le differenze nei risultati prodotti dai due software e di analizzare le differenti prestazioni a parità di input.

5.1 PANPROVA

In tutti i test sperimentali effettuati, i genomi sintetici sono stati generati con *PANPROVA* (PANgenomic PROkaryotic eVolution) [Bonnici and Giugno, 2022], un software in grado di simulare l'evoluzione pangenomica procariotica partendo da un genoma *root*.

A differenza di *ALF*, il software utilizzato per la generazione dei dataset citato nella sezione 3.3, *PANPROVA* genera genomi sintetici nei quali si presentano anche *geni singleton*, inoltre produce un report dettagliato sulla distribuzione della famiglie geniche. In questo modo, attraverso i genomi generati, è possibile effettuare un confronto tra le famiglie di geni e i *geni singleton* individuati da uno strumento pangenomico, e la distribuzione corretta.

5.2 Test sperimentali

Con lo scopo di facilitare le esecuzioni dei test sperimentali, a seguito di nuove modifiche al software, le procedure di sviluppo del software e di integrazione

adottate hanno seguito un approccio denominato *Continuous Integration/-Continuous Deployment* (CI/CD).

Questo approccio è caratterizzato da una serie di fasi automatizzate che permettono un'integrazione continua del software sviluppato.

L'*integrazione continua* (CI) di un software consiste nel creare una *build* della nuova versione di quest'ultimo, effettuare le dovute procedure di *testing del software* e rilasciare le modifiche effettuate nel repository ufficiale.

Il *Deployment continuo* (CD) di un software consiste nel rilascio automatico della nuova versione da un *ambiente di test* privato all'*ambiente di produzione*, di norma pubblico. Gli automatismi di questo approccio permettono di minimizzare i tempi del rilascio stesso nonchè eventuali tempi di downtime dell'applicativo.

Nel contesto relativo allo sviluppo del software corrente, l'*ambiente di test* privato è stato rappresentato dall'IDE (*Integrated Development Environment*) *CLion* installato su un computer personale, mentre l'*ambiente di produzione* è stato rappresentato da server virtuali di *Amazon Web Services*.

In particolare, è stato utilizzato il servizio *Amazon Elastic Compute Cloud* (EC2) per creare *macchine virtuali* on-demand con sistema operativo *Ubuntu Server* e con capacità di calcolo ridimensionabile.

Le macchine virtuali utilizzate appartengono alla tipologia commerciale *c6i*, la quale adotta processori *Intel Xeon Ice Lake 8375C @ 3.5 GHz*.

A seconda dei test sperimentali condotti sono state utilizzate tali tipologie di istanze con diversa dimensione in termini di CPU virtuale (vCPU). Nello specifico sono state utilizzate macchine virtuali da 1,2,4,8,16,32,48,64,96 vCPU, tenendo presente che ogni vCPU corrisponde ad un core fisico del processore adottato.

5.2.1 Generazione dei dataset

Sono state condotte due tipologie di test sperimentali, entrambe basate sulla generazione di due popolazioni di genomi sintetici appartenenti a due specie batteriche diverse cioè *Mycoplasma genitalium* ed *Escherichia coli*.

Entrambe le popolazioni sono state generate partendo da un genoma *root* sequenziato da un batterio reale, a partire dal quale è stata simulata la sua evoluzione combinando processi di *trasferimento orizzontale* e *trasferimento verticale* con altri genomi reali della stessa specie, in un ordine filogenetico casuale.

Considerando che la dimensione genica della specie batterica *Escherichia coli* è circa 10 volte più grande rispetto a quella della specie batterica *Mycoplasma genitalium*, il numero di genomi generati per tali specie differisce in maniera sostanziale.

In particolare, partendo dai genomi *root mycoplasma_genitalium_G37* ed *escherichia_coli_O157H7*, sono stati generati due dataset composti, rispettivamente, da una popolazione di 16 genomi di *Escherichia coli* ed una popolazione di 64 genomi di *Mycoplasma genitalium*.

Ogni genoma generato da *PANPROVA* è suddiviso in tre file con estensioni diverse:

- Il file *.fna* contiene la sequenza completa del DNA;
- Il file *.gff* contiene le coordinate dei geni relative alla sequenza di DNA descritta nel file *.fna* con struttura *TSV* (tab-separated values);
- Il file *.gbff* contiene, anch'esso, le coordinate dei geni, ma con una diversa struttura.

Per ogni genoma, attraverso uno script presente nel software dello stato dell'arte, i tre file esso costituente vengono utilizzati per generare un unico file con estensione *gbk*. Questa estensione rappresenta un file nel formato standard *GenBank*, utilizzato per memorizzare la sequenza nucleotidica e i metadati del genoma, nonchè le relative *coordinate* di ogni gene.

Inoltre, con uno script diverso (anch'esso presente nel software dello stato dell'arte), viene generato un unico file con estensione *.faa* (FASTA) composto da tutte le sequenze di geni appartenenti ad n file *.gbk*.

In particolare, all'interno di un file *FASTA*, ogni gene è identificato da due stringhe. La prima stringa contiene informazioni di indicizzazione sia per il genoma di appartenenza che per tutto il file, la seconda contiene l'intera sequenza, nucleotidica o amminoacidica, del gene.

Per entrambe le popolazioni generate sono stati utilizzati questi due script per costruire file *FASTA* composti, a seconda della specie batterica, da 2 a 64 genomi con l'obiettivo di misurare i tempi di esecuzione del software sviluppato e i requisiti di memoria RAM, nonchè di confrontare i risultati ottenuti dal nuovo software sviluppato e PanDelos.

Inoltre, tenendo presente che sono state ideate ed implementate due versioni della fase di *pre-filtering*, i test sperimentali sono stati condotti su tali versioni ed hanno prodotto differenze sostanziali in termini di tempi di esecuzione e requisiti di di memoria RAM.

5.3 Tempi di esecuzione

Di seguito sono presentati i dati relativi ai tempi di esecuzione, in forma tabellare, testuale e grafica, ottenuti dai test sperimentali basati sulle due popolazioni batteriche già descritte.

5.3.1 Mycoplasma genitalium

Questa sezione si occupa di rappresentare dei dati relativi al dataset di *Mycoplasma genitalium*.

