SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Pronalazak mutacija pomoću treće generacije sekvenciranja

Laura Barišić i Mia Nazor

Mentor: Doc. dr. sc. Krešimir Križanović

Zagreb, lipanj 2025.

Sadržaj

1. Uvod

Genomika je grana genetike koja primjenjuje metode DNA sekvenciranja i bioinformatike u cilju sekvenciranja, sastavljanja i analize funkcija i strukture genoma.

Sekvenciranje DNA omogućuje brzo i točno očitavanje nukleotidnih baza (A, T, C, G) u biološkom materijalu. Moderni sekvenceri proizvode milijune očitanih sekvenci koje je potrebno poravnati na referentni genom kako bi se identificirale varijacije između uzorka i referentne sekvence.

Poravnanja sekvenci omogućuju prepoznavanje:

- Supstitucija (substitutions) – gdje se očitana baza razlikuje od baze u referenci.
- Umetanja (insertions) – dodatne baze prisutne u očitanju koje nisu u referenci.
- Brisanja (deletions) – baze koje nedostaju u očitanju, a prisutne su u referenci.

Analiza ovih varijacija ključna je za:

- Detekciju mutacija povezanih s bolestima.
- Identifikaciju genetskih markera i varijacija u populacijama.
- Razumijevanje evolucijskih promjena u genomima.

U ovom projektu analiziramo poravnanja očitanja (SAM datoteka) na referentni genom (FASTA datoteka) s ciljem identifikacije mutacija. U SAM datoteci nalazi se puno očitanja na različitim pozicijama referentnog genoma. Cilj je za svaku poziciju referentnog genoma izračunati broj očitanja bez mutacija, broj supstitucija, umetanja i brisanja, te najčešće očitanu bazu u slučaju supstitucija ili umetanja na temelju većinskog glasanja.

Rezultati ove analize omogućuju detaljan uvid u kvalitetu poravnanja i distribuciju mutacija, što je ključno za primjene u genomici, medicini i biotehnologiji.

2. Opis algoritma

Glavni dio algoritma koji se bavi traženjem mutacija na pozicijama referentnog genoma nalazi se u *bioinf.cpp* datoteci. U datoteci *converter.cpp* nalazi se kod koji na „ljepši“ i prikladniji način formatira podatke dobivene korištenjem FreeBayes alata kako bismo lakše usporedili svoje rezultate s referentnima. Na početku svake .cpp datoteke nalazi se uključivanje svih biblioteka potrebnih za rad.

2.1 bioinf.cpp

2.1.1 Struktura SamRecord

```
// Structure for storing SAM record information
struct SamRecord {
    string qname;
    int flag;
    string rname;
    int64_t pos;
    string cigar;
    string seq;
};
```

U prikazanoj strukturi SamRecord nalaze se podaci svakog očitanja koji se dobivaju iz SAM datoteke. Jako bitan član strukture je *flag* po kojem gledamo koja očitanja odmah zanemarujemo, zatim *pos* koji nam govori na kojoj poziciji u referentnom genomu se („1-based“) očitanje nalazi, *cigar* koji čuva CIGAR zapis te *seq* koji čuva sekvencu očitanja koji se uspoređuje s referentnim genomom.

2.1.2 Struktura PosVotes

```
// Structure for storing votes at a position in the reference genome
struct PosVotes {
    int none = 0;
    int deleted = 0;
    int inserted = 0;
    int substituted = 0;
    vector<char> substitutionBases;
    vector<char> insertionBases;
};
```

Struktura PosVotes sadrži članove kojima se broji koliko je podudaranja (*none*), broj obrisanih baza (*deleted*), broj umetnutih baza (*inserted*), broj zamjena (*substituted*) te pripadajući vektor baza koje su zamijenjene i vektor umetnutih baza. Ova struktura je bitna kako bi se kasnije u funkciji *voting* moglo većinskim glasanjem odrediti koja je najvjerojatnija mutacija na toj poziciji referentnog genoma.

2.1.3 Funkcija reversee

```cpp
// Function for reverse complementing a sequence
string reversee(const string &seq) {
    string reversed_seq(seq.length(), ' ');
    size_t index = 0;
    for (int i = (int)seq.length() - 1; i >= 0; i--) {
        char base = seq[i];
        char new_base;
        switch (base) {
        case 'A':
            new_base = 'T';
            break;
        case 'T':
            new_base = 'A';
            break;
        case 'G':
            new_base = 'C';
            break;
        case 'C':
            new_base = 'G';
            break;
        default:
            new_base = base;
        }
        reversed_seq[index++] = new_base;
    }
    return reversed_seq;
}
```

Funkcija *reversee* koristi se u slučajevima kada očitanje dolazi s komplementarnog lanca DNA ili kada je poravnanje očitanja u suprotnom smjeru u odnosu na referentnu sekvencu. Ako je član strukture SamRecord *flag* jednak 16 za neko očitanje potrebno je reverzno komplementirati očitanje, tj. pozvati za njega navedenu funkciju.

