SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5751

Približni algoritam za brzo računanje poravnanja dvaju dugačkih nizova

Filip Sodić

Zagreb, srpanj 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 15. ožujka 2018.

ZAVRŠNI ZADATAK br. 5751

Pristupnik: **Filip Sodić (0036491335)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Približni algoritam za brzo računanje poravnanja dvaju dugačkih nizova**

Opis zadatka:

Optimalna metoda za računanje poravnanja između dva niza ima kvadratnu složenost što ju čini neprikladnom za dugačke nizove bioloških podataka kao što su sljedovi nukleotida pojedinih kromosoma. Dodatni problem predstavlja to što optimalna metoda ne uzima u obzir inverzije i translacije u računanje te često rješenje ne odgovara biološkim razlikama između dva slijeda nukleotida.
U radu je potrebno implementirati približni algoritam opisan u radu "A fast adaptive algorithm for computing whole-genome homology maps" te svoju implementaciju usporedti s implementacijom iz originalnog rada te s drugim rješenjima.
Rješenje treba biti napisano u programskom jeziku C/C++. Programski kod je potrebno komentirati i pri pisanju pratiti neki od standardnih stilova. Kompletnu aplikaciju postaviti na repozitorij Github.

Zadatak uručen pristupniku: 16. ožujka 2018.
Rok za predaju rada: 15. lipnja 2018.

Mentor:
Izv. prof. dr. sc. Mile Šikić

Djelovođa:
Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za završni rad modula:
Prof. dr. sc. Siniša Srbljić

Zahvaljujem profesoru Šikiću na svim uputama i savjetima koje mi je kao mentor pružio.

Zahvaljujem autorima programa MashMap *na izuzetno urednoj, čitkoj i dobro dokumentiranoj originalnoj implementaciji algoritma bez čijeg proučavanja moja implementacija ne bi bila moguća.*

SADRŽAJ

1. Uvod

Globalno poravnanje genoma problem je od vrlo velike važnosti. Mnogi ciljevi genomike, kao što su otkrivanje evolucijskog srodstva ili identifikacija zajedničkih karakteristika unutar vrste, ostvaruju se upravo poravnavanjem genoma reprezentativnih jedinki.

Pristup problemu globalnog poravnanja točnim algoritmima dinamičkog programiranja izrazito je neobećavajući. Njihova primjena na nizove reda veličine cijelih genoma vremenski je neizvediva već za najmanje organizme. Dodatan je problem što je pojam poravnanja u okviru takvih algoritama definiran strogo matematički. Drugim riječima, ne uvažavaju se mnoge specifičnosti bioloških nizova poput inverzije i translacije. Spomenuta su ograničenja motivirala razvoj mnogih heurističkih metoda poravnanja. Većina ih se temelji na ideji pronalaska malih segmenata velike međusobne sličnosti te njihovog proširenja i grupiranja u veće regije (engl. *seed-and-extend*). Usprkos mnogostruko većoj računalnoj učinkovitosti navedenih metoda u usporedbi s dinamičkim programiranjem, one su još uvijek računalno intenzivne. Posljednjih nekoliko godina dolazi do znatnog napretka u području analize dugih očitanja (engl. *long read analysis*) zahvaljujući novom pristupu temeljenom na tehnici *MinHash* (Broder, 1997)

Cilj ovog rada implementacija je algoritma globalnog poravnanja genoma koji bi bio brži i memorijski manje zahtjevan od drugih dostupnih opcija, a u isto vrijeme zadržao dovoljno visoku razinu preciznosti. Algoritam se temelji na kombinaciji uobičajenog pristupa računanja predstavnika (Schleimer et al., 2003) i relativno novog pristupa računanja sličnosti tehnikom *MinHash*. Osmislila ga je skupina znanstvenika početkom 2018. godine (Jain et al., 2018)[1].

