

Integrazione e Ottimizzazione dell'Algoritmo LROD Per il Rilevamento delle Sovrapposizioni nelle Letture Lunghe: Un Approccio Senza Dipendenze Esterne

Progetto di bioinformatica - Manuel Sica matr. 0522501870

COSA VEDREMO

1

Introduzione

2

LROD

3

Modifiche al codice
LROD

4

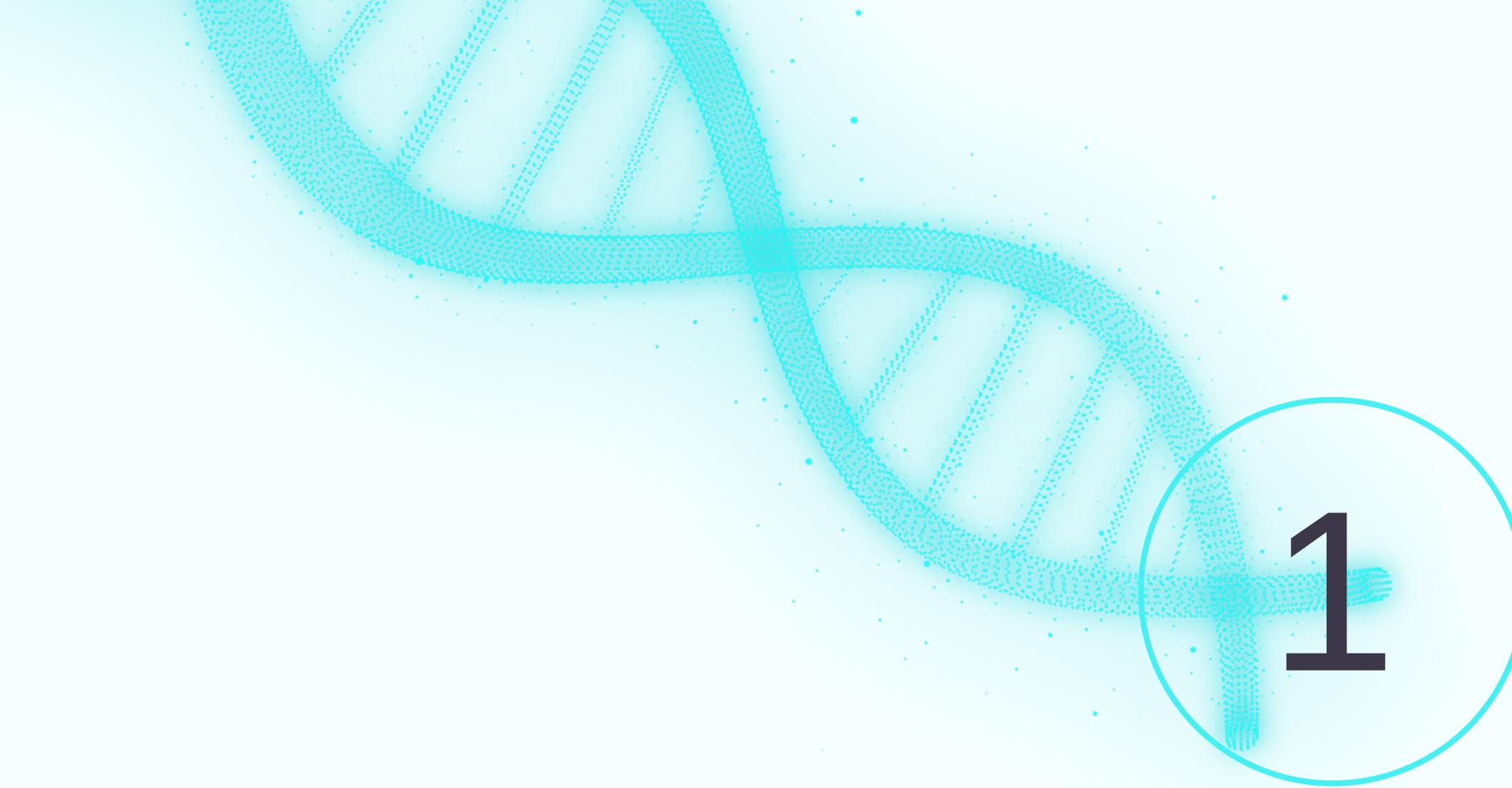
Implementazione di
LROD senza l'Uso di
DSK

5

Risultati ottenuti

6

Conclusioni e
sviluppi futuri



1

INTRODUZIONE

Introduzione

Le tecnologie di sequenziamento di terza generazione (TGS) come SMRT e ONT producono letture molto lunghe, cruciali per l'assemblaggio del genoma. Tuttavia, queste letture sono affette da un alto tasso di errore di sequenziamento, complicando il rilevamento accurato delle sovrapposizioni.