2.1.4 Funkcije read_fasta i read_sam

```cpp
// Function to read a FASTA file and return the sequence as a string
string read_fasta(const string &filename) {
    ifstream file(filename);
    if (!file) {
        cout << "Greška pri otvaranju FASTA datoteke." << endl;
        return "";
    }

    string line, sequence;
    while (getline(file, line)) {
        if (!line.empty() && line[0] != '>') {
            sequence += line;
        }
    }

    return sequence;
}
```

```cpp
// Function to read a SAM file and return a vector of SamRecord structures
vector<SamRecord> read_sam(const string &filename) {
    ifstream file(filename);
    if (!file) {
        cout << "Greška pri otvaranju SAM datoteke." << endl;
        return {};
    }

    vector<SamRecord> records;
    string line;

    while (getline(file, line)) {
        if (!line.empty() && line[0] == '@')
            continue;

        SamRecord record;
        if (parse_sam_line(line, record)) {
            records.push_back(record);
        }
    }

    return records;
}
```

Funkcija *read_fasta* vraća zapis referentnog genoma. Preskače se prvi red koji započinje znakom „>“. Funkcija *read_sam* iz SAM datoteke uzima očitanja i sprema ih u vektor SamRecorda.

2.1.5 Funkcija parse_sam_line

```cpp
// Function to parse a SAM line and fill the SamRecord structure
// Returns true if the line is successfully parsed, otherwise false
bool parse_sam_line(const string &line, SamRecord &record) {
    istringstream iss(line);
    vector<string> fields;
    string field;

    while (getline(iss, field, '\t')) {
        fields.push_back(field);
    }

    if (fields.size() < 11 || stoi(fields[1]) & 4) {
        return false;
    }

    record.qname = fields[0];
    record.flag = stoi(fields[1]);
    record.rname = fields[2];
    record.pos = stoll(fields[3]);
    record.cigar = fields[5];
    record.seq = fields[9];

    // If this is commented we get better results
    // but it is not correct according to the SAM specification
    if (record.flag & 16)
        record.seq = reversee(record.seq);

    return true;
}
```

Za svako očitanje u *ime_datoteke.sam* potrebno je pozvati navedenu funkciju kako bi se dobili svi potrebni parametri za stvoriti zapis oblika SamRecord. Funkcija vraća bool vrijednost kako bismo preskočili uzimanje „neispravnih“ očitanja (ona koja imaju manje od 11 zapisa ili ona koja nisu poravnata *flag* = 4) u funkciji *read*_sam. Također, komentirane su linije za provjeru reverznog komplementa. To je zato što smo uzimajući u obzir reverzno komplementirana očitanja u primjeru s *lambda.sam* i *lambda.fasta* datotekama dobivale oko 5000 mutacija, dok smo komentiranjem tih linija dobivale oko 160 mutacija što je puno sličnije broju mutacija koje se dobiju s rezultatima dobivenim u *lambda_mutated.csv* datoteci.