[1] Originalna implementacija nalazi se ovdje

2. Pregled pojmova

U svrhu razvoja učinkovitog algoritma, potrebno je izgraditi prikladan model problema, odnosno, ograničiti problem određenim pretpostavkama. Kako bismo algoritam mogli detaljno objasniti, neophodno je pobrojiti matematičke pojmove i tehnike kojima se on koristi. Ovo poglavlje ujedno definira model problema i objašnjava tehnike korištene u njegovom rješavanju. U odjeljku 2.1 iznesena je za algoritam nužna pretpostavka o razdiobi pogrešaka unutar očitanja. Odjeljak 2.2 definira pojam i objašnjava važnost Jaccardovog koeficijenta sličnosti, a odjeljak 2.3 iznosi učinkovitu tehniku za računanje njegove aproksimacije. U odjeljku 2.4, objašnjen je vrlo popularan algoritam određivanja sličnosti traženjem predstavnika.

2.1. Model razdiobe pogrešaka

Pretpostavljamo da se pogreške na očitanjima pojavljuju međusobno nezavisno te su podložne Poissonovoj razdiobi. Neka je $\epsilon \in [0, 1]$ stopa pogreške (engl. *per-base error rate*). Stopa pogreške jednaka je vjerojatnosti pojavljivanja pogreške na pojedinoj bazi. Kako su događaji pojavljivanja pogrešaka međusobno nezavisni, očekivani broj pogrešaka unutar jedne k-torke iznosi $k \cdot \epsilon$. Konačno, koristeći očekivanje i pretpostavku o Poissonovoj razdiobi, dolazimo do vjerojatnosti očitanja k-torke bez pogrešaka:

$$\lambda = k \cdot \epsilon \qquad P(\text{broj pogrešaka} = 0) = \frac{\lambda^0}{0!} \cdot e^{-\lambda} = e^{-\epsilon k} \tag{2.1}$$

Zaključujemo da vjerojatnost nepojavljivanja pogreške unutar svake pojedine k-torke iznosi $e^{-\epsilon k}$. Pretpostavit ćemo da iznesena ograničenja, a time i jednakost 2.1, vrijede za sve vrste pogrešaka (Fan et al., 2015).

2.2. Jaccardov koeficijent sličnosti

Jaccardov koeficijent sličnosti (engl. *Jaccard*) statistička je mjera sličnosti dvaju konačnih skupova. Definira se kao omjer kardinalnih brojeva njihovog presjeka i njihove

unije. (Jaccard, 1901):

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Jaccardova udaljenost (engl. *Jaccard distance*) statistička je mjera različitosti dvaju konačnih skupova komplementarna Jaccardovom koeficijentu sličnosti. Računa se na sljedeći način (Jaccard, 1901):

$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

U kontekstu bioinformatike, definicija Jaccardovog koeficijent sličnosti proširuje se i na nizove. Pretpostavimo da su $\mathcal{X}$ i $\mathcal{Y}$ skupovi k-torki dobiveni respektivno iz nizova X i Y. Jaccardov koeficijent sličnosti dvaju nizova jednak je Jaccardovom koeficijentu sličnosti pripadajućih skupova k -torki: $J(X,Y) = |\mathcal{X} \cap \mathcal{Y}|/|\mathcal{X} \cup \mathcal{Y}|$. Korisno je primjetiti da smo definiravši sličnost nizova na ovaj način izgubili pojam o njihovom temeljnom svojstvu, poretku. Korištenjem modela razdiobe pogrešaka uvedenog odjeljkom 2.1 dolazimo do izraza koji povezuje koeficijent sličnosti sa stopom pogreške unutar jednog poravnanja.

Razmatramo pogreške do kojih dolazi na očitanju A izvedenom iz dugačkog niza B. Uzmimo da je A očitanje dobiveno čitanjem B_i, gdje B_i označava podniz niza B duljine $|A|$ s početkom na poziciji i. Kada očitanja ne bi bila podložna pogreškama, niz A bio bi jednak nizu B_i te bi za njihov koeficijent sličnosti vrijedilo $J(A,B_i) = 1$. Kako su za svaki $\epsilon > 0$ pogreške pri čitanju moguće, Jaccardov koeficijent sličnosti izvornog niza i njegovog očitanja u praksi je uvijek manji od 1. Neka je n ukupan broj k-torki prisutnih u čitanom podnizu B_i, a c neka je broj k-torki očitanih bez pogreške. S povećanjem broja k-torki, omjer c/n konvergira prema vjerojatnosti nepojavljivanja pogreške unutar jedne k-torke. Prema tome, uz dovoljno velik n i pretpostavke iznesene u odjeljku 2.1, može se zaključiti da vrijedi:

