Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Pronalazak mutacija pomoću treće generacije sekvenciranja

Tea Ćetojević-Tisaj

Karlo Vrančić

mentor: doc. dr. sc. Krešimir Križanović

Zagreb, lipanj 2024.

1. UVOD

U današnjoj molekularnoj biologiji i genetskim istraživanjima, analiza genomskih sekvenci postala je ključna za razumijevanje različitih bioloških procesa i bolesti. Sekvenciranje čitavog genoma je složen i skup proces koji se obično obavlja samo jednom kako bi se dobila referentna sekvenca. Nakon toga, sve daljnje analize obavljaju se uspoređivanjem očitanih sekvenci s referentnim genomom. Ovaj pristup omogućava uštedu vremena i resursa jer se umjesto ponovnog sekvenciranja čitavog genoma, provode samo manja očitanja koja se zatim poravnavaju s referentnom sekvencom kako bi se otkrile eventualne mutacije.

U ovom projektu za poravnanje očitanih sekvenci s referentnim genomom koristimo alat minimap2, koji je vrlo učinkovit u mapiranju kratkih sekvenci na velike referentne genome. Nakon što minimap2 izvrši poravnanje, koristimo vlastiti algoritam za evaluaciju tih poravnanja i detekciju mutacija. Algoritam je dizajniran da bude jednostavan i brz, koristeći osnovne programerske paradigme kako bi analizirao podatke dobivene iz minimap2 poravnanja. Ukratko, algoritam parsira SAM datoteke, obrađuje CIGAR stringove te predlaže i potvrđuje mutacije na temelju većinskog glasanja na svakoj poziciji.

Dakle, dok javno dostupni alati poput Freebayes-a koriste napredne statističke metode za detekciju varijacija, uzimajući u obzir različite izvore nesigurnosti što kao posljedicu ima ogroman utrošak računalnih resursa i vremena izvođenja.

U nastavku ćemo opisati naš algoritam te razmotriti potencijalnu kompenzaciju u preciznosti donošenja odluka zbog uštede vremena i resursa.

2. IMPLEMENTACIJA

2.1. Skripta za pokretanje i pozivanje vanjskih funkcija

Implementacija projekta i njegovo pokretanje je u potpunosti automatizirano pomoću shell skripte mutation-detector.sh. Ova skripta sadrži sve potrebne korake za izvršavanje detekcije mutacija, uključujući poravnanje očitanja, detekciju mutacija i evaluaciju rezultata.

```
#!/bin/bash

set -e

# Hard-coded paths for simplicity
reference="data/lambda.fasta"
reads="data/lambda_simulated_reads.fasta"
samOutput="data/alignment.sam"
bamOutput="data/alignment.bam"
sortedBam="data/alignment_sorted.bam"
csvOutput="data/mutations.csv"
vcfOutput="data/variants.vcf"
markedBam="data/alignment_marked.bam"

# Step 1: Run Minimap2
minimapCmd="minimap2 -ax map-ont $reference $reads > $samOutput"
echo "Running Minimap2: $minimapCmd"
eval $minimapCmd

# Step 2: Call detector to detect mutations
detectorCmd="./detector $samOutput $reference $csvOutput"
echo "Running mutation detection: $detectorCmd"
eval $detectorCmd

echo "Mutation detection completed. Results are stored in $csvOutput"

# Step 3: Generate index for the reference genome
indexCmd="samtools faidx $reference"
echo "Generating index for the reference genome..."
eval $indexCmd

# Step 4: Convert SAM to BAM
samToBamCmd="samtools view -bS $samOutput > $bamOutput"
echo "Converting SAM to BAM: $samToBamCmd"
eval $samToBamCmd

# Step 5: Sort BAM file
sortCmd="samtools sort -o $sortedBam $bamOutput"
echo "Sorting BAM file: $sortCmd"
eval $sortCmd

# Step 6: Index marked BAM file
indexMarkedCmd="samtools index $markedBam"
echo "Indexing marked BAM file: $indexMarkedCmd"
eval $indexMarkedCmd

# Step 7: Run FreeBayes for evaluation with optimization
freebayesCmd="freebayes -f $reference --use-best-n-alleles 4 --min-alternate-count 3 $markedBam > $vcfOutput"
echo "Running FreeBayes with optimization: $freebayesCmd"
eval $freebayesCmd

echo "Variant calling completed. Results are stored in $vcfOutput"
```

Minimap2 je alat za poravnanje očitanja na referentni genom. Ovdje ga koristimo s opcijom -ax map-ont za mapiranje Oxford Nanopore tehnologije sekvenciranja. Rezultat poravnanja je pohranjen u SAM datoteku alignment.sam. Za optimizaciju zastavica korištenih u Minimap2,

napisali smo skriptu koja testira različite kombinacije zastavica i računa postotke poklapanja s referentnim genomom. Na temelju rezultata testiranja, odabrali smo zastavicu -ax map-ont koja je pokazala najbolje performanse za naš skup podataka.