2.1.6 Funkcija voting

```cpp
// Function to perform voting on the mutations
// at each position in the reference genome
void voting(unordered_map<int64_t, PosVotes> &dict,
            unordered_map<int64_t, pair<string, string>> &final_dict) {
    string max_votes;
    string max_base;
    for (const auto &[pos, votes] : dict) {
        max_votes = "none";
        max_base = "-";

        int total_votes =
            votes.none + votes.deleted + votes.inserted + votes.substituted;

        if (votes.none >= votes.substituted && votes.none >= votes.inserted &&
            votes.none >= votes.deleted && votes.none > 2 &&
            votes.none >= ceil(0.4 * total_votes)) {
            continue;
        } else if (votes.substituted >= votes.inserted &&
                   votes.substituted >= votes.deleted &&
                   votes.substituted >= votes.none && votes.substituted > 2 &&
                   votes.substituted >= ceil(0.4 * total_votes)) {
            max_votes = "X";
            unordered_map<char, int> freq;
            char max_elem = '\0';
            int max_count = 0;
            for (char c : votes.substitutionBases) {
                int count = ++freq[c];
                if (count > max_count) {
                    max_count = count;
                    max_elem = c;
                }
            }
            max_base = string(1, max_elem);
```

```
        } else if (votes.inserted >= votes.substituted &&
                   votes.inserted >= votes.deleted &&
                   votes.inserted >= votes.none && votes.inserted > 2 &&
                   votes.inserted >= ceil(0.4 * total_votes)) {
            max_votes = "I";
            unordered_map<char, int> freq;
            char max_elem = '\0';
            int max_count = 0;
            for (char c : votes.insertionBases) {
                int count = ++freq[c];
                if (count > max_count) {
                    max_count = count;
                    max_elem = c;
                }
            }
            max_base = string(1, max_elem);
        } else if (votes.deleted >= votes.substituted &&
                   votes.deleted >= votes.inserted &&
                   votes.deleted >= votes.none && votes.deleted > 2 &&
                   votes.deleted >= ceil(0.4 * total_votes)) {
            max_votes = "D";
            max_base = "-";
        }

        final_dict[pos] = {max_votes, max_base};
    }
}
```

Jedan od parametara koje funkcija *voting* prima je *dict* koja za svaku poziciju referentnog genoma ima spremljenu SamRecord strukturu. Na temelju članova unutar strukture provjeravamo na svakoj poziciji koje mutacije je bilo najviše, također uzimamo u obzir ako je bilo više od 2 glasa za tu mutaciju te mora vrijediti pravilo većinskog glasanja. Za pravilo većinskog glasanja u početku smo uzele kako i po definiciji jest da je mutacija koja ima više od 50% ukupnih glasova najvjerojatnija, no promatrajući promjenu točnosti zaključile smo da je bolje ako stavimo da je veće od 40% ukupnih glasova. Mutacija koja je zadovoljila navedene uvjete postaje najvjerojatnija mutacija na toj poziciji i ulazi u vektor *final_dict*. Kao opcija mutacije u *final_dict* uzima se i podudaranje koje će biti izbačeno na kraju u vektoru *sorted_mutations*. Navedene parametre dobile smo podešavanjem da dobijemo rezultate što sličnije referentnima.

2.1.7 Funkcija mutations

```
// Function to process and identify mutations
void mutations(const vector<SamRecord> &sam_records,
               unordered_map<int64_t, PosVotes> &dict,
               const string &fasta_sequence,
               unordered_map<int64_t, pair<string, string>> &final_dict) {
    ofstream matchingFile(DATA_DIR + "matching.txt");
    if (!matchingFile) {
        cerr << "Greška pri otvaranju datoteke matching.txt" << endl;
        return;
    }
    for (const SamRecord &record : sam_records) {

        int64_t refPos = record.pos - 1;
        int64_t readPos = 0;
        int i = 0;

        while (i < record.cigar.length()) {
            int64_t length = 0;
            while (i < record.cigar.length() && isdigit(record.cigar[i])) {
                length *= 10;
                length += record.cigar[i] - '0';
                i++;
            }

            if (i >= record.cigar.length())
                break;
            char op = record.cigar[i];
            i++;

            // Check mutation operation and process accordingly
            if (op == 'M') {
                for (int j = 0; j < length; ++j) {
                    if (refPos >= fasta_sequence.size() ||
                        readPos >= record.seq.size())
                        break;

                    char refBase = fasta_sequence[refPos];
                    char readBase = record.seq[readPos];
```

```
            // Check mutation operation and process accordingly
            if (op == 'M') {
                for (int j = 0; j < length; ++j) {
                    if (refPos >= fasta_sequence.size() ||
                        readPos >= record.seq.size())
                        break;

                    char refBase = fasta_sequence[refPos];
                    char readBase = record.seq[readPos];

                    if (refBase == readBase) {
                        matchingFile << "refpos " << refPos << " - REF_BASE "
                                     << refBase << " | " << "readpos "
                                     << readPos + 1 << " - READ_BASE "
                                     << readBase << " [MATCH]" << endl;
                        dict[refPos].none++;
                    } else {
                        matchingFile << "refpos " << refPos << " - REF_BASE "
                                     << refBase << " | " << "readpos "
                                     << readPos + 1 << " - READ_BASE "
                                     << readBase << " [MISS]" << endl;
                        dict[refPos].substituted++;
                        dict[refPos].substitutionBases.push_back(readBase);
                    }
                    refPos++;
                    readPos++;
                }
            } else if (op == 'I') {
                for (int j = 0; j < length; ++j) {
                    if (readPos + j >= record.seq.size())
                        break;

                    char refBase = fasta_sequence[refPos];
                    char readBase = record.seq[readPos + j];

                    matchingFile << "refpos " << refPos << " - REF_BASE "
                                 << refBase << " | " << "readpos "
                                 << (readPos + j + 1) << " - READ_BASE "
                                 << readBase << " [INSERT]" << endl;

                    dict[refPos].insertionBases.push_back(readBase);
                    dict[refPos].inserted++;
                }
                readPos += length;
            } else if (op == 'D') {
                for (int j = 0; j < length; ++j) {
                    if (refPos >= fasta_sequence.size() ||
                        readPos >= record.seq.size())
                        break;

                    char refBase = fasta_sequence[refPos];
                    char readBase = record.seq[readPos];

                    matchingFile << "refpos " << refPos << " - REF_BASE "
                                 << refBase << " | " << "readpos "
                                 << readPos + 1 << " - READ_BASE " << readBase
                                 << " [DELETE]" << endl;
                    dict[refPos].deleted++;
                    refPos++;
                }
            } else if (op == 'S') {
                readPos += length;
            }
        }
    }
```

```
    matchingFile.close();

    ofstream votingFile(DATA_DIR + "voting.txt");
    // Writing votes to the file
    votingFile << "\n--- Glasovi po pozicijama u referentnom genomu ---\n";
    for (const auto &[pos, votes] : dict) {
        votingFile << "Pozicija: " << pos << "\n";
        votingFile << "  - Podudaranja (none): " << votes.none << "\n";
        votingFile << "  - Supstitucije: " << votes.substituted << " ";
        if (!votes.substitutionBases.empty()) {
            votingFile << "[";
            for (char base : votes.substitutionBases)
                votingFile << base << ",";
            votingFile << "]";
        }
        votingFile << "\n";
        votingFile << "  - Brisanja (deleted): " << votes.deleted << "\n";
        votingFile << "  - Umetanja (inserted): " << votes.inserted << " ";
        if (!votes.insertionBases.empty()) {
            votingFile << "[";
            for (char base : votes.insertionBases)
                votingFile << base << ",";
            votingFile << "]";
        }
        votingFile << "\n\n";
    }
    // Closing the voting file
    votingFile.close();

    voting(dict, final_dict);
}
```