$$\frac{c}{n} \approx e^{-\epsilon k} \tag{2.2}$$

Omjer c/n naziva se još i *vjerojatnost preživljavanja k-torke*, a njegovo očekivanje iznosi $e^{-\epsilon k}$. Budući da su nizovi A i B_i jednako dugački, uzmemo li dovoljno velik k da vjerojatnost pojavljivanja dvije identične k-torke unutar razmatranih nizova postane zanemariva, zaključujemo da vrijedi:

$$|\mathcal{A} \cap \mathcal{B}_i| = c \qquad |\mathcal{A} \cup \mathcal{B}_i| = 2n - c \qquad J(A,B_i) = \frac{c}{2n-c} \tag{2.3}$$

Koristeći izraze 2.2 i 2.3, dolazimo do funkcija koje stopu pogreške ϵ povezuju s koeficijentom sličnosti J (Jain et al., 2017):

$$\mathbb{E}(\epsilon) = \mathcal{F}(J,k) = -\frac{1}{k} \cdot \log\left(\frac{2J}{1+J}\right) \qquad \mathbb{E}(J) = \mathcal{G}(\epsilon,k) = \frac{1}{2e^{\epsilon k}-1} \tag{2.4}$$

Sve pretpostavke o duljini k-torki i njihovom broju iznesene u ovom poglavlju valjane su u kontekstu problema poravnanja. Prema tome, izvedene će funkcije uvijek vrijediti. Ipak, kako su vrijednosti ovih izraza očekivanja, poželjno ih je promatrati unutar određenog intervala tolerancije. Izrazi 2.4 u algoritmu se koriste za procjenu kvalitete pronađenih preklapanja.

2.3. MinHash aproksimacija

Jaccardov koeficijent sličnosti pokazao se dobrom heuristikom određivanja stvarne sličnosti nizova. Međutim, postupak njegovog izračuna zahtijeva konstrukciju skupova i izvođenje operacija unije i presjeka. Kod rada s izuzetno dugačkim nizovima, navedene su operacije vrlo skupe i dugotrajne što ih čini neprikladnima za korištenje u brzim algoritmima.

Algoritam *MinHash* brza je i prostorno učinkovita tehnika za računanje nepristrane procjene Jaccardovog koeficijenta sličnosti koja ne zahtijeva eksplicitno računanje presjeka i unije skupova (Broder, 1997).

Neka je K skup svih mogućih k-torki, a V neki skup s potpuno definiranim poretkom. Definirajmo funkciju sažimanja (engl. *hash function*) $\Omega : K \to V$ koja svakoj k-torki pridružuje jedan element skupa V. Funkciju Ω koristimo za definiciju poretka nad k-torkama. Razmotrimo vjerojatnost da skupovi $\mathcal{A}, \mathcal{B}_i \subset K$ dijele istu najmanju k-torku uz pretpostavku da pri određivanju poretka ne dolazi do kolizija (funkcija Ω je injektivna). Drugim riječima, zanima nas kolika je vjerojatnost da se najmanja k-torka unutar unije skupova $\mathcal{A}$ i $\mathcal{B}$ nalazi upravo u njihovom presjeku. Prema tome, tražena vjerojatnost iznosi:

$$P(\min_{x\in A}\Omega(x) = \min_{x\in B_i}\Omega(x)) = \frac{|\mathcal{A}\cap\mathcal{B}_i|}{|\mathcal{A}\cup\mathcal{B}_i|} = J(A,B_i) \tag{2.5}$$

Izraz 2.5 govori nam sljedeće. Ako je r slučajna varijabla koja poprima vrijednost 1 kada vrijedi $\min_{x\in A}\Omega(x) = \min_{x\in B_i}\Omega(x)$ i vrijednost 0 u svim ostalim slučajevima, onda nam r daje nepristranu procjenu za $J(A,B_i)$. Budući da varijabla r poprima isključivo vrijednosti 0 ili 1, njena je varijanca previsoka da ona bude koristan procjenitelj koeficijenta sličnosti. Ideja MinHash algoritma smanjivanje je visoke varijance

kombiniranjem više varijabli konstruiranih upravo na opisani način. Postoji nekoliko različitih pristupa od kojih je najpopularniji objašnjen u nastavku.