Le tabelle 5.1 e 5.2, rispettivamente rappresentate in forma grafica nelle figure 5.1 e 5.2, mostrano un resoconto dei tempi di esecuzione del software sviluppato e PanDelos, relativi ai test sperimentali condotti sulla specie *Mycoplasma genitalium*. La prima colonna contiene il numero di genomi che compongono il dataset in input, mentre le successive contengono i dati relativi ai tempi di esecuzione a seconda del numero di core a disposizione. L'ultima colonna contiene i tempi di esecuzione di PanDelos.

In generale, il software sviluppato, avendo a disposizione basse risorse di calcolo (da 1 a 4 vCPU), produce i risultati in tempi più lunghi rispetto a quelli prodotti da PanDelos.

Viceversa, avendo a disposizione risorse di calcolo medio-grandi, anche con un numero di genomi elevato (128), il software sviluppato produce i risultati in tempi generalmente minori rispetto a quelli prodotti da PanDelos.

Inoltre, l'utilizzo dell'implementazione del parallelismo nel confronto tra coppie di geni, nella fase di *pre-filtering*, permette di eseguire quest'ultima in una quantità di tempo tra il 5% e 12% in meno rispetto all'esecuzione con l'implementazione del parallelismo nel confronto tra coppie di genomi.

Da notare che, utilizzando l'implementazione del parallelismo nel confronto tra coppie di genomi, sono presenti limiti visibili ai tempi minimi ottenibili per un certo numero di genomi. Ad esempio il tempo necessario per produrre i risultati con 16 genomi in input non cambia sensibilmente da 32 vCPU (20") a 96 vCPU (15"). Viceversa, prendendo in considerazione lo stesso numero di genomi, con l'implementazione del parallelismo nel confronto tra coppie di geni, la quantità di tempo necessaria per produrre i risultati con 32 vCPU è del 41% più alta rispetto a quella necessaria per produrre i risultati con 96 vCPU.

Dim	1	2	4	8	16	32	48	64	96	PanDelos
2	2”	1”	1”	1”	1”	1”	1”	1”	1”	1”
4	6”	4”	3”	3”	2”	2”	2”	2”	2”	4”
6	13”	8”	8”	4”	4”	4”	4”	4”	4”	6”
8	25”	14”	12”	6”	6”	6”	6”	6”	6”	9”
10	43”	24”	21”	10”	9”	8”	8”	8”	8”	13”
12	68”	40”	33”	15”	14”	13”	10”	11”	10”	14”
14	97”	55”	47”	21”	18”	16”	13”	13”	12”	21”
16	2.1’	1.18’	60”	25”	22”	20”	16”	15”	15”	27”
32	-	10.06’	8’	3.28’	2.38’	1.48’	1.28’	1.15’	1’	1.1’
64	-	-	22’	12’	6.05’	4.2’	3.33’	3’	2.55’	5.23’
128	-	-	-	24’	20’	13.2’	10.5’	9’	8’	20’

Tabella 5.1: Tempi di esecuzione *Mycoplasma genitalium* - implementazione del parallelismo nel confronto tra coppie di genomi nella fase di pre-filtering

Dim	1	2	4	8	16	32	48	64	96	PanDelos
2	2”	1”	1”	1”	1”	0”	1”	0”	0”	1”
4	7”	3”	3”	2”	1”	1”	1”	1”	1”	4”
6	14”	7”	6”	3”	2”	1”	1”	2”	2”	6”
8	26”	13”	11’	7”	4”	3”	2”	2”	2”	9”
10	44”	23”	20”	10”	7”	5”	3”	4”	4”	13”
12	1.1’	38”	31”	17”	11”	7”	6”	6”	5”	14”
14	1.4’	54”	45”	24”	15”	9”	7”	7”	6”	21”
16	2.11’	1.11’	1’	32”	22”	12”	10”	9”	7”	27”
32	-	9.32’	7.38’	4’	2.15’	1.23’	1.04’	57”	46”	1.1’
64	-	-	21’	11.15’	6.2’	3.47’	2.55’	2.39’	2.6’	5.23’
128	-	-	-	32’	19.53’	12’	9.34’	8.29’	7.1’	20’

Tabella 5.2: Tempi di esecuzione *Mycoplasma genitalium* - implementazione del parallelismo nel confronto tra coppie di geni nella fase di pre-filtering