Funkcija *mutations* prolazi kroz svako očitanje (*read*) unutar vektora strukture SamRecords. U varijablu *refPos* sprema se pozicija u referentnom genomu s kojom je poravnato očitanje („0-based“), dok se u *readPos* sprema pozicija baze u očitanju. Na temelju *cigar* zapisa dobiva se broj mutacija i sama mutacija koju je potrebno provjeriti i zapisati za određenu poziciju referentnog genoma. To se odvija dok se ne dođe do kraja *cigar* zapisa. Bitno je bilo paziti po kojem se nizu pomičemo ovisno koja je mutacija. Kada se radi o M (podudaranje), pomiču se pozicije referentnog genoma i očitanja, za I (umetanje) pomiče se samo pozicija u očitanju, za D (brisanje) pomiče se samo pozicija u referentnom genomu, a za S (ignoriranje) pomiče samo pozicija u očitanju jer se te baze preskaču. Umetnute i zamijenjene baze uvijek se dodaju na poziciju referentnog genoma na kojoj se trenutno uspoređuje. Potrebno je pripaziti na slučaj da je jedan zapis kraći od drugoga. Svaka provjera i zapisuje se u posebnu datoteku *matching.txt* kako bismo lakše kasnije usporedile slaže li se algoritam s rezultatima dobivenim drugim alatima. Također, na kraju prolaska kroz sva očitanja spremile smo u datoteku *voting.txt* prikaz koliko je kojih mutacija bilo, koje su to baze umetnute ili zamijenjene i na kojim pozicijama se nalaze te mutacije. Konačno s takvim vektorom *dict* koji sadrži sve parametre spremne za većinsko glasanje moguće je pozvati funkciju *voting*.