L'algoritmo LROD (Long Read Overlap Detection) migliora l'accuratezza del rilevamento delle sovrapposizioni tra lunghe letture analizzando la distribuzione dei k-mer. In origine, LROD richiedeva l'uso del software DSK per calcolare le frequenze dei k-mer.

COSA SONO I K-MERS?

Un k-mer è una sottosequenza di lunghezza k derivata da una sequenza di DNA, RNA o proteine. Nel contesto del sequenziamento genomico, i k-mer sono utilizzati per rappresentare frammenti di sequenze più lunghe.

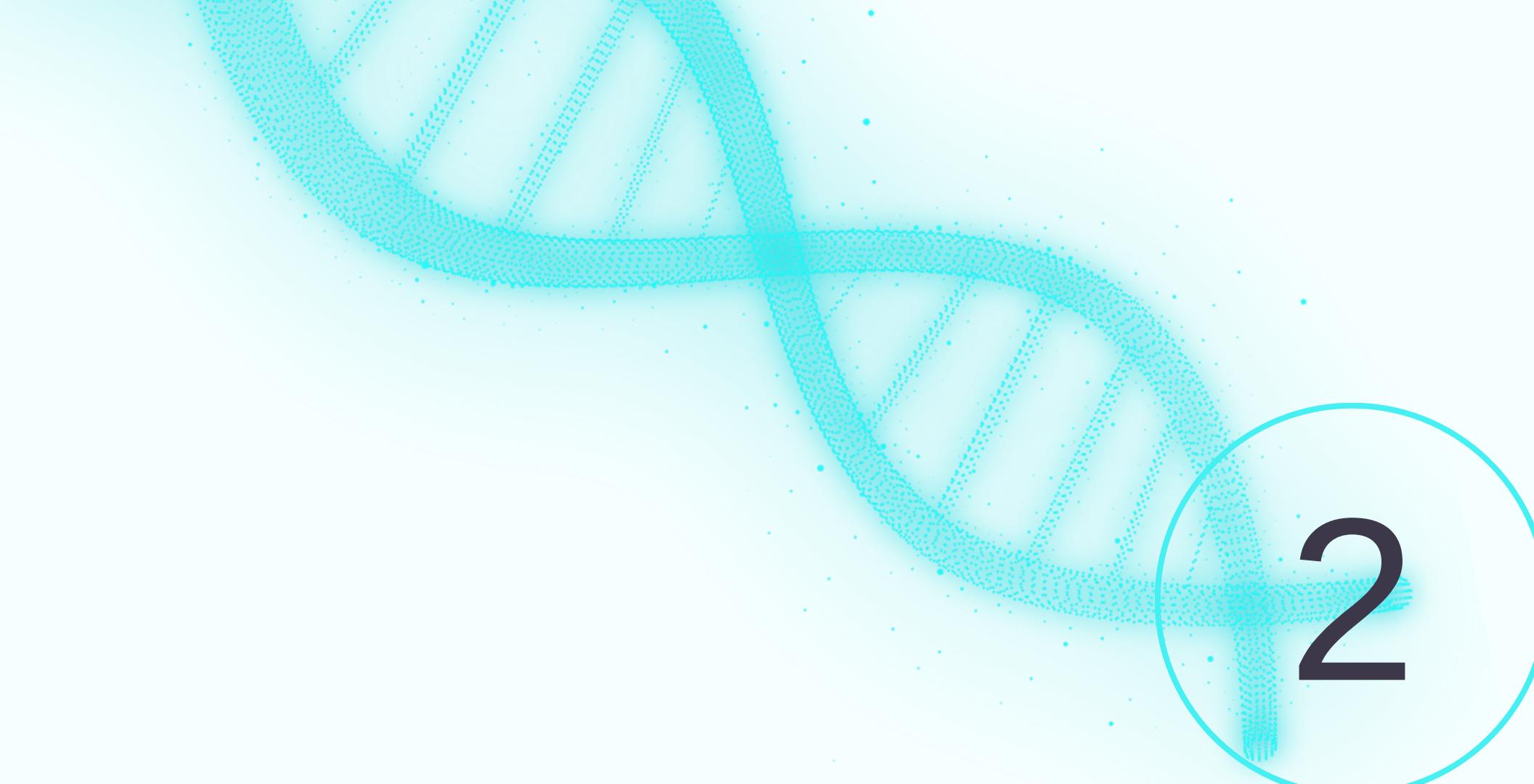
La scelta del valore di k è cruciale.