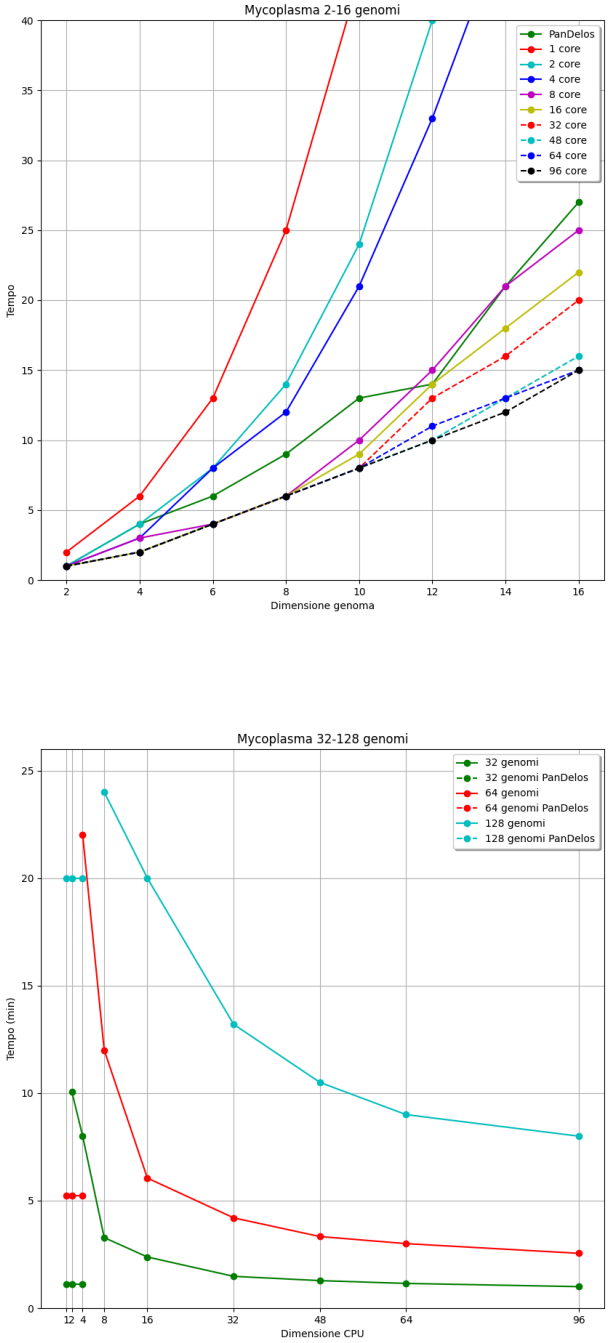

Figura 5.1: Tempi di esecuzione *Mycoplasma genitalium* 2-16 e 32-128 genomi - implementazione del parallelismo nel confronto tra coppie di genomi nella fase di pre-filtering

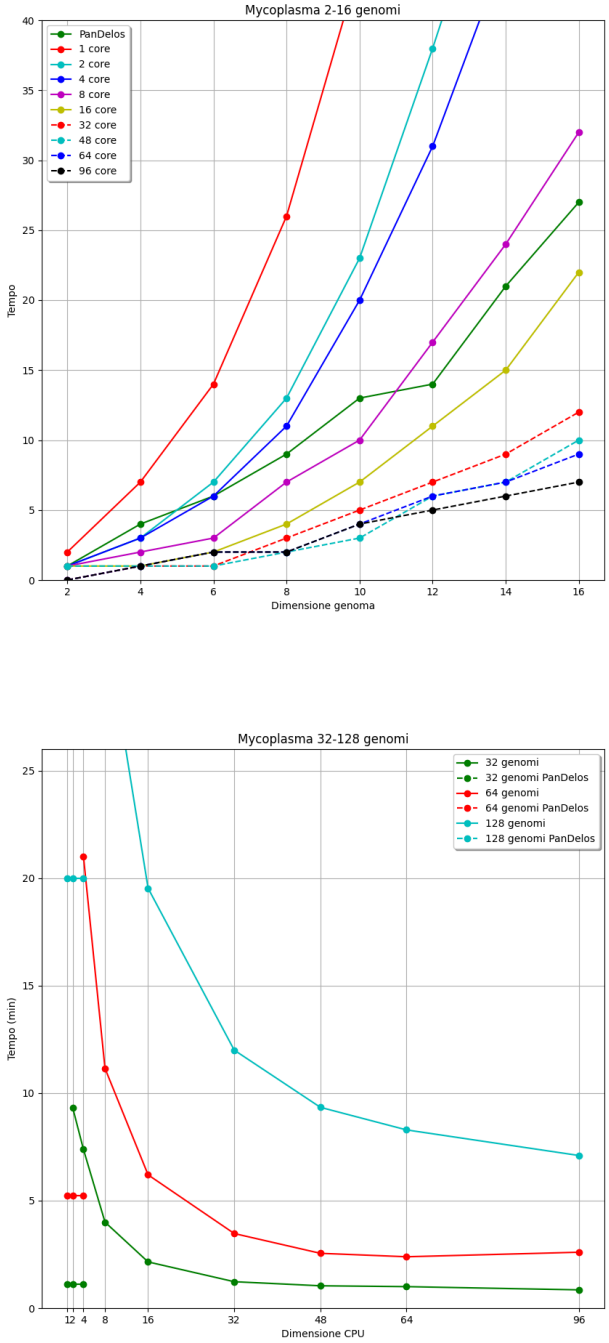

Figura 5.2: Tempi di esecuzione *Mycoplasma genitalium* 2-16 e 32-128 genomi - implementazione del parallelismo nel confronto tra coppie di geni nella fase di pre-filtering

5.3.2 Escherichia coli

Le tabelle 5.3 e 5.4, rappresentate in forma grafica nella figura 5.3, mostrano un resoconto dei tempi di esecuzione del software sviluppato e PanDelos, relativi ai test sperimentali condotti sulla specie *Escherichia coli*.

In generale, i tempi di esecuzione dei test sperimentali condotti con questa specie batterica sono più lunghi rispetto a quelli relativi ai test sperimentali condotti con la specie batterica *Mycoplasma genitalium*, data la maggiore dimensione genica.

Inoltre, utilizzando l'implementazione del parallelismo nel confronto tra coppie di genomi nella fase di pre-filtering, nonostante l'ausilio di una notevole capacità di calcolo, i tempi di esecuzione sono molto più alti rispetto a quelli di PanDelos.

Viceversa, utilizzando l'implementazione del parallelismo nel confronto tra coppie di geni nella fase di pre-filtering, nel complesso i tempi di esecuzione sono più bassi, ma comunque più alti dei tempi di esecuzione di PanDelos.

Tuttavia, utilizzando l'implementazione del parallelismo nel confronto tra coppie di geni nella fase di pre-filtering, e avendo a disposizione una notevole capacità di calcolo, i tempi di esecuzione sono più bassi rispetto a quelli di PanDelos.

Dim	1	2	4	8	16	32	48	64	96	PanDelos
2	-	-	-	5.46'	3.3'	2.3'	2.1'	2'	1.45'	20"
4	-	-	-	27'	15.35'	10'	8'	7.16'	6'	1'
6	-	-	-	-	22.23'	14.38'	12.10'	11.8'	10'	2.1'
8	-	-	-	-	-	21'	17.22'	16'	14'	4'
10	-	-	-	-	-	-	19.3'	18'	15.4'	5.1'
12	-	-	-	-	-	-	24.33'	22.18'	20.7'	8'
14	-	-	-	-	-	-	30.19'	27.32'	25.26'	16'

Tabella 5.3: Tempi di esecuzione *Escherichia coli* 2-14 genomi - implementazione del parallelismo nel confronto tra coppie di genomi nella fase di pre-filtering