2.1.8 Funkcija main

```cpp
int main() {
    auto start_time = std::chrono::high_resolution_clock::now();

    string fasta_path = DATA_DIR + "lambda.fasta";
    string sam_path = DATA_DIR + "lambda.sam";

    string fasta_sequence = read_fasta(fasta_path);
    cout << "FASTA duljina: " << fasta_sequence.size() << " znakova\n";

    vector<SamRecord> sam_records = read_sam(sam_path);
    cout << "\nUkupno očitanih SAM zapisa: " << sam_records.size() << "\n";

    for (const auto &r : sam_records) {
        cout << "QNAME: " << r.qname << " | FLAG: " << r.flag
             << " | RNAME: " << r.rname << " | POS: " << r.pos
             << " | CIGAR: " << r.cigar << " | SEQ: " << r.seq << "\n";
    }

    unordered_map<int64_t, PosVotes> dict;
    unordered_map<int64_t, pair<string, string>> final_dict;
    mutations(sam_records, dict, fasta_sequence, final_dict);

    // Writing to CSV file
    ofstream outfile(DATA_DIR + "mutations.csv");
    if (!outfile) {
        cerr << "Greška pri otvaranju datoteke za pisanje mutacija." << endl;
        return 1;
    }

    vector<pair<int64_t, pair<string, string>>> sorted_mutations(
        final_dict.begin(), final_dict.end());
    sort(sorted_mutations.begin(), sorted_mutations.end());

    for (const auto &[pos, result] : sorted_mutations) {
        if (result.first != "none") {
            outfile << result.first << "," << pos << "," << result.second
                    << "\n";
        }
    }
    outfile.close();

    auto end_time = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed_seconds = end_time - start_time;

    cout << "Vrijeme izvođenja: " << elapsed_seconds.count() << " sekundi\n";
    return 0;
}
```

Funkcija *main* napisana je tako da korisniku omogući što jednostavnije korištenje sa što manje zamaranja što je u pozadini. Ostale funkcije nazvane su smisleno stoga je pri njihovom pozivu odmah jasno čemu služe. U *fasta_sequence* upisuje se referentni genom, vektor u *sam_records* upisuju se očitanja i svi parametri iz SAM datoteke. Inicijaliziraju se mape s kojima će se pozivati funkcija *mutations*. U *sorted_mutations* prvo se sortiraju mutacije po pozicijama u referentnom genomu, a zatim se izbacuju podudaranja koja su do tog trenutka i dalje tretirana kao mutacije. Informacija o mutaciji na svakoj poziciji referentnog genoma zapisuje se u datoteku *mutations.csv*. Na početku i na kraju funkcije očitana su vremena kako bi se moglo odrediti trajanje programa.

2.2 converter.cpp

```cpp
// Determines mutation type based on REF and ALT values from VCF
// Returns:
//   "X" - substitution (same length, single base change)
//   "I" - insertion (ALT longer than REF)
//   "D" - deletion (REF longer than ALT)
string tip_mutacije(const string& ref, const string& alt_raw) {
    string alt = alt_raw;

    // If multiple ALT values (comma-separated), only consider the first
    size_t comma_pos = alt.find(',');
    if (comma_pos != string::npos) {
        alt = alt.substr(0, comma_pos);
    }

    // Determine mutation type by comparing lengths of REF and ALT
    if (ref.length() == 1 && alt.length() == 1) {
        return "X";
    } else if (ref.length() < alt.length()) {
        return "I";
    } else if (ref.length() > alt.length()) {
        return "D";
    } else {
        return "X";
    }
}

int main() {
    // Open input VCF file and output CSV file
    ifstream vcf(DATA_DIR + "freebayes.vcf");
    ofstream csv_out(DATA_DIR + "freebayes_mutations.csv");

    if (!vcf.is_open() || !csv_out.is_open()) {
        cerr << "Error opening files!" << endl;
        return 1;
    }

    // Write CSV header
    csv_out << "Position,Type,REF,ALT\n";

    string line;
    while (getline(vcf, line)) {
        // Skip empty lines and VCF metadata (lines starting with '#')
        if (line.empty() || line[0] == '#') continue;

        stringstream ss(line);
        string column;
        string columns[5];
        int i = 0;

        // Extract the first 5 tab-separated columns from the VCF line
        while (getline(ss, column, '\t') && i < 5) {
            columns[i++] = column;
        }

        if (i < 5) continue;

        string position = columns[1];
        string ref = columns[3];
        string alt = columns[4];
        string mut_type = tip_mutacije(ref, alt);

        // Write mutation data to CSV
        csv_out << position << "," << mut_type << "," << ref << "," << alt << "\n";
    }

    // Close files
    vcf.close();
    csv_out.close();

    return 0;
}
```

Program učitava VCF datoteku generiranu alatkom FreeBayes, parsira informacije o mutacijama te ih sprema u CSV format. Iz VCF-a se izvlače pozicija, tip mutacije (supstitucija, umetanja, brisanja) i sekvence REF i ALT. Ako ALT sadrži više vrijednosti, koristi se samo prva. Izlazni CSV olakšava daljnju analizu mutacija.