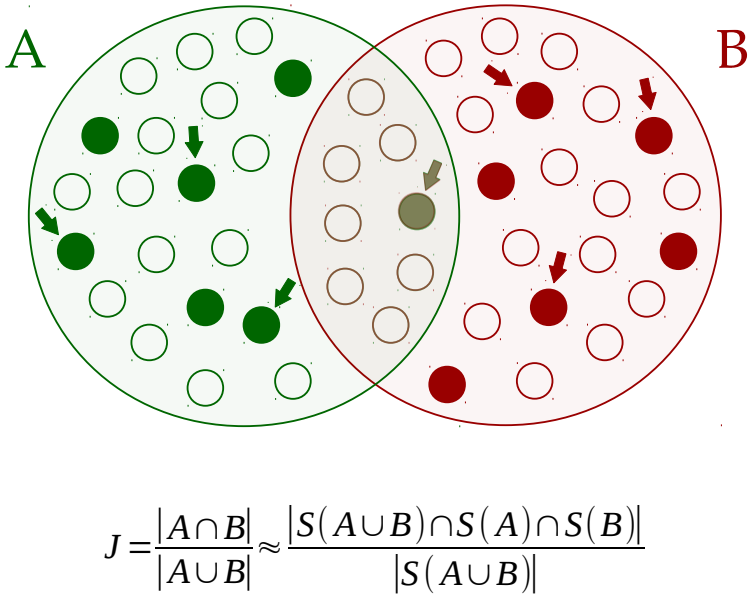
Neka je s fiksirani parametar. Skica (engl. *sketch*) skupa $\mathcal{A}$ naziv je za skup s najmanjih sažetaka k-torki iz $\mathcal{A}$:

$$S(\mathcal{A}) = \min_s\{\Omega(x) \mid x \in \mathcal{A}\} \tag{2.6}$$

Cilj skiciranja smanjivanje je varijance kako bismo dobili što vjerodostojniju procjenu Jaccardovog koeficijenta sličnosti. Umjesto razmatranja najmanjeg elementa, skiciranje razmatra njih s. Dokazano je da se nepristrana procjena Jaccardovog koeficijenta sličnosti može dobiti na sljedeći način (Broder, 1997):

$$J(A, B_i) \approx \frac{|S(\mathcal{A} \cup \mathcal{B}_i) \cap S(\mathcal{A}) \cap S(\mathcal{B}_i)|}{|S(\mathcal{A} \cup \mathcal{B}_i)|} \tag{2.7}$$

Što je parametar s veći, to nam ovaj postupak daje točniju procjenu stvarnog koeficijenta sličnosti.



Slika 2.1: Ilustracija algoritma *MinHash* za dva proizvoljna skupa A i B. Zeleno obojeni krugovi označavaju pripadnost skupu $S(A)$, dok crveno obojeni krugovi označavaju pripadnost skupu $S(B)$. Pripadnost skupu $S(A \cup B)$ naznačena je strelicama. Veličina skice s u ovom primjeru iznosi 7. Jaccardov koeficijent sličnosti iznosi $1/6$, a *MinHash* aproksimacija $1/7$.

2.4. Winnowing

Izvorno predložen za otkrivanje plagijata i određivanja sličnosti između tekstualnih dokumenata (Schleimer et al., 2003), algoritam *Winnowing* pokazao se izuzetno korisnim za brzu usporedbu nizova u bioinformatici. Za razliku od algoritma *MinHash*, ovaj algoritam nudi opciju razmatranja udaljenosti između pronađenih podudaranja. Prema tome, sličnosti nizova ovise i o položajima zajedničkih k-torki, a ne samo o njihovoj prisutnosti.

Algoritam se temelji na uzorkovanju najmanjih k-torki u uzastopnim prozorima fiksne duljine. Poredak nad k-torkama definiran je funkcijom sažimanja koja se bira ovisno o potrebama algoritma. Ponekad je poželjno k-torke birati slučajno, dok se u drugim slučajevima preferiraju određena svojstva kao što su učestalost ili prisutnost pojedinih baza.