Dim	1	2	4	8	16	32	48	64	96	PanDelos
2	-	-	-	4.47’	2.32’	1.2’	58”	49”	32”	20”
4	-	-	-	24.5’	12.4’	6.31’	4.31’	3.42’	2.31’	1’
6	-	-	-	-	18’	9.13’	6.23’	5.15’	3.38’	2.1’
8	-	-	-	-	-	13.34’	9.22’	7.46’	5.19’	4’
10	-	-	-	-	-	-	10.54’	9’	6.2’	5.1’
12	-	-	-	-	-	-	13.31’	11.16’	7.49’	8’
14	-	-	-	-	-	-	18.14’	15.9’	10.32’	16’
16	-	-	-	-	-	-	24’	20’	14’	19’

Tabella 5.4: Tempi di esecuzione *Escherichia coli* 2-16 genomi - implementazione del parallelismo nel confronto tra coppie di genomi nella fase di pre-filtering

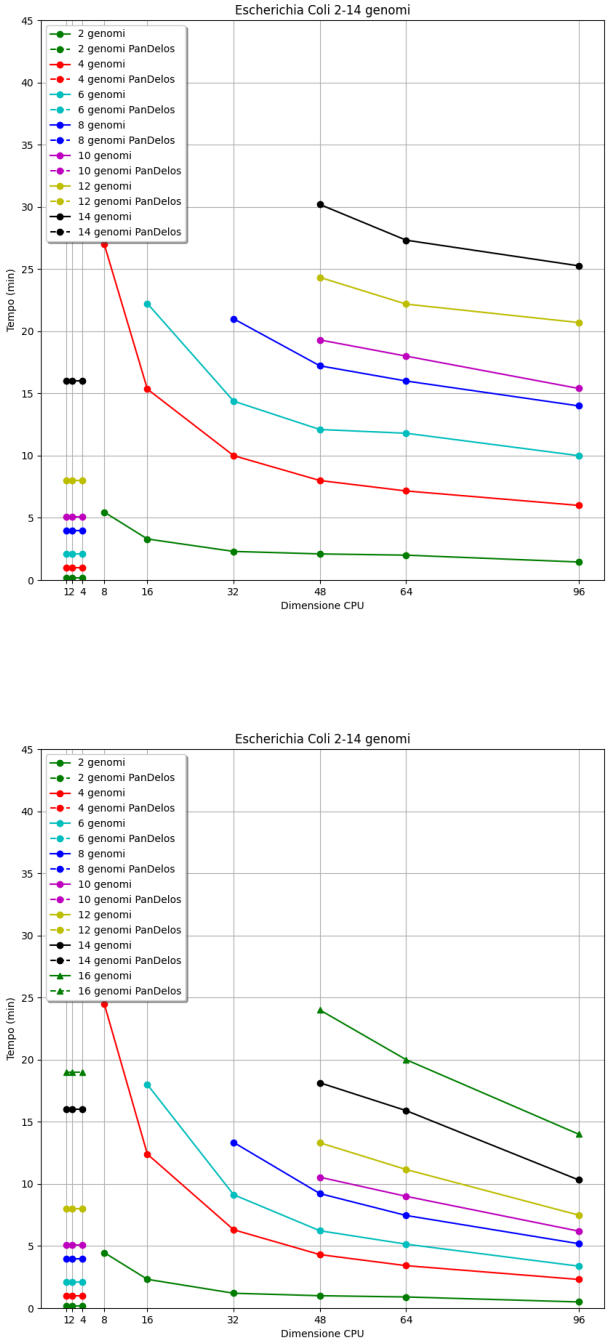

Figura 5.3: Tempi di esecuzione *Escherichia coli* 2-14 genomi - implementazione del parallelismo nel confronto tra coppie di genomi e geni nella fase di pre-filtering

5.4 Requisiti di memoria RAM

I requisiti di memoria RAM del software sviluppato sono stati misurati con il comando */usr/bin/time -f"%M"*, il quale restituisce il picco massimo di memoria richiesta da un processo.

Dai dati raccolti si evidenzia una crescita dei requisiti di memoria RAM al crescere del numero di unità di calcolo utilizzate. Ciò è dato dalla presenza del *thread local storage*, cioè uno spazio di memoria privato posseduto da ogni thread. Ne consegue che l'utilizzo di molte unità di calcolo comporta un naturale aumento dei requisiti di memoria RAM. In aggiunta, tutti i thread effettuano, in parallelo, confronti tra coppie di genomi e/o tra coppie di geni e quindi ognuno necessiterà di spazio in RAM per i propri calcoli.

5.4.1 Mycoplasma genitalium

Le tabelle 5.5 e 5.6 mostrano un confronto dei requisiti di memoria RAM (in gigabyte) tra il software sviluppato e PanDelos, relativi ai test sperimentali condotti sulla specie *Mycoplasma genitalium*.

In questa tipologia di test, con un numero di genomi crescente, PanDelos presenta un aumento lineare della quota di memoria RAM richiesta.

Il software sviluppato, invece, nonostante per dimensioni di genomi modeste, abbia requisiti di memoria RAM inferiori rispetto a quelli di PanDelos, presenta globalmente un aumento esponenziale degli stessi.

Inoltre, l'implementazione del parallelismo nel confronto tra coppie di geni nella fase di pre-filtering ha, tendenzialmente, dei requisiti di memoria RAM più alti rispetto all'utilizzo dell'implementazione del parallelismo nel confronto tra coppie di genomi, perché utilizza ogni core a disposizione per confrontare coppie di geni.