2.3 accuracy.cpp

```cpp
// Represents a single mutation
struct Mutation {
    string type;   // "X", "I", "D"
    int position;
    string newValue;
};

// Aliases for better readability
using MutationList = vector<Mutation>;
using MutationMap = unordered_map<int, pair<string, string>>; // position -> (type, newValue)


// Parse a CSV line into a Mutation object
Mutation parseLine(const string& line) {
    stringstream ss(line);
    string type, posStr, val;
    getline(ss, type, ',');
    getline(ss, posStr, ',');
    getline(ss, val, ',');
    return {type, stoi(posStr), val};
}

// Load mutations from a CSV file (ignores header)
MutationList loadFromFile(const string& filename) {
    MutationList list;
    ifstream file(filename);
    if (!file) {
        cerr << "Error: cannot open file " << filename << endl;
        return list;
    }

    string line;
    getline(file, line); // Skip header
    while (getline(file, line)) {
        list.push_back(parseLine(line));
    }

    return list;
}

// Create a hash map for fast mutation lookup by position
MutationMap indexByPosition(const MutationList& list) {
    MutationMap map;
    for (const auto& m : list) {
        map[m.position] = {m.type, m.newValue};
    }
    return map;
}

// Compare predicted mutations against reference mutations
double evaluate(const MutationList& predicted, const MutationList& reference) {

    const int scorePos = 1;
    const int scoreType = 1;
    const int scoreValue = 1;

    int total = 0;
    int earned = 0;

    MutationMap predictedMap = indexByPosition(predicted);

    for (const auto& ref : reference) {
        total += scorePos + scoreType + scoreValue;

        auto it = predictedMap.find(ref.position);
        if (it != predictedMap.end()) {
            const auto& [pType, pVal] = it->second;
            earned += scorePos;

            if (pType == ref.type) {
                earned += scoreType;
                if (pVal == ref.newValue) {
                    earned += scoreValue;
                }
            }
        }
    }

    double accuracy = (total > 0) ? (earned * 100.0 / total) : 0.0;

    return accuracy;
}
```

```cpp
int main() {
    const string predFile = DATA_PATH + "mutations.csv";
    const string refFile = DATA_PATH + "lambda_mutated.csv";

    MutationList predicted = loadFromFile(predFile);
    MutationList reference = loadFromFile(refFile);

    double accuracy = evaluate(predicted, reference);

    cout << fixed << "Mutation detection accuracy: " << accuracy << "%" << endl;
    return 0;
}
```

Ovaj program u C++-u služi za evaluaciju točnosti predikcije genetskih mutacija. Učitava dvije CSV datoteke: jednu s predikcijama mutacija (*mutations.csv*), a drugu s referentnim (točnim) mutacijama (*lambda_mutated.csv*). Svaka mutacija definirana je tipom (X - supstitucija, I - umetanje, D - brisanje), pozicijom u genomu i novom vrijednošću baze. Program uspoređuje predikcije s referencom prema tri kriterija: pozicija, tip i vrijednost. Na temelju toga dodjeljuje bodove i računa ukupnu točnost u postocima. 1 bod daje za svaku točnu poziciju, tip mutacije i bazu mutacije. Rezultati evaluacije ispisuju se na standardni izlaz.

3. Pristup programu

3.1 Tehnička podrška

3.1.1 Alat minimap2

Minimap2 je brz i učinkovit program koji služi za pronalaženje sličnosti (preklapanja) između dugačkih bioloških sekvenci s velikim brojem grešaka, kao i za mapiranje dugačkih očitanja ili njihovih sklopova na poznati referentni genom. Po potrebi, može napraviti detaljno poravnanje koje pokazuje točno kako se očitanje poklapa s referencom (to se zove CIGAR zapis).

Ovaj alat je posebno prilagođen za rad sa sekvencama duljine od nekoliko tisuća do stotinu milijuna baza, uz stopu pogrešaka od oko 15%. Rezultate izbacuje u dva formata:

PAF (*Pairwise mApping Format*) – jednostavan format koji sadrži osnovne informacije o poravnanju između dviju sekvenci.

SAM (Sequence Alignment/Map format) – detaljniji format koji uključuje informacije o položaju očitanja na referentnom genomu, kvaliteti poravnanja, i CIGAR zapis koji opisuje kako su sekvence poravnate. CIGAR zapis je oblika niza brojeva koji označavaju koliko puta se određena mutacija dogodila i slova koja označavaju o kojoj mutaciji je riječ. Ovaj format smo koristile u našem algoritmu.