- Un valore troppo piccolo può non catturare sufficientemente le caratteristiche uniche della sequenza, mentre un valore troppo grande può ridurre il numero di k-mer comuni rilevati tra le sequenze, rendendo più difficile il rilevamento delle sovrapposizioni.

LROD utilizza i k-mers solidi ovvero quei k-mer che hanno una frequenza compresa tra due soglie predefinite, f_{min} e f_{max} . Questi k-mer sono considerati affidabili per l'analisi perché:

- Frequenze basse (inferiori a f_{min}) possono indicare errori di sequenziamento.
- Frequenze alte (superiori a f_{max}) possono indicare regioni ripetitive del genoma.

Selezionando solo i k-mer con frequenze all'interno di questo intervallo, si riduce l'impatto degli errori di sequenziamento e delle regioni ripetitive.



2

LROD

L'ALGORITMO SI ARTICOLA IN TRE FASI PRINCIPALI

1

Selezione dei k-mer Solidi

Si scelgono i k-mer con frequenze comprese tra due soglie, f_{min} e f_{max} , per ridurre l'impatto degli errori di sequenziamento e delle regioni ripetitive.

2

Costruzione della Catena di k-mer

Si cerca una catena di k-mer consistenti che rappresentano una potenziale sovrapposizione tra due letture.

3

Valutazione delle Sovrapposizioni

Si utilizzano due fasi per determinare la consistenza dei k-mer e si revisionano le sovrapposizioni candidate per ottenere quelle reali.

Selezione dei k-mer Solidi

Ridurre l'impatto degli errori di sequenziamento e delle regioni ripetitive selezionando solo i k-mer con frequenze adeguate.

Calcolo della Frequenza dei k-mer:

Utilizzo di strumenti come DSK per determinare quante volte ogni k-mer appare nel dataset di letture lunghe.

Determinazione degli Intervalli di Frequenza:

Eliminazione dei k-mer con frequenze al di fuori dell'intervallo predefinito [fmin,fmax]

Costruzione della Tabella Hash dei k-mer:

Indicizzazione dei k-mer solidi in una tabella hash per una ricerca rapida durante il confronto tra letture.