Nakon poravnanja, pokrećemo naš detektor mutacija koji koristi rezultirajuću SAM datoteku i referentni genom za detekciju mutacija. Rezultati su pohranjeni u CSV datoteku mutations.csv.

Kasnije koristimo `samtools` za generiranje indeksa za referentni genom i pretvorbu u BAM format koju zatim sortiramo i šaljemo freebayes-u.

FreeBayes se koristi za variant calling. Ovdje koristimo zastavice --use-best-n-alleles 4 (ograničava broj najboljih alela koji se koriste za pozivanje varijanti što može pomoći u smanjenju lažno pozitivnih rezultata) i --min-alternate-count 3 (postavlja minimalan broj očitanja koja podržavaju alternativni alel) za optimizaciju osjetljivosti i preciznosti. Rezultati su pohranjeni u VCF datoteku variants.vcf.

2.2. Algoritam za detekciju mutacija

Algoritam za detekciju mutacija glavna je komponenta ovog projekta te je pohranjen u datoteci detector.cpp.

U nastavku slijedi detaljno objašnjenje glavnih dijelova koda, funkcija po funkciju:

2.2.1. Funkcija main

```cpp
int main() {
    string samFile = "data/alignment.sam";
    string referenceFile = "data/lambda.fasta";
    string outputCsv = "data/mutations.csv";
    string logFilename = "data/mutation_detection_log.txt";

    // Parsiraj SAM datoteku
    vector<SamEntry> samEntries = parseSamFile(filename: samFile);

    // Učitaj referentni genom
    ifstream refFile(referenceFile);
    string reference;
    string line;
    while (getline(Istr: refFile, Str: line)) {
        if (line[0] != '>') {
            reference += line;
        }
    }

    // Detekcija mutacija
    detectMutations(samEntries, reference, outputCsv, logFilename);

    cout << "Mutation detection completed, output saved to " << outputCsv << endl;
    cout << "Log saved to " << logFilename << endl;
    return 0;
}
```

Glavna funkcija *main* je točka ulaska u program. Prvo definira nazive datoteka za ulazne i izlazne podatke. Zatim parsira SAM datoteku kako bi dobila očitanja sekvenci, učitava referentni genom iz FASTA datoteke, i konačno poziva funkciju detectMutations kako bi detektirala mutacije i zapisala rezultate.

2.2.2. Stuktura SamEntry

```cpp
    You, 4 days ago | 1 author (You)
11  struct SamEntry {
12      int flag;
13      int pos;
14      string cigar;
15      string seq;
16  };
```

Ova struktura predstavlja jedan zapis iz SAM datoteke. Iako se sama SAM datoteka sastoji od više dijelova, mi pamtimo samo relevantne: zastavicu (flag), poziciju na referentnom genomu (pos), CIGAR string (cigar) koji opisuje poravnanje, i sekvencu očitanja (seq).

2.2.3. Funkcija parseSamFile

```
// Funkcija za parsiranje SAM datoteke
vector<SamEntry> parseSamFile(const string& filename) {
    vector<SamEntry> entries;
    ifstream infile(filename);
    if (!infile) {
        cerr << "Error opening file: " << filename << endl;
        return entries;
    }

    string line;
    while (getline(Istr: infile, Str: line)) {
        if (line[0] == '@') continue; // preskoči zaglavlja

        istringstream iss(line);
        string qname, rname, cigar, rnext, seq, qual;
        int flag, pos, mapq, pnext, tlen;

        iss >> qname >> flag >> rname >> pos >> mapq >> cigar >> rnext >> pnext >> tlen >> seq >> qual;

        if (flag & 4) continue; // preskoči redove gdje je FLAG 4

        SamEntry entry;
        entry.flag = flag;
        entry.pos = pos;
        entry.cigar = cigar;
        entry.seq = seq;

        entries.push_back(Val: entry);
    }

    return entries;
}
```

Ova funkcija parsira SAM datoteku i ekstraktira relevantne informacije za svako očitanje koje je uspješno mapirano na referentni genom. Redovi koji započinju sa @ su zaglavlja i preskaču se. Informacije iz svakog očitanja pohranjuju se u SamEntry strukturu i dodaju u vektor *entries*.