Dim	1	2	4	8	16	32	48	64	96	PanDelos
2	0.08	0.08	0.08	0.08	0.09	0.09	0.09	0.10	0.10	0.34
4	0.15	0.15	0.16	0.16	0.17	0.18	0.19	0.21	0.22	1.01
6	0.24	0.25	0.25	0.25	0.26	0.27	0.28	0.31	0.32	2.04
8	0.30	0.30	0.31	0.31	0.32	0.33	0.34	0.37	0.38	3.07
10	0.42	0.42	0.42	0.42	0.43	0.33	0.34	0.37	0.38	3.08
12	0.46	0.46	0.47	0.47	0.48	0.49	0.50	0.54	0.54	3.07
14	0.51	0.51	0.51	0.52	0.52	0.54	0.55	0.59	0.60	3.05
16	0.56	0.56	0.56	0.56	0.57	0.59	0.60	0.63	0.65	3.10
32	-	1.07	1.07	1.07	1.08	1.10	1.12	1.15	1.18	3.19
64	-	-	2.18	2.18	2.19	2.21	2.23	2.26	2.29	3.35
128	-	-	-	4.75	4.75	4.78	4.80	4.82	4.86	3.50

Tabella 5.5: Requisiti di memoria RAM *Mycoplasma genitalium* - implementazione del parallelismo nel confronto tra coppie di genomi nella fase di pre-filtering

Dim	1	2	4	8	16	32	48	64	96	PanDelos
2	0.04	0.04	0.05	0.06	0.06	0.07	0.07	0.08	0.09	0.34
4	0.11	0.11	0.10	0.09	0.11	0.14	0.15	0.16	0.19	1.01
6	0.18	0.17	0.18	0.18	0.19	0.20	0.22	0.23	0.27	2.04
8	0.21	0.22	0.22	0.23	0.23	0.24	0.26	0.27	0.31	3.07
10	0.31	0.32	0.32	0.31	0.33	0.34	0.34	0.37	0.41	3.08
12	0.33	0.33	0.34	0.34	0.35	0.36	0.38	0.39	0.43	3.07
14	0.42	0.42	0.42	0.42	0.42	0.42	0.42	0.42	0.45	3.05
16	0.44	0.44	0.44	0.44	0.44	0.44	0.44	0.44	0.48	3.10
32	-	1.39	1.39	1.39	1.39	1.39	1.39	1.39	1.39	3.19
64	-	-	4.88	4.88	4.88	4.88	4.88	4.88	4.88	3.35
128	-	-	-	18.30	18.30	18.30	18.30	18.302	18.30	3.50

Tabella 5.6: Requisiti di memoria RAM *Mycoplasma genitalium* - implementazione del parallelismo nel confronto tra coppie di geni nella fase di pre-filtering

5.4.2 Escherichia coli

Le tabelle 5.7 e 5.8 mostrano un confronto dei requisiti di memoria RAM tra il software sviluppato e PanDelos, relativi ai test sperimentali condotti sulla specie *Escherichia coli*.

In questa tipologia di test, diversamente dalla precedente, PanDelos presenta un aumento esponenziale della quota di memoria RAM richiesta, con un picco di circa 63 Gb per elaborare un dataset composto da 16 genomi.

Anche il software sviluppato mantiene, come per la precedente tipologia di test, un aumento esponenziale. Tuttavia, i requisiti di memoria RAM sono sempre inferiori rispetto a quelli di PanDelos, soprattutto quando viene utilizzata l'implementazione del parallelismo nel confronto tra coppie di genomi, che tuttavia, contrariamente all'implementazione del parallelismo nel confronto tra coppie di geni, è più lenta.

Dim	1	2	4	8	16	32	48	64	96	PanDelos
2	-	-	-	0.88	0.90	0.90	0.91	0.92	0.92	4.81
4	-	-	-	1.72	1.73	1.73	1.74	1.74	1.75	7.59
6	-	-	-	-	2.23	2.26	2.30	2.31	2.33	6.87
8	-	-	-	-	-	3.22	3.23	3.24	3.25	6.78
10	-	-	-	-	-	-	3.51	3.56	3.58	8.56
12	-	-	-	-	-	-	3.91	3.92	3.92	54.73
14	-	-	-	-	-	-	5.87	5.88	5.89	60.41

Tabella 5.7: Requisiti di memoria RAM *Escherichia coli* - implementazione del parallelismo nel confronto tra coppie di genomi nella fase di pre-filtering

Dim	1	2	4	8	16	32	48	64	96	PanDelos
2	-	-	-	0.59	0.60	0.60	0.61	0.63	0.66	4.81
4	-	-	-	2.55	2.55	2.55	2.55	2.55	2.55	7.59
6	-	-	-	-	4.87	4.87	4.87	4.87	4.88	6.87
8	-	-	-	-	-	9.35	9.35	9.35	9.45	6.78
10	-	-	-	-	-	-	9.45	9.45	9.45	8.56
12	-	-	-	-	-	-	18.21	18.21	18.21	54.73
14	-	-	-	-	-	-	35.58	35.58	35.58	60.41
16	-	-	-	-	-	-	35.84	35.84	35.84	63.34

Tabella 5.8: Requisiti di memoria RAM *Escherichia coli* - implementazione del parallelismo nel confronto tra coppie di geni nella fase di pre-filtering

5.5 Contenuto pangenomico

Dopo aver analizzato i confronti tra le performance dei due software, è necessario analizzare qualitativamente la nuova metodologia e confrontarla con PanDelos, al fine di evidenziare eventuali incongruenze nei risultati.

Infatti, la sostituzione delle strutture dati, originariamente poco parallelizzabili, ha portato ad una leggera modifica nelle funzionalità del software.

In particolare, è state eliminata la soglia definita nella sezione 3.2.2, ed è stata introdotta la soglia definita nella sezione 4.3.2.

Per effettuare questo tipo di analisi, ad esempio con la specie *Mycoplasma genitalium*, è possibile generare diversi dataset sintetici di diversa dimensione con *PANPROVA* e confrontare le *distribuzioni pangenomiche* che producono i due software con quella prodotta dal generatore.

Di seguito sono presenti i grafici che sintetizzano tali analisi, condotte con dataset costituiti da 2 a 128 genomi di *Mycoplasma genitalium*.

I grafici presenti nella figura 5.4 si riferiscono al confronto delle distribuzioni pangenomiche relative a 2 e 4 genomi. In queste due distribuzioni si presenta una perfetta sovrapposizione tra i risultati.

I grafici presenti nella figura 5.5 si riferiscono al confronto delle distribuzioni pangenomiche relative a 8 e 16 genomi. In queste due distribuzioni sono presenti piccole discrepanze per quanto riguarda le parti di sinistra e destra della distribuzione, mentre le parti centrali sono esattamente uguali.