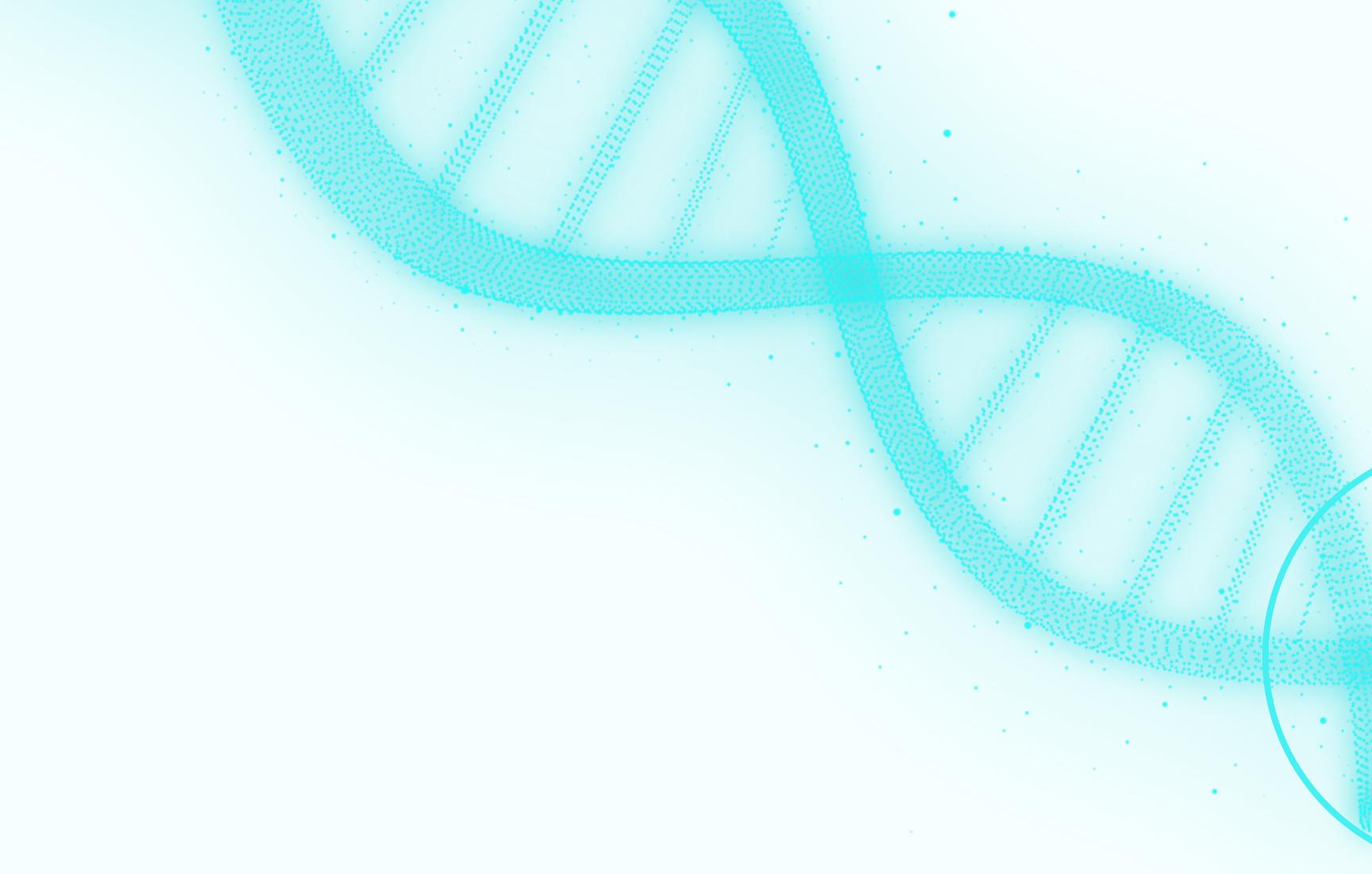
Rilevamento delle Sovrapposizioni

Identificare i k-mer comuni tra due letture e costruire catene di k-mer consistenti che rappresentano potenziali sovrapposizioni

Rilevamento dei k-mer Comuni

Rimozione dei k-mer Comuni Ripetitivi

Creazione di Catene di k-mer Consistenti



3

Modifiche al codice LROD

Modifiche nel file read.cpp

La funzione originale getReadSetHead eseguiva le seguenti operazioni:

1. Apertura del file long_read.fa e conteggio del numero di letture presenti nel file.
2. Chiusura del file.
3. Riapertura del file long_read.fa per eliminare i caratteri di terminazione (\n o \r) e popolare la struttura readSetHead con le letture.

Questa doppia lettura del file non era necessaria e introduceva un overhead aggiuntivo.

```
printf("\nfunzione GetReadSetHead\n");
ReadSetHead * readSetHead = (ReadSetHead *)malloc(sizeof(ReadSetHead));

FILE *fp;
long int m=1;
if((fp = fopen(filename,"r")) == NULL){
    printf("error!");
    return NULL;
}
```

```
FILE *fp1; //apre il file della long read
if((fp1 = fopen(filename,"r")) == NULL){
    printf("error!");
    return NULL;
}

long int readIndex = -1;
//long int j = 0;
long int allReadLen = 0;
```

Modifiche nel file read.cpp

La funzione getReadSetHead è stata migliorata per eseguire conteggio delle letture ed eliminazione dei caratteri di terminazione in un solo ciclo di lettura del file. Le modifiche principali includono:

1. Apertura del file una sola volta.
2. Durante la lettura, conteggio delle letture e popolazione della struttura readSetHead contemporaneamente.
3. Implementazione di riallocazione dinamica della memoria per readSet: iniziale allocazione per 100 letture, con incremento di blocchi di 100 letture se necessario.
4. Aggiunta di controlli per gestione corretta della memoria e prevenzione di fughe di memoria.

```
FILE *fp;
if ((fp = fopen(filename, "r")) == NULL) {
    printf("error opening file!\n");
    free(readSetHead);
    return NULL;
}

long int readIndex = -1;
long int allocatedReads = 0;

// Conta le reads e popola la struttura readSetHead in un unico ciclo
while ((fgets(StrLine, maxSize, fp)) != NULL) { // legge maxSize caratteri in ogni ciclo
    if (StrLine[0] == '>') { // se il primo carattere è > -> è una read -> incrementa il contatore
        readSetHead->readCount++;
        readIndex++;
        if (readIndex >= allocatedReads) {
            // Rialloca memoria per readSet
            allocatedReads += 100; // incrementa di 100000 alla volta per ridurre le riallocazioni
            readSetHead->readSet = (ReadSet *)realloc(readSetHead->readSet, sizeof(ReadSet) * allocatedReads);
            if (!readSetHead->readSet) {
                printf("Memory allocation error!\n");
                fclose(fp);
                free(readSetHead);
                return NULL;
            }
            // Inizializza le nuove reads allocate
            for (long int i = readIndex; i < allocatedReads; i++) {
                readSetHead->readSet[i].readLength = 0;
                readSetHead->readSet[i].read = NULL; // Inizializza i puntatori a NULL
            }
        }
        continue;
    }
}
```