2.2.4. Funkcija parseCigar

```
51  // Funkcija za obradu CIGAR stringa
52  vector<pair<char, int>> parseCigar(const string& cigar) {
53      vector<pair<char, int>> operations;
54      istringstream iss(cigar);
55      char op;
56      int count;
57      while (iss >> count >> op) {
58          operations.push_back(Val: std::make_pair(Val1: op, Val2: count));
59      }
60      return operations;
61  }
```

Ova funkcija parsira CIGAR string, koji opisuje poravnanje očitanja s referentnim genomom. Funkcija vraća vektor parova gdje svaki par sadrži operaciju (op) i broj baza na koje se operacija odnosi (count).

2.2.5. Struktura MutationProposal

```
    You, 4 days ago | 1 author (You)
63  struct MutationProposal {
64      int substitutionVotes = 0;
65      vector<char> substitutionBases;
66      int insertionVotes = 0;
67      vector<char> insertionBases;
68      int deletionVotes = 0;
69      int noneVotes = 0;
70  };
```

Ova struktura služi za pohranu prijedloga mutacija na određenoj poziciji. Sadrži brojače za glasove za zamjene (substitutionVotes), umetanja (insertionVotes), brisanja (deletionVotes) i poklapanja (noneVotes). Također, sadrži vektore baza koje su predložene za zamjene i umetanja kako bi se kasnije moglo većinskim glasanjem utvrditi koju bazu umećemo ili mijenjamo.

2.2.6. Funkcija addMutationProposal

```
void addMutationProposal(map<int, MutationProposal>& mutationProposals, int startPos, int readPos, const string& readSeq, const string& reference,
char mutationType, int count, ofstream& logFile) {
    logFile << "Adding mutation proposal at position " << startPos << " for type " << mutationType << " with count " << count << endl;
    for (int i = 0; i < count; ++i) {
        int pos = startPos + i;
        if (mutationType == 'M') {
            if (pos < reference.size() && readPos + i < readSeq.size() && reference[pos] == readSeq[readPos + i]) {
                mutationProposals[pos].noneVotes++;
            } else {
                mutationProposals[pos].substitutionVotes++;
                mutationProposals[pos].substitutionBases.push_back(Val: readSeq[readPos + i]);
            }
        } else if (mutationType == 'I') {
            mutationProposals[pos].insertionVotes++;
            if (readPos + i < readSeq.size()) {
                mutationProposals[pos].insertionBases.push_back(Val: readSeq[readPos + i]);
            }
        } else if (mutationType == 'D') {
            mutationProposals[pos].deletionVotes++;
        }
        logFile << "Position: " << pos << ", Mutation Proposal: {substitutionVotes: " << mutationProposals[pos].substitutionVotes
                << ", insertionVotes: " << mutationProposals[pos].insertionVotes << ", deletionVotes: " << mutationProposals[pos].deletionVotes
                << ", noneVotes: " << mutationProposals[pos].noneVotes << "}" << endl;
    }
}
```

Ova funkcija dodaje prijedloge mutacija za svaku poziciju u referentnom genomu. Na temelju tipa mutacije (mutationType), funkcija ažurira brojače glasova i pohranjuje predložene baze za zamjene i umetanja. Informacije o prijedlozima se također zapisuju u log datoteku radi praćenja.

2.2.7. Funkcija detectMutations

Funkcija detectMutations je glavna funkcija koja provodi detekciju mutacija.

S obzirom na njenu dužinu, razložit ćemo je u djelove koje ćemo opisati:

Njezina svrha je detektirati mutacije u očitanim sekvencama koje su poravnate s referentnim genomom. Ova funkcija prima vektor unosa iz SAM datoteke, referentni genom, ime izlazne CSV datoteke za zapisivanje mutacija i ime log datoteke za praćenje postupka.

```
// Funkcija za detekciju mutacija
void detectMutations(const vector<SamEntry>& samEntries, const string& reference, const string& outputCsv, const string& logFilename) {
    ofstream outfile(outputCsv);
    if (!outfile) {
        cerr << "Error opening file: " << outputCsv << endl;
        return;
    }

    ofstream logFile(logFilename);
    if (!logFile) {
        cerr << "Error opening file: " << logFilename << endl;
        return;
    }

    outfile << "type,X,pos,base\n";
    logFile << "type,X,pos,base\n";
```