I grafici presenti nella figura 5.6 si riferiscono al confronto delle distribuzioni pangenomiche relative a 32 e 64 genomi. Nel secondo grafico si evidenzia uno scostamento, dalla distribuzione pangenomica di *PANPROVA*, verso le parti iniziali e finali delle due distribuzioni. In particolare, il software sviluppato tende ad estrarre più geni singleton e meno geni core.

Il grafico presente nella figura 5.7 si riferisce al confronto delle distribuzioni pangenomiche relative a 128 genomi. In questo caso, il software sviluppato tende ad estrarre più geni core rispetto a PanDelos, allineandosi meglio alla distribuzione pangenomica di *PANPROVA*.

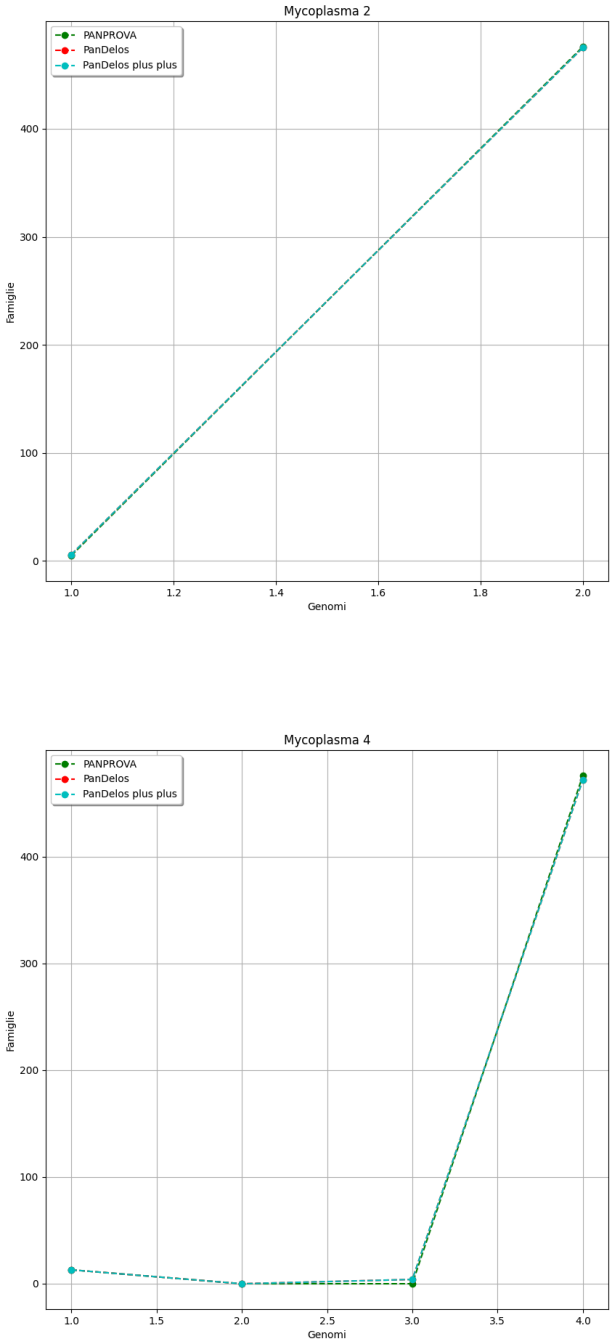

Figura 5.4: Confronto delle distribuzioni pangenomiche - *Mycoplasma genitalium*, 2-4 genomi

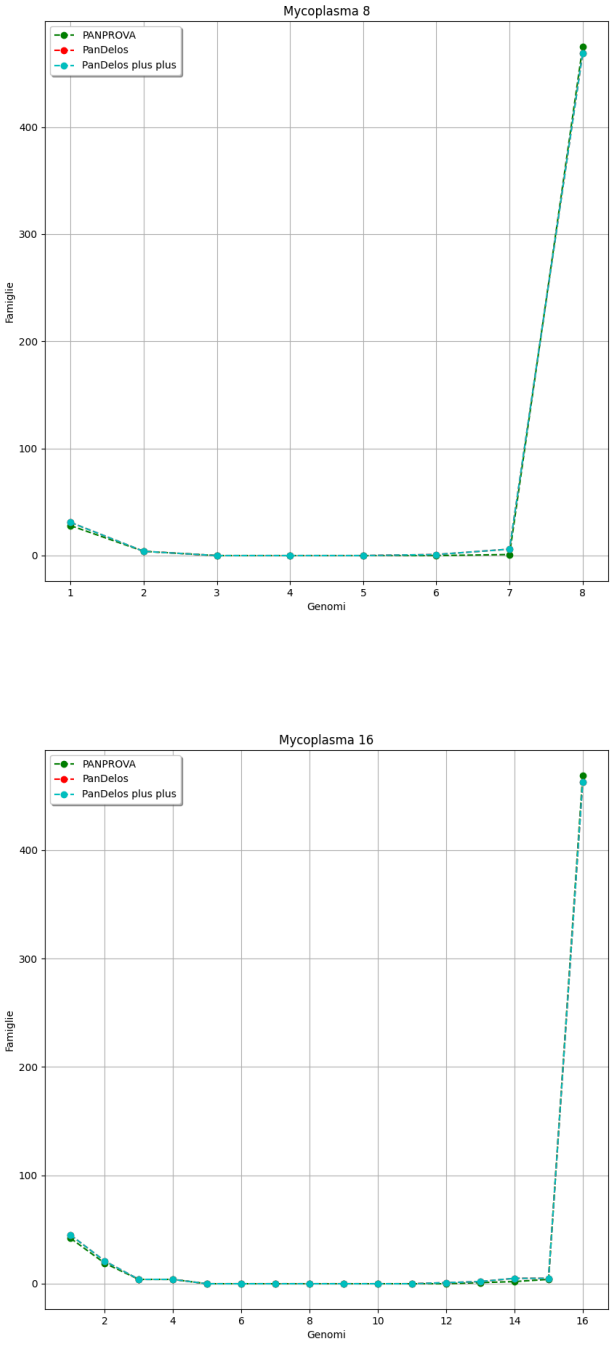

Figura 5.5: Confronto delle distribuzioni pangenomiche - *Mycoplasma genitalium*, 8-16 genomi