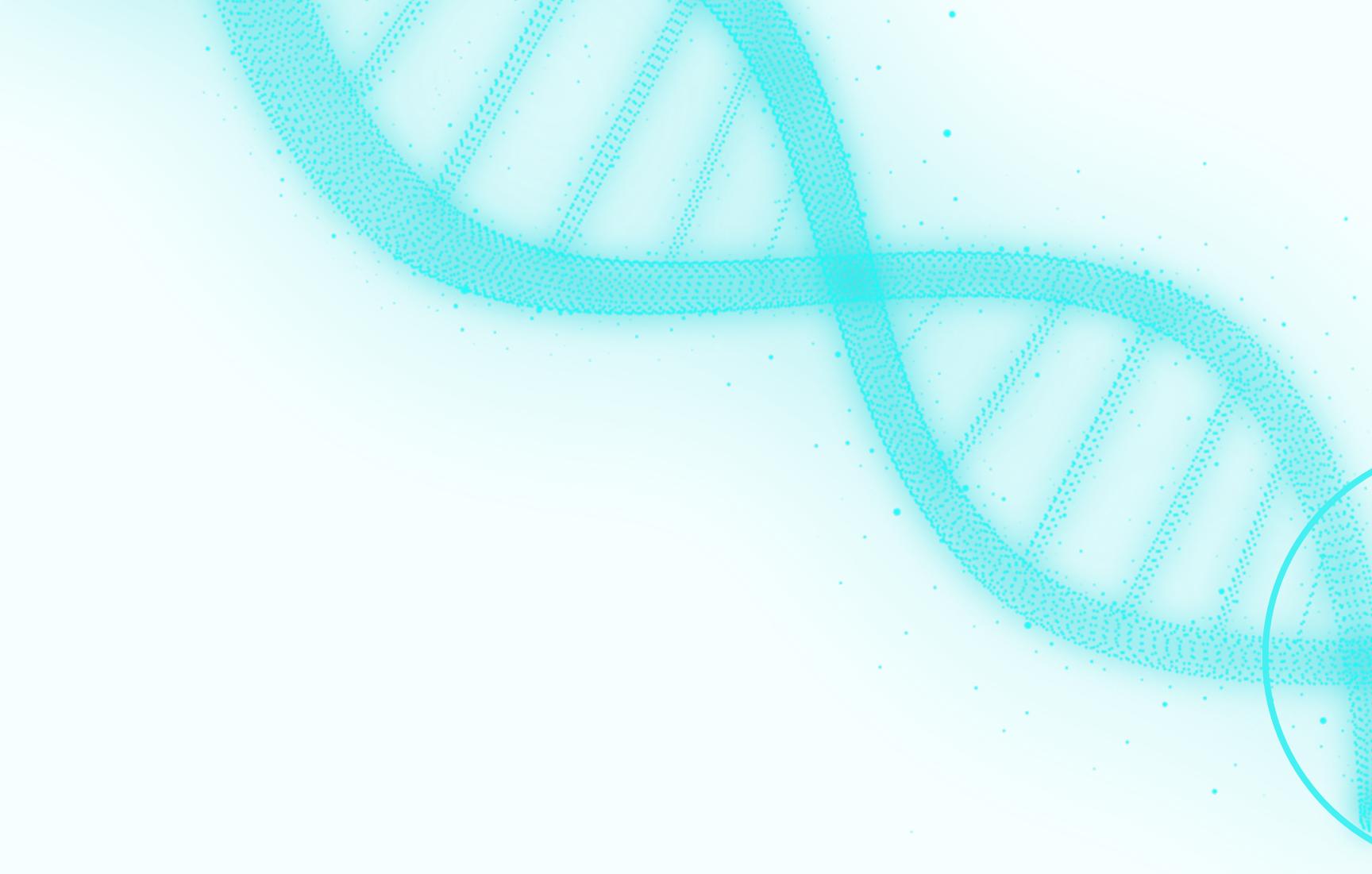
Modifiche nel file kmer.cpp

La funzione originale GetKmerHashTableHead presentava le seguenti problematiche:

1. Prima Apertura del File: Copiava solo le frequenze (kmerF) dei k-mer e le analizzava. Faceva un break se almeno un carattere della frequenza era diverso, ma non escludeva correttamente i k-mer con frequenze formate da cifre uguali (ad esempio, 111, 222) perché la funzione DetectSameKmer confrontava erroneamente la lunghezza dei k-mer (15 caratteri) con la lunghezza delle frequenze (massimo 4 cifre).
2. Seconda Apertura del File: Copiava solo i k-mer (senza la frequenza) ed eseguiva correttamente il confronto sui caratteri del k-mer, escludendo quelli che differivano in almeno un carattere, ottenendo il conteggio totale dei k-mer.
3. Terza Apertura del File: Eseguiva nuovamente il confronto del punto precedente, poi convertiva i k-mer in bytes e li memorizzava in una tabella hash.

La funzione GetKmerHashTableHead è stata migliorata affinché:

1. Lettura Unica del File: Il file viene letto una sola volta, riducendo il tempo di I/O e migliorando l'efficienza.
2. Eliminazione dei k-mer con Frequenze Non Valide: Durante la lettura del file, i k-mer con frequenze formate da cifre uguali vengono correttamente esclusi.
3. Popolazione della Tabella Hash: I k-mer validi vengono convertiti in bytes e memorizzati nella tabella hash durante la stessa lettura del file.



4

Implementazione di LROD senza l'Uso di DSK

Implementazione di LROD senza l'Uso di DSK

Inizialmente, LROD utilizzava DSK per generare un file contenente le frequenze dei k-mer dalle lunghe letture. Questo processo avveniva in due passaggi:

1. Esecuzione di DSK per generare il file delle frequenze dei k-mer.
2. Conversione del file binario di output di DSK in un file di testo leggibile da LROD.

Per eliminare la dipendenza da DSK, abbiamo integrato il calcolo della frequenza dei k-mer direttamente nel codice

Abbiamo aggiunto funzioni per leggere le lunghe letture dal file di input, generare k-mer dalle sequenze e calcolare le loro frequenze.