Prvo se otvaraju dvije datoteke: jedna za zapisivanje detektiranih mutacija (CSV datoteka) i druga za logiranje (tekstualna datoteka).

```
map<int, MutationProposal> mutationProposals;
```

Inicijalizira se mapa mutationProposals, koja će pohranjivati prijedloge mutacija za svaku poziciju u referentnom genomu. Ključ je pozicija u genomu, a vrijednost je struktura MutationProposal koja sadrži brojače i baze predložene za zamjenu i umetanje.

```
for (const auto& const SamEntry &entry : samEntries) {
    int refPos = entry.pos - 1; // SAM format uses 1-based indexing
    int readPos = 0;
    string readSeq = entry.seq;

    if (entry.flag & 16) {
        // Reverzni komplement, preokreni sekvencu
        reverse(First: readSeq.begin(), Last: readSeq.end());
    }

    auto std::vector<s...r<char, int>> cigarOperations = parseCigar(entry.cigar);
    for (const auto& const std::pair<char, int> &op_count : cigarOperations) {
        char op = op_count.first;
        int count = op_count.second;

        // add output to log file so we can see what is happening
        logFile << "----------------------------------------------------" << endl;
        logFile << "Operation: " << op << ", Count: " << count << endl;


        addMutationProposal(mutationProposals, startPos: refPos, readPos, readSeq, reference, mutationType: op, count, logFile);

        if (op == 'M' || op == 'D') {
            refPos += count;
        }
        if (op == 'M' || op == 'I') {
            readPos += count;
        } else if (op == 'S') {
            readPos += count;
        }
    }
}
```

Ovdje iteriramo kroz sve unose iz SAM datoteke. refPos predstavlja trenutnu poziciju u referentnom genomu, prilagođenu iz 1-baznog indeksa (kako koristi SAM format) u 0-bazni indeks (kako koristi C++). readPos prati trenutnu poziciju unutar očitane sekvence. Ako je flag postavljen na 16, sekvenca je komplementarna i reverzna, pa je potrebno preokrenuti. CIGAR string se parsira u niz operacija pomoću funkcije parseCigar. Svaka operacija se sastoji od operacije (op) i broja baza na koje se odnosi (count). Ovisno o tipu operacije, pozivamo addMutationProposal za dodavanje prijedloga mutacija. Nakon obrade svake operacije, ažuriramo pozicije u referentnom genomu (refPos) i očitanju (readPos).

```
// Većinsko glasanje za svaku poziciju
for (auto& std::pair<con...onProposal> & proposal : mutationProposals) {
    int pos = proposal.first;
    MutationProposal& votes = proposal.second;

    // Log current mutation proposal
    logFile << "-------------------------------------------------------" << endl;
    logFile << "Position: " << pos << ", Mutation Proposal: {substitutionVotes: " << votes.substitutionVotes
            << ", insertionVotes: " << votes.insertionVotes << ", deletionVotes: " << votes.deletionVotes
            << ", noneVotes: " << votes.noneVotes << "}" << endl;

    string type;
    char base = ' ';
    // if none votes is max
    if (votes.noneVotes >= votes.substitutionVotes - 1 && votes.noneVotes >= votes.insertionVotes - 1 && votes.noneVotes >= votes.deletionVotes -
    1) {

        logFile << "None votes won the voting, no mutation detected." << endl;

        continue;

    } else if (votes.substitutionVotes >= votes.insertionVotes && votes.substitutionVotes >= votes.deletionVotes && votes.substitutionVotes >=
    votes.noneVotes) {
        logFile << "Substitution won the voting, possible bases: ";
        for (char base : votes.substitutionBases) {
            logFile << base << " ";
        }
        logFile << endl;

        if (!votes.substitutionBases.empty()) {
            map<char, int> substitutionBaseCounts;
            for (char base : votes.substitutionBases) {
                substitutionBaseCounts[base]++;
            }
            base = max_element(
                First: substitutionBaseCounts.begin(),
                Last: substitutionBaseCounts.end(),
                Pred: [](const pair<char, int>& a, const pair<char, int>& b) {
                    return a.second < b.second;
                }
            )->first;
        }
        // inform which base has "won"
        logFile << "Base " << base << " has won the voting." << endl;

        type = "substitution";
```

```
} else if (votes.insertionVotes >= votes.substitutionVotes && votes.insertionVotes >= votes.deletionVotes && votes.insertionVotes >= votes.
noneVotes) {

    logFile << "Insertion won the voting, possible bases: ";
    for (char base : votes.insertionBases) {
        logFile << base << " ";
    }
    logFile << endl;

    if (!votes.insertionBases.empty()) {
        map<char, int> insertionBaseCounts;
        for (char base : votes.insertionBases) {
            insertionBaseCounts[base]++;
        }
        base = max_element(
            First: insertionBaseCounts.begin(),
            Last: insertionBaseCounts.end(),
            Pred: [](const pair<char, int>& a, const pair<char, int>& b) {
                return a.second < b.second;
            }
        )->first;
    }
    type = "insertion";

    logFile << "Base " << base << " has won the voting." << endl;

} else if (votes.deletionVotes >= votes.substitutionVotes && votes.deletionVotes >= votes.insertionVotes && votes.deletionVotes >= votes.
noneVotes) {
    type = "deletion";
    base = '-';
} else {
    continue;
}

logFile << type << "," << (type == "insertion" ? "I" : (type == "deletion" ? "D" : "X")) << "," << pos << "," << base << "\n";

// Ispis rezultata
if (base != ' ') {
    outfile  << (type == "insertion" ? "I" : (type == "deletion" ? "D" : "X")) << "," << pos << "," << base << "," << reference[pos] << "\n";

}
```