Neka A_0 označava skup uređenih parova oblika $\langle k_i, i\rangle$ svih k-torki i njihovih pozicija unutar niza A. Neka je w fiksna duljina kližućeg prozora, a K_j neka bude skup w uzastopnih k-torki s početkom na poziciji j: $K_j = \{\langle k_i, i\rangle \mid j \leq i < j + w\}$. Definiramo li poredak nad k-torkama funkcijom Ω, skup predstavnika (engl. *minimizer*) dobivenih algoritmom *Winnowing* nad nizom A definira se na sljedeći način:

$$W(A) = \{\min_{\langle k,i\rangle \in K_j} \langle \Omega(k), i\rangle \mid 0 \leq j \leq |A_0| - w\}$$

Funkcija poretka nad uređenim parovima k-torki i pozicija definirana se kao:

$$\min(\langle k_1, i_1\rangle, \langle k_2, i_2\rangle) = \begin{cases} \langle k_1, i_1\rangle, & \text{ako } k_1 < k_2 \text{ ili } (k_1 = k_2 \text{ i } i_1 < i_2). \\ \langle k_2, i_2\rangle, & \text{inače.} \end{cases}$$

Naveden je najčešći i najjednostavniji način definiranja predstavnika u kojem oni sadrže podatke samo o vrijednosti k-torke i njenom položaju unutar niza. Informacije sadržane unutar strukture predstavnika mogu varirati ovisno o potrebama algoritma.

Dokazano je da očekivani broj predstavnika odabranih iz niza A iznosi $2|A_0|/w$ (Schleimer et al., 2003). Za pronalazak skupa $W(A)$ koristit ćemo učinkovit algoritam čija vremenska složenost iznosi $\mathcal{O}(|A|)$, a prostorna složenost $\mathcal{O}(w)$ (Jain et al., 2017).

3. Formulacija problema

3.1. Globalno poravnanje nizova

Cilj algoritma pronalazak je optimalnog puta poravnanja između dva dugačka niza. Nizovi će uvijek predstavljati cijele genome određenih organizama. Nazovimo nizove Q (engl. *query*) i R (engl. *reference*).

Optimalan put poravnanja određen je odabranim podskupom svih lokalnih poravnanja minimalne duljine l_0 s maksimalnom stopom pogreške ϵ_{max}. Lokalna poravnanja koja su dio optimalnog puta moraju biti odabrana na temelju biološke važnosti. Potrebno je eliminirati inferiorna poravnanja do kojih može doći uslijed prisutnosti paralognih segmenata te lažna poravnanja (engl. *false positive*) koja se pojavljuju kao posljedica jednostavnih ponavljanja unutar istog niza.

U svrhu pronalaska lokalnih poravnanja, potrebno je razviti algoritam koji će s velikim nizom poravnati s-kraja-na-kraj (engl. *end-to-end*) mnogo manjih segmenata. Kako je ovo zaseban problem opisan u prethodnom istraživačkom radu iste skupine znanstvenika (Jain et al., 2017), njegova je formulacija izdvojena u nastavku.

3.2. Poravnanje očitanja u okviru referentnog niza

Uz očitanje A i zadanu maksimalnu stopu pogreške ϵ_{max}, potrebno je pronaći sve pozicije unutar referentnog niza B s kojima se očitanje preklapa sa stopom pogreške manjom ili jednakom ϵ_{max}. Zbog duljine nizova, pristup problemu točnim algoritmom poravnanja kvadratne složenosti računalno je neizvediv. Problem nije moguće riješiti potpuno točno te se moramo zadovoljiti aproksimacijom.

Za aproksimaciju sličnosti nizova koristimo Jaccardov koeficijent sličnosti (detaljno opisan u poglavlju 2.2). Neka je A očitanje dugačkog niza B dobiveno čitanjem podniza B_i. Prisjetimo se, B_i tada označava podniz niza B duljine $|A|$ s početkom na poziciji i. Uz zadanu veličinu k-torke k, tražimo sve pozicije i u nizu B za koje je

zadovoljen uvjet:

$$J(A, B_i) \geq \mathcal{G}(\epsilon_{max}, k) - \delta \tag{3.1}$$

Uz zadanu maksimalnu stopu pogreške ϵ_{max}, koristeći vezu iznesenu izrazom 2.4, dolazimo do minimalne zahtijevane granice Jaccardovog koeficijenta sličnosti. Kao što je spomenuto kod izvoda funkcije $\mathcal{G}$, prag koji ona određuje potrebno je smanjiti kako bi se u model uračunalo određeno odstupanje od očekivanja. Parametar δ definira toleranciju na pogrešku u procjeni koristeći prikladan interval povjerenja (Jain et al., 2017).