Una funzione legge il file FASTA contenente le lunghe letture.

```
std::string readLongReadFile(const std::string& filePath) {
    std::ifstream file(filePath);
    std::string sequence;
    std::string line;

    while (std::getline(file, line)) {
        if (line[0] != '>') {
            sequence += line;
        }
    }

    return sequence;
}
```

Generazione dei k-mer

Una funzione genera i k-mer dalla sequenza di lettura specificata. Ad esempio, per una sequenza di DNA ACGTGCA e una lunghezza k-mer di 3, i k-mer generati saranno ACG, CGT, GTG, TGC e GCA

```
void generateKmerFrequencyFile(const std::string& inputFilePath, const std::string& outputPath, int kmerLength, int step) {
    std::ifstream inputFile(inputFilePath);
    std::ofstream outputFile(outputPath);
    std::string sequence = readLongReadFile(inputFilePath);
    std::unordered_map<std::string, int> kmerFrequencies = computeKmerFrequencies(sequence, kmerLength, step);

    for (const auto& entry : kmerFrequencies) {
        outputFile << entry.first << " " << entry.second << "\n";
    }
}
```

Calcolo delle Frequenze dei k-mer

Una funzione calcola le frequenze dei k-mer generati e le memorizza in una tabella hash. Ad esempio, se i k-mer generati sono ACG, CGT, ACG, la frequenza di ACG sarà 2 e quella di CGT sarà 1

```
std::unordered_map<std::string, int> computeKmerFrequencies(const std::string& sequence, int kmerLength, int step) {
    std::unordered_map<std::string, int> kmerFrequencies;
    NtHash nth(sequence, 1, kmerLength);

    while (nth.roll()) {
        std::string kmer = sequence.substr(nth.get_pos(), kmerLength);
        kmerFrequencies[kmer]++;
    }

    return kmerFrequencies;
}
```

NtHash è stato utilizzato per calcolare rapidamente e accuratamente gli hash per i k-mer estratti dalle lunghe letture, permettendo un calcolo efficiente delle frequenze dei k-mer.



5

Risultati ottenuti

Per i test sono stati utilizzati tre dataset: uno artificiale creato dagli sviluppatori di LROD denominato long_read.fa, un dataset di E. coli, e un dataset denominato w303. Il file delle frequenze dei k-mer è stato generato

```
// Funzione per ottenere l'uso della memoria
long getMemoryUsage() {
    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    return usage.ru_maxrss; // La memoria massima residente in KB
}

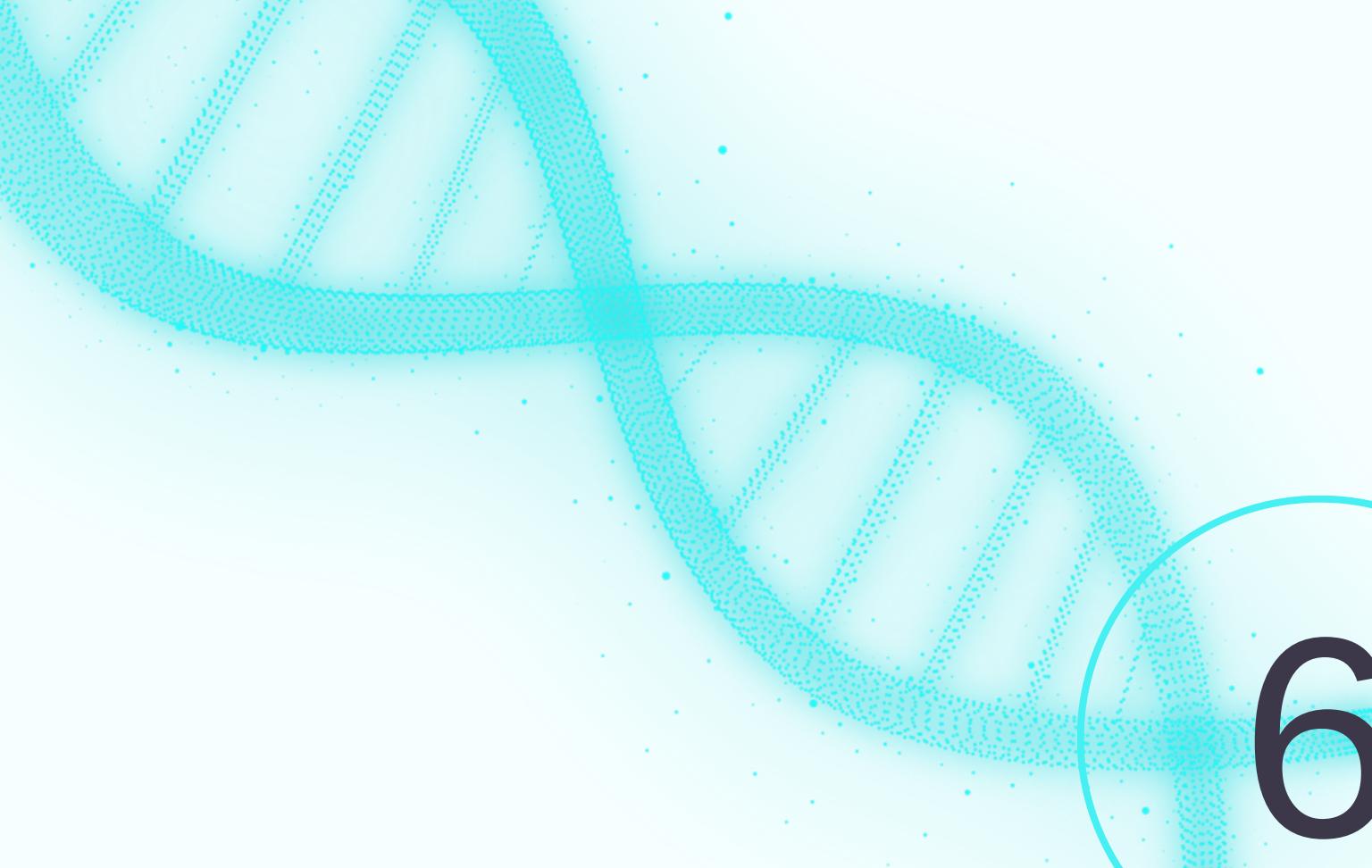
auto start = high_resolution_clock::now();
long initialMemoryUsage = getMemoryUsage();

auto end = high_resolution_clock::now();
long finalMemoryUsage = getMemoryUsage();
double elapsedSeconds = duration_cast<seconds>(end - start).count();
long memoryUsage = finalMemoryUsage - initialMemoryUsage;
```

```
// Funzione per scrivere i benchmark su un file CSV
void writeBenchmarkToCSV(double elapsedSeconds, long memoryUsage) {
    ofstream file("benchmark_results_w303.csv");
    if (file.is_open()) {
        file << "Running time,Memory (Mb)\n";
        file << elapsedSeconds << " s," << (memoryUsage / 1024) << " Mb\n";
        file.close();
    }
}
```