Ovaj dio funkcije provodi većinsko glasanje na temelju prijedloga mutacija za svaku poziciju u referentnom genomu.

Postavljaju se lokalne varijable type i base koje će pohraniti tip mutacije i bazu koja je najčešće predložena za zamjenu ili umetanje.

Zatim se provjerava koji prijedlog ima najviše glasova.

Ako su noneVotes dominantni, to znači da nema mutacije na toj poziciji i preskače se daljnja obrada za tu poziciju.

Ako su substitutionVotes dominantni, određuje se baza koja je najčešće predložena za zamjenu.

Ako su insertionVotes dominantni, određuje se baza koja je najčešće predložena za umetanje.

Ako su deletionVotes dominantni, bilježi se brisanje.

Nakon što se utvrdi najdominantniji tip mutacije, zapisuje se u log datoteku i CSV datoteku.

2.3. Alternativni pristup algoritmu

U jednom smo trenutku pokušali razviti alternativni algoritam koji bi, nakon svake odluke o kojoj se operaciji radi, izvršio izmjene na očitanjima koja su davala drukčiji rezultat. Ideja je bila da na taj način bolje poravnamo ta očitanja za buduće pozicije. To smo radili tako da smo umetali i uklanjali baze iz očitanja kada se ono nije slagalo s odabranom operacijom. Vjerovali smo da, budući da više ne gledamo tu poziciju, sama unesena baza nije bitna, a sve je pomaknuto.

Takav pristup, iako esencijalno sadrži slične funkcionalnosti kao gore opisani algoritam, nije lako dobiti iz gornjeg algoritma budući da se u ovom slučaju treba iterirati kroz pozicije, a ne kroz očitanja. Isto tako, nakon analize svih očitanja na pojedinoj poziciji treba provesti glasovanje i eventualno uređivanje očitanja s „netočnom“ operacijom na toj poziciji, što nije moguće u gornjoj implementaciji koja prvo zapisuje glasovanje na svakoj poziciji, a tek ga onda provodi.

Alternativni pristup smo pokušali implementirati kako bi vidjeli hoćemo li dobiti bolje rezultate. Međutim, ovaj pristup je bio vrlo osjetljiv na greške koje se događaju zbog pojedinačnih netočnosti ("one off" greške). Ove greške su stalno narušavale poravnanje, uzrokujući lančane reakcije netočnih izmjena koje su se protezale kroz cijelu sekvencu. To je rezultiralo time da su mnoge pozicije bile pogrešno poravnate, što je značajno smanjivalo točnost detekcije mutacija.

Nakon konzultacija s mentorom, zaključili smo da je ovaj pristup prekompliciran i previše osjetljiv na pojedinačne greške. Stoga smo odustali od njega i posvetili se gore opisanom pristupu koji koristi većinsko glasanje. Ovaj pristup se pokazao robusnijim i jednostavnijim, te daje uvjerljive rezultate bez potrebe za dodatnim izmjenama očitanja.

3. EVALUACIJA

Evaluacija našeg algoritma za detekciju mutacija u genomu temelji se prvenstveno na usporedbi s referentnom CSV datotekom koju smo dobili s podacima. Osjetljivost algoritma (koliko je "glasova" dovoljno za odluku o mutaciji) uspoređivana je s alatom FreeBayes.