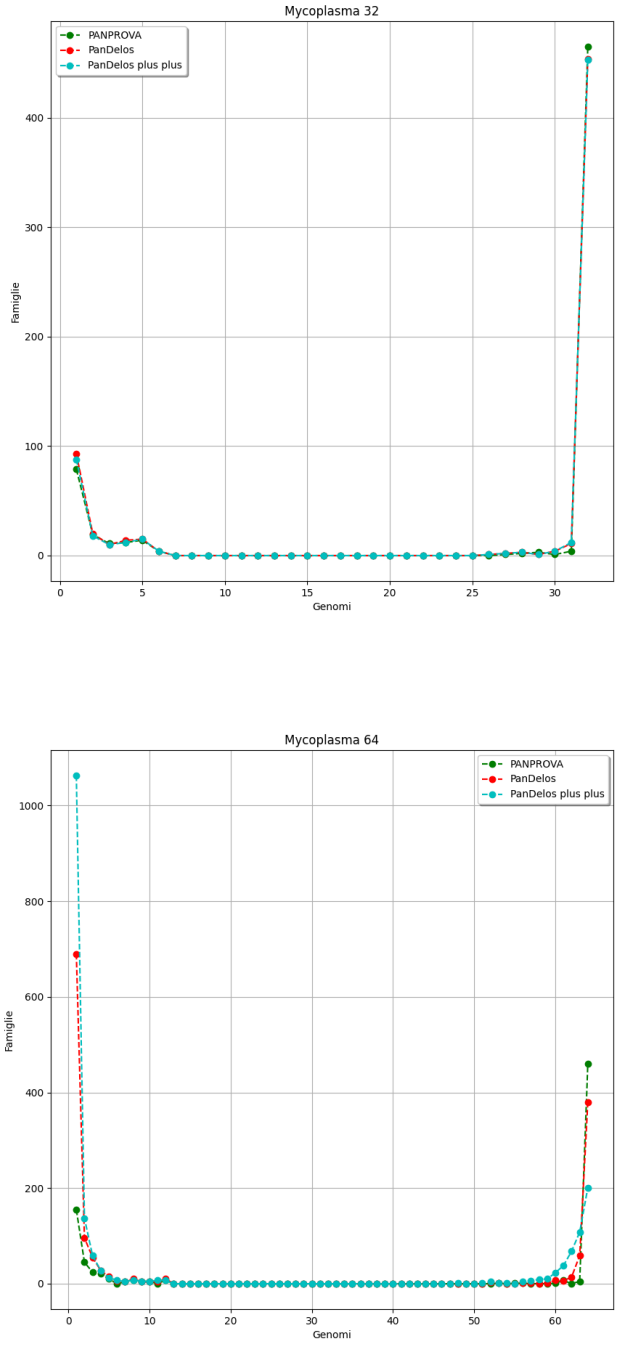

Figura 5.6: Confronto delle distribuzioni pangenomiche - *Mycoplasma genitalium*, 32-64 genomi

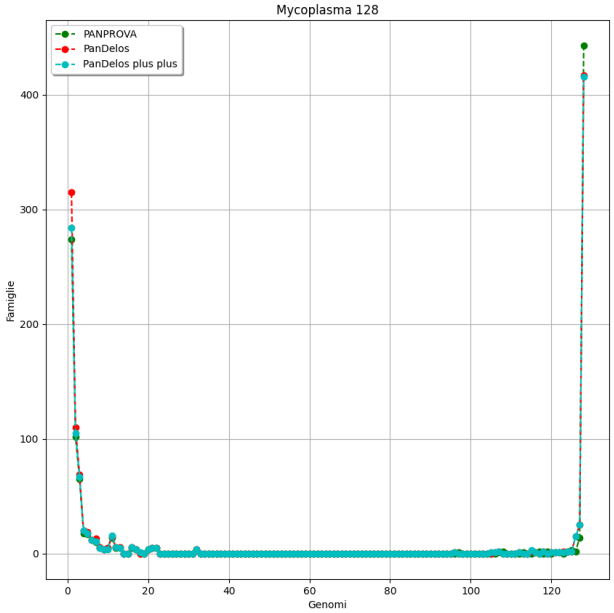

Figura 5.7: Confronto delle distribuzioni pangenomiche - *Mycoplasma genitalium*, 128 genomi

Conclusione

L'obiettivo principale di questa tesi è stato ricercare ed implementare una metodologia scalabile per calcolare l'omologia di sequenza tra geni. In tale ambito, sono state affrontate problematiche di ricerca ed implementazione di algoritmi per la scoperta del contenuto pangenomico.

Le soluzioni ricercate sono state progettate e sviluppate con lo scopo di massimizzare l'efficienza computazionale e minimizzare, quindi, l'utilizzo delle risorse hardware.

Tale scopo è stato ottenuto massimizzando il grado di parallelismo delle strutture dati e, quindi, degli algoritmi applicati per il calcolo della omologia.

Lo stato dell'arte da cui è partita la ricerca, è PanDelos, uno dei migliori software, a livello qualitativo, per la scoperta del contenuto pangenomico [Bonnici et al., 2018].

In particolare, dopo aver analizzato nel dettaglio il funzionamento del software e aver condotto test sperimentali preliminari al fine di identificare eventuali criticità, la prosecuzione del lavoro è stata caratterizzata dalla ricerca di una nuova metodologia scalabile per calcolare l'omologia di sequenza tra geni.

La metodologia individuata rispetta le caratteristiche dei metodi per la scoperta del contenuto pangenomico, in quanto per ogni gene permette di trovare i geni più simili all'interno degli altri genomi. Tale metodologia, inoltre, appartiene agli approcci *senza allineamento* ed è basata sull'enumerazione e analisi dei k-meri appartenenti a coppie di sequenze geniche, al fine di quantificare una similarità e inferire una omologia.

Dai test sperimentali condotti, i risultati ottenuti mostrano che, ad esempio, per la specie *Mycoplasma genitalium*, la metodologia individuata risulta vantaggiosa (in termini di risorse hardware richieste) a partire da 16 genomi in input, in quanto sono necessari almeno 12 core per ottenere tempi migliori rispetto a PanDelos. Inoltre, è emerso che a parità del numero di core utiliz-

zati, la nuova metodologia utilizza meno memoria RAM.