Osim što se može izračunati koristeći maksimalnu stopu pogreške, minimalni identitet može biti zadan i izravno. Ako je umjesto maksimalne stope pogreške ϵ_{max} zadan minimalni identitet I_{min}, lako se dolazi do alternativnog uvjeta odluke o preklapanju. Za identitet poravnanja vrijedi $I = 1 - \epsilon$. Budući da nam je poznat koeficijent sličnosti $J(A, B_i)$, pomoću izraza 2.4 i tolerancije na pogrešku δ dolazimo do izraza $I = 1 - (\mathcal{F}(J(A, B_i), k) - \delta)$. Dobiveni identitet mora biti veći ili jednak zadanom identitetu I_{min}. Prema tome, konačan uvjet preklapanja glasi:

$$1 - \mathcal{F}(J(A, B_i), k) + \delta \geq I_{min} \tag{3.2}$$

4. Algoritam

Algoritam konceptualno možemo rastaviti u dva dijela. Zadatak prvog dijela pronalazak je svih parova podnizova iz Q i R čija poravnanja zadovoljavaju uvjete minimalne duljine l_0 i maksimalne dopuštene stope pogreške ϵ_{max}. Ovaj dio algoritma opisan je poglavljem 4.1. Drugi dio algoritma zadužen je za eliminaciju redundantnih i paralognih poravnanja, opisan je poglavljem 4.2.

4.1. Računanje granica lokalnih poravnanja

4.1.1. Osnovni algoritam

Potrebno je pronaći sva lokalna poravnanja oblika $Q[i..j]$ između nizova Q i R sa stopom pogreške po bazi manjom od ϵ_{max} i duljinom većom od l_0.

Prvi je korak podijeliti niz Q u nepreklapajuće fragmente duljine $l_0/2$. Ako se podniz niza Q minimalne duljine l_0, nazovimo ga Q_{sub}, preklapa s nekim podnizom niza R uz stopu pogreške ϵ koja je manja od ϵ_{max}, tada postoji barem jedan fragment veličine $l_0/2$ koji se poravnava s-kraja-na-kraj s dijelom optimalnog puta poravnanja uz očekivani broj pogrešaka $\epsilon \cdot l_0/2$. Za poravnanja pojedinih fragmenata koristimo algoritam opisan u odjeljku 4.1.2. Nakon pronalaska svih lokalnih poravnanja, ona se grupiraju algoritmom opisanom u odjeljku 4.1.3

4.1.2. Poravnanje fragmenta

U ovom odjeljku detaljno je opisan algoritam poravnavanja jednog fragmenta niza Q duljine $l_0/2$ s referentnim nizom R. Navedeni problem možemo promatrati kao problem poravnanja dugačkog očitanja s referentnim nizom čija je formalna definicija iznesena u poglavlju 3.2. Kako bismo ostali konzistentni prethodno definiranim oznakama, fragment niza Q koji interpretiramo kao očitanje označujemo s A, a referentni niz R označujemo s B.

Eksplicitno računanje Jaccardovog koeficijenta sličnosti niza A sa svakim podnizom B_i dovodi do složenosti previsoke za praktičnu primjenu. Potrebno je još jednom napraviti kompromis između točnosti i brzine algoritma. Kombinacijom tehnika skiciranja i traženja predstavnika opisanih redom u poglavljima 2.3 i 2.4 razvijamo heuristiku za učinkovito računanje procjene koeficijenta sličnosti.