3.1. Freebayes

FreeBayes je napredni alat za detekciju mutacija koji koristi složene statističke metode. Međutim, primijetili smo značajne razlike između rezultata FreeBayesa i referentne CSV datoteke. Također, FreeBayes se s defaultnim postavkama izuzetno dugo izvršavao na malom testnom primjeru, trajući satima. Stoga smo koristili preporučene postavke osjetljivosti s interneta, što je dovelo do detekcije oko 5000 mutacija na lambda očitanju.
Naš algoritam je otkrio otprilike jednak broj mutacija kao FreeBayes, ali nismo mogli izravno usporediti rezultate zbog različitih formata izlaza. FreeBayes-ov izlaz nije konstantan u zapisima izmjena, dok je referentna CSV datoteka pisana u istom formatu kao i naš izlaz, što je značajno olakšalo evaluaciju.

3.2. Skripta za usporedbu s referentnom CSV datotekom

Za evaluaciju smo napisali skriptu u Pythonu koja uspoređuje naše rezultate s referentnim rezultatima. Skripta provjerava poklapanje pozicija, operacija i baza te dodjeljuje bodove za svako poklapanje. Na kraju, izračunava ukupni postotak točnosti.

```python
import csv

def read_results(filename):
    results = []
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        next(reader)  # Skip header
        for row in reader:
            results.append((row[0], int(row[1]), row[2]))
    return results

def evaluate_results(my_results, ref_results, log_filename):
    total_points = 0
    max_points = 0

    x_points = 1
    y_points = 1
    z_points = 1

    with open(log_filename, 'w') as log_file:
        log_file.write("Evaluation Log\n")
        log_file.write("====================\n")

        my_dict = {pos: (type_, base) for type_, pos, base in my_results}

        for ref_type, ref_pos, ref_base in ref_results:
            max_points += x_points + y_points + z_points
            log_file.write(f"Checking reference result: {ref_type},{ref_pos},{ref_base}\n")
            if ref_pos in my_dict:
                my_type, my_base = my_dict[ref_pos]
                log_file.write(f"  Found matching position in my results: {my_type},{ref_pos},{my_base}\n")
                total_points += x_points
                if my_type == ref_type:
                    total_points += y_points
                    if my_base == ref_base:
                        total_points += z_points
                        log_file.write(f"    All match: +{x_points + y_points + z_points} points\n")
                    else:
                        log_file.write(f"    Type match, but base mismatch: +{x_points + y_points} points\n")
                else:
                    log_file.write(f"    Position match, but type mismatch: +{x_points} points\n")
            else:
                log_file.write(f"  No matching position in my results\n")

        log_file.write("====================\n")
        accuracy = (total_points / max_points) * 100
        log_file.write(f"Total points: {total_points}\n")
        log_file.write(f"Max points: {max_points}\n")
        log_file.write(f"Accuracy: {accuracy:.2f}%\n")

    print(f"Accuracy: {accuracy:.2f}%")

# Replace with the actual file paths
my_results_file = "data/mutations.csv"
ref_results_file = "data/lambda_mutated.csv"
log_file = "evaluation_log.txt"

my_results = read_results(my_results_file)
ref_results = read_results(ref_results_file)
evaluate_results(my_results, ref_results, log_file)
```

Na kraju evaluacije, naš algoritam je postigao preciznost od točno **83%**, s čime smo iznimno zadovoljni. Preciznost varira ovisno o tome koliko bodova dodjeljujemo za točnu poziciju, operaciju i bazu. Postotak točnosti može varirati između 82% i 85%, ali smatramo da je najpoštenije dodjeljivati 1 bod za točnu poziciju, 2 boda za točnu poziciju i operaciju te 3 boda za točnu poziciju, operaciju i bazu.

4. ZAKLJUČAK

Projekt detekcije mutacija uspješno je implementiran korištenjem Minimap2 za poravnanje očitanja i našeg vlastitog algoritma za detekciju mutacija.

Algoritam u C++ radi sljedeće:

- Parsira SAM datoteku i učitava referentni genom.
- Obrađuje CIGAR stringove kako bi identificirao mutacije.
- Za svaku poziciju, većinskim glasanjem određuje vrstu i bazu mutacije.
- Rezultate zapisuje u CSV datoteku.

Evaluacija rezultata pokazala je preciznost od 83%, što potvrđuje učinkovitost našeg pristupa, pogotovo ako se uzme u obzir da se radi o pristupu koji je nekoliko redova veličina brži od vanjski funkcija kao što je npr. Freebayes.