Riguardo la qualità dei risultati, sono stati eseguiti dei test preliminari al fine di confrontare le distribuzioni pangenomiche prodotte dal software sviluppato con quelle prodotte da PanDelos e *PANPROVA*.

Da tali test è emerso che nella maggior parte dei casi, il software sviluppato produce risultati compatibili con quelli prodotti da PanDelos, anche se uno dei possibili sviluppi futuri potrebbe riguardare una ricerca più approfondita di situazioni nelle quali il software sviluppato produce risultati qualitativamente migliori rispetto a quelli di PanDelos, o viceversa.

Altri possibili sviluppi futuri di questa tesi possono riguardare i seguenti aspetti:

- Ricerca di nuovi metodi di *decomposizione del dominio* relativo alle sequenze di geni in input;
- Ricerca di nuovi algoritmi più efficienti nell'utilizzo di risorse hardware e/o più performanti nelle attività di calcolo parallelo;
- Implementazione della metodologia ideata in un nuovo software che faccia uso della programmazione *CUDA* per sfruttare le potenzialità dell'altissima parallelizzazione dell'architettura delle GPU;
- Sviluppo di un sistema software per le analisi pangenomiche aperto alle estensioni di nuovi algoritmi per la scoperta del contenuto pangenomico (tra cui la metodologia presentata in questa tesi). Tale sistema software potrebbe adottare il modello di servizio del software *Software as a service*, consentendo agli utilizzatori di fornire i dati di input e ricevere i risultati attraverso API pubbliche o private.

Bibliografia

[Blom et al., 2016] Blom, J., Kreis, J., Spänig, S., Juhre, T., Bertelli, C., Ernst, C., and Goesmann, A. (2016). Edgar 2.0: an enhanced software platform for comparative gene content analyses. *Nucleic acids research*, 44(W1):W22–W28.

[Boguski, 1998] Boguski, M. S. (1998). Bioinformatics–a new era. *Trends in Biotechnology*, 16:1–3.

[Bonnici and Giugno, 2022] Bonnici, V. and Giugno, R. (2022). Panprova: pangenomic prokaryotic evolution of full assemblies. *Bioinformatics*, 38(9):2631–2632.

[Bonnici et al., 2018] Bonnici, V., Giugno, R., and Manca, V. (2018). Pandelos: a dictionary-based method for pan-genome content discovery. *BMC bioinformatics*, 19(15):47–59.

[Bonnici et al., 2020] Bonnici, V., maresi, e., and Giugno, R. (2020). Challenges in gene-oriented approaches for pangenome content discovery. *Briefings in Bioinformatics*, 22:1–11.

[Brandes, 2001] Brandes, U. (2001). A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177.

[Brazell and Bettersworth, 2008] Brazell, J. and Bettersworth, M. (2008). High performance computing. *Technical brief, Texas State Technical College–TSTC Forecasting*.

[Cavallaro,] Cavallaro. Genomica e genomica funzionale.

[Dalquen et al., 2012] Dalquen, D. A., Anisimova, M., Gonnet, G. H., and Dessimoz, C. (2012). Alf—a simulation framework for genome evolution. *Molecular biology and evolution*, 29(4):1115–1123.

[Ereshefsky, 1990] Ereshefsky, M. (1990). Ernst mayr. toward a new philosophy of biology. cambridge, mass.: Belknap/harvard university press, 1988, 564 pp. $35.00 (cloth). *Philosophy of Science*, 57(4):725–727.

[Holt et al., 2008] Holt, K. E., Parkhill, J., Mazzoni, C. J., Roumagnac, P., Weill, F.-X., Goodhead, I., Rance, R., Baker, S., Maskell, D. J., Wain, J., et al. (2008). High-throughput sequencing provides insights into genome variation and evolution in salmonella typhi. *Nature genetics*, 40(8):987–993.

[Leskovec et al., 2009] Leskovec, J., Lang, K. J., Dasgupta, A., and Mahoney, M. W. (2009). Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123.

[Manber and Myers, 1993] Manber, U. and Myers, G. (1993). Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948.

[McNaught et al., 1997] McNaught, A. D., Wilkinson, A., et al. (1997). *Compendium of chemical terminology*, volume 1669. Blackwell Science Oxford.

[Medini et al., 2005] Medini, D., Donati, C., Tettelin, H., Masignani, V., and Rappuoli, R. (2005). The microbial pan-genome. *Current opinion in genetics & development*, 15(6):589–594.

[Mira et al., 2010] Mira, A., Mart'ın-Cuadrado, A. B., D'Auria, G., and Rodr'ıguez-Valera, F. (2010). The bacterial pan-genome: a new paradigm in microbiology. *Int Microbiol*, 13(2):45–57.

[Page et al., 2015] Page, A. J., Cummins, C. A., Hunt, M., Wong, V. K., Reuter, S., Holden, M. T., Fookes, M., Falush, D., Keane, J. A., and Parkhill, J. (2015). Roary: rapid large-scale prokaryote pan genome analysis. *Bioinformatics*, 31(22):3691–3693.

[Serruto et al., 2009] Serruto, D., Serino, L., Masignani, V., and Pizza, M. (2009). Genome-based approaches to develop vaccines against bacterial pathogens. *Vaccine*, 27(25-26):3245–3250.

[Silvestroni and Pasquali, 1996] Silvestroni, P. and Pasquali, M. (1996). *Fondamenti di chimica*. Veschi.

[Tettelin et al., 2005] Tettelin, H., Masignani, V., Cieslewicz, M. J., Donati, C., Medini, D., Ward, N. L., Angiuoli, S. V., Crabtree, J., Jones, A. L., Durkin, A. S., et al. (2005). Genome analysis of multiple pathogenic isolates of streptococcus agalactiae: implications for the microbial "pan-genome". *Proceedings of the National Academy of Sciences*, 102(39):13950–13955.

[Vincenzo Bonnici, 2015] Vincenzo Bonnici, V. M. (2015). Infogenomics tools: A computational suite for informational analyses of genomes. *Bioinformatics, Proteomics and Immaging Analysis*, 1:7–14.