Primjer CIGAR zapisa: 7M2I5D3S

7M – označava sedam podudaranja ili supstitucija na tim pozicijama

2I – označava dva umetanja u očitanju, a te baze nedostaju u referentnom genomu

5D – označava pet brisanja u očitanju, a postoji neka baza u referentnom genomu

3S – označava da su tri baze očitanja odrezane (npr. ne poravnavaju se na referencu), ali su i dalje uključene u očitanje

3.1.2 Alat FreeBayes

FreeBayes je softver koji se koristi za pronalaženje genetskih varijacija, odnosno razlika između DNA sekvence nekog organizma i referentne sekvence. Funkcionira tako da koristi poravnanja sekvenci očitanja (najčešće u BAM ili CRAM formatima), koja su prethodno mapirana na referentni genom, i zatim pronalazi mjesta gdje se sekvence razlikuju od reference.

FreeBayes koristi Bayesov statistički model za detekciju mutacija, što mu omogućuje da procijeni vjerojatnost postojanja određene mutacije na osnovu podataka iz očitanja. Ovo je

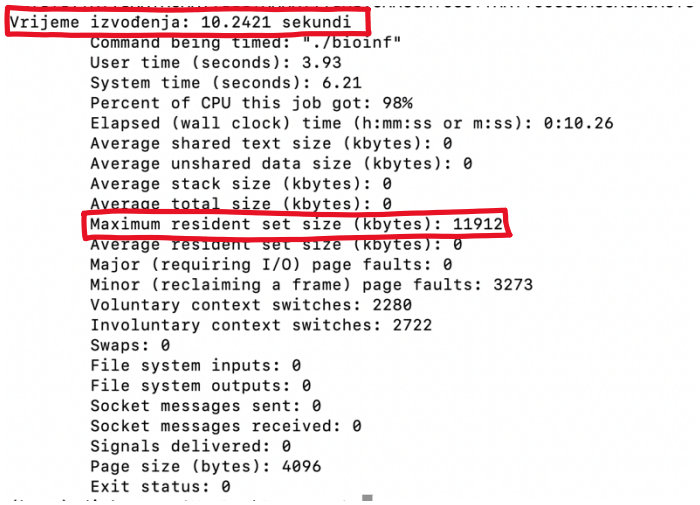
važno jer podaci sekvenciranja mogu sadržavati pogreške, pa je potrebno pažljivo razlikovati prave mutacije od artefakata.

Jedna od prednosti FreeBayesa je da može analizirati više uzoraka istovremeno, što je korisno u populacijskim studijama ili prilikom analize uzoraka iz različitih izvora. Rezultati koje FreeBayes daje zapisani su u VCF formatu (*Variant Call Format*), koji sadrži detaljne informacije o pronađenim varijacijama, njihovim pozicijama u genomu, vrstama mutacija, dubinama pokrivenosti i drugim relevantnim podacima.

FreeBayes je tako dizajniran da bude fleksibilan, može se koristiti za razne organizme i eksperimentalne postavke, a njegova snaga leži u sposobnosti da generira visokokvalitetne rezultate koristeći napredne statističke pristupe, dok istovremeno omogućava jednostavno izvođenje analize čak i na velikim skupovima podataka. Potrebno je pripremiti datoteke za obradu alatom, obzirom da algoritam u projektu koristi SAM i FASTA datoteke bilo je potrebno pretvoriti SAM datoteku s očitanjima u BAM datoteku.

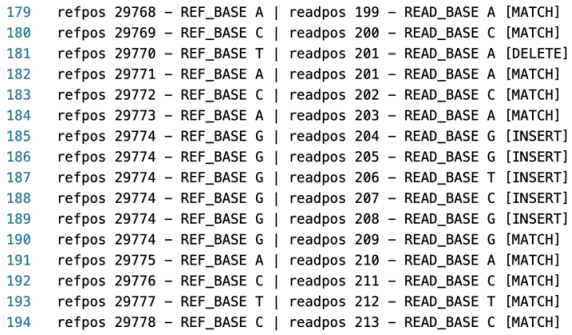
4. Rezultati

Algoritam smo prvo probale na kraćim datotekama (lambda.sam i lambda.fasta) koje smo dobile uz definiciju projekta. Pokretanjem programa ispostavilo se da algoritam radi s točnošću od 84.98% što je dobiveno programskim kodom u *accuracy.cpp* datoteci. Mjerenjem vremena trajanja programa pokazalo se da program traje oko 10 sekunda. Također, mjerile smo i koliko memorije zauzima naš program i pokazalo se da je to oko 11.63 MB.