RISULTATI

CODICE	DATASET	TEMPO DI ESECUZIONE	MEMORIA
CODICE ORIGINALE	artificiale	4s	43424Mb
CODICE MODIFICATO	artificiale	1s	43424Mb
CODICE ORIGINALE	E. coli	18s	93184Mb
CODICE MODIFICATO	E. coli	12s	93184Mb
CODICE ORIGINALE	w303	124s	250912Mb
CODICE MODIFICATO	w303	93s	250000 Mb



6

Conclusioni e sviluppi futuri

Futuri Sviluppi

Ottimizzazione del Codice

Parallelizzazione e Multi-threading

Implementare il supporto per l'esecuzione parallela e il multi-threading per sfruttare al meglio le capacità dei processori multi-core moderni.

Algoritmi di Hashing Avanzati

Sperimentare con nuovi algoritmi di hashing nel codice kmer.cpp

Futuri Sviluppi

Espansione delle Applicazioni

**Sequenziamento
dell'RNA**

**Sequenziamento
delle Proteine**

Conclusioni

Le modifiche all'algoritmo LROD hanno integrato il calcolo delle frequenze dei k-mer, eliminando la dipendenza da strumenti esterni come DSK. Questo ha migliorato significativamente l'efficienza nel rilevamento delle sovrapposizioni nelle lunghe letture di DNA, riducendo i tempi di esecuzione senza aumentare il consumo di memoria. I test su vari dataset hanno mostrato che il nuovo approccio è più veloce e altrettanto accurato rispetto alla versione originale. Questo miglioramento è cruciale per l'analisi su larga scala di dataset complessi, rendendo LROD più efficace per l'assemblaggio del genoma.

The background features two sets of abstract particle waves. On the left, there are blue and white particles forming a diagonal band. On the right, there are pink and white particles forming a similar band. Both sets of waves have a dotted, textured appearance.

Grazie per
l'attenzione