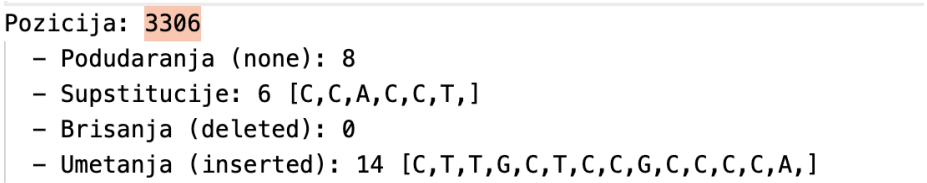


```
Vrijeme izvodenja: 10.2421 sekundi
        Command being timed: "./bioinf"
        User time (seconds): 3.93
        System time (seconds): 6.21
        Percent of CPU this job got: 98%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:10.26
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 11912
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 3273
        Voluntary context switches: 2280
        Involuntary context switches: 2722
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

Ispod su slike isječak datoteke *matching.txt* u kojoj su pokazane koje su se mutacije dogodile na kojim pozicijama referentnog genoma te primjer ispisa za poziciju 3306 iz *voting.txt* datoteke.



```
179   refpos 29768 - REF_BASE A | readpos 199 - READ_BASE A [MATCH]
180   refpos 29769 - REF_BASE C | readpos 200 - READ_BASE C [MATCH]
181   refpos 29770 - REF_BASE T | readpos 201 - READ_BASE A [DELETE]
182   refpos 29771 - REF_BASE A | readpos 201 - READ_BASE A [MATCH]
183   refpos 29772 - REF_BASE C | readpos 202 - READ_BASE C [MATCH]
184   refpos 29773 - REF_BASE A | readpos 203 - READ_BASE A [MATCH]
185   refpos 29774 - REF_BASE G | readpos 204 - READ_BASE G [INSERT]
186   refpos 29774 - REF_BASE G | readpos 205 - READ_BASE G [INSERT]
187   refpos 29774 - REF_BASE G | readpos 206 - READ_BASE T [INSERT]
188   refpos 29774 - REF_BASE G | readpos 207 - READ_BASE C [INSERT]
189   refpos 29774 - REF_BASE G | readpos 208 - READ_BASE G [INSERT]
190   refpos 29774 - REF_BASE G | readpos 209 - READ_BASE G [MATCH]
191   refpos 29775 - REF_BASE A | readpos 210 - READ_BASE A [MATCH]
192   refpos 29776 - REF_BASE C | readpos 211 - READ_BASE C [MATCH]
193   refpos 29777 - REF_BASE T | readpos 212 - READ_BASE T [MATCH]
194   refpos 29778 - REF_BASE C | readpos 213 - READ_BASE C [MATCH]
```



```
Pozicija: 3306
  - Podudaranja (none): 8
  - Supstitucije: 6 [C,C,A,C,C,T,]
  - Brisanja (deleted): 0
  - Umetanja (inserted): 14 [C,T,T,G,C,T,C,C,G,C,C,C,C,A,]
```

5. Zaključak i sažetak

Projekt „Pronalazak mutacija pomoću treće generacije sekvenciranja“ uspješno je implementiran korištenjem minimap2 i FreeBayes alata, uz vlastiti algoritam za detekciju mutacija razvijen u C++. Algoritam koristi podatke iz SAM i FASTA datoteka, računa broj različitih tipova mutacija (supstitucija, umetanja, brisanja) i određuje najvjerojatnije mutacije na temelju pravila većinskog glasanja. Posebna pažnja posvećena je analizi CIGAR zapisa i optimizaciji parametara većinskog glasanja. U konačnici, algoritam postiže točnost od 84,98%, uz relativno nisku memorijsku potrošnju (11,63 MB) i vrijeme izvršavanja od oko 10 sekundi na testnom skupu podataka (lambda sekvence). Rezultati pokazuju da razvijeni sustav pruža pouzdane i brze rezultate detekcije mutacija, što je primjenjivo u bioinformatici, medicini i biotehnologiji.

6. Literatura

Alat minimap2 - https://github.com/lh3/minimap2

Alat FreeBayes – https://github.com/freebayes/freebayes

Skripta iz bioinformatike

CIGAR string - https://timd.one/blog/genomics/cigar.php