Heuristika procjene Jaccardovog koeficijenta sličnosti

Primjenom algoritma *Winnowing* na niz A dolazimo do skupa predstavnika $W(A)$. Svaki je predstavnik uređena trojka $\langle h, p, d\rangle$ koja redom čuva podatke o sažetku k-torke (engl. *hash*), njenoj poziciji unutar niza i njenom smjeru (engl. *direction, strand*). Tehnikom skiciranja opisanom u poglavlju 2.3 računamo skicu skupa $W(A)$:

$$S(\mathcal{A}) = \min_{s}\{h \mid \langle h, p, d\rangle \in W(A)\} \tag{4.1}$$

Primjenom jednadžbe 2.7 na skupove predstavnika, za koeficijent sličnosti J definira se procjena $\mathcal{J}$ na sljedeći način:

$$J(A, B_i) \approx \frac{|S(W(A) \cup W(B_i)) \cap S(W(A)) \cap S(W(B_i))|}{|S(W(A) \cup W(B_i))|} = \mathcal{J}(A, B_i) \tag{4.2}$$

Cijela ideja iza skiciranja ostaje ista, jedina je razlika što umjesto biranja uzorka najmanjih sažetaka iz skupa svih k-torki ovdje biramo uzorak najmanjih sažetaka iz skupa predstavnika. Navedena modifikacija znatno smanjuje skup ulaznih podataka, a ne utječe na točnost procjene (Jain et al., 2017).

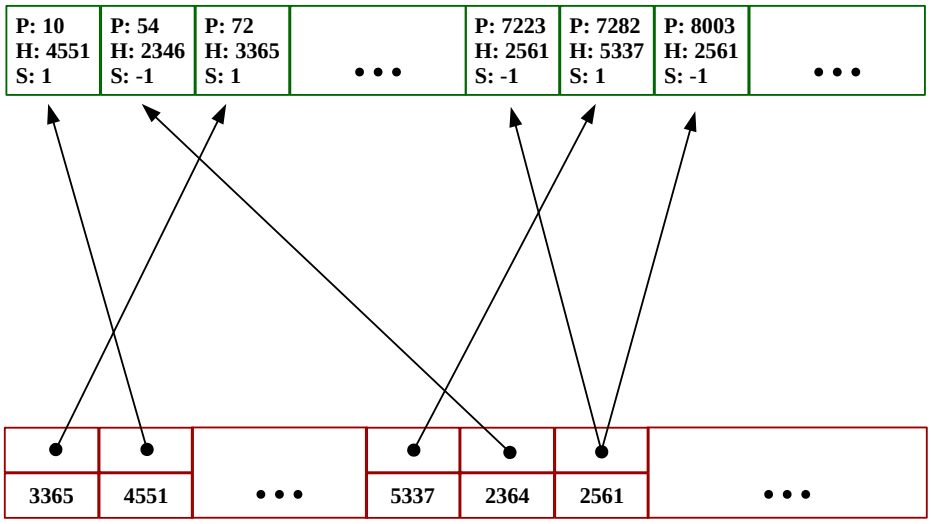
Računanje skupa predstavnika

Kako bismo uspješno koristili izraz 4.2, potrebno je pronaći predstavnike niza A i podnizova B_i za $0 \leq i \leq |B| - |A|$. Umjesto da računamo $W(B_i)$ za svaki podniz zasebno, odjednom računamo skup predstavnika cijelog niza $W(B)$. Izdvajanje predstavnika pojedinog podniza ostvarujemo dodatnom podatkovnom strukturom ili binarnim pretraživanjem. Potrebno je osigurati i strukturu za učinkovit pristup predstavnicima preko njihovih sažetaka.

Skup predstavnika $W(B)$ pohranjen je u memoriju kao polje uređenih trojki $\langle h, p, d\rangle$. Pohrana predstavnika unutar polja osigurava povoljan memorijski raspored i brz slijedni prolazak po članovima. Glavni je nedostatak linearna složenost pristupa predstavnicima preko njihovih sažetaka.

Kako bismo omogućili brži pristup predstavnicima preko sažetaka, uz polje predstavnika konstruiramo i asocijativno polje $\mathcal{H}$ s preslikavanjem $sazetak \rightarrow \{\langle h, p, d\rangle \mid$

$\langle h, p, d\rangle \in W(B) \wedge h = sazetak\}$. Pod pretpostavkom da je asocijativno polje implementirano tablicom raspršenog adresiranja, predstavnicima je preko njihovih sažetaka moguće pristupati u konstantnoj složenosti. Opisane strukture prikazane su na slici 4.1.



Slika 4.1: Polje predstavnika $W(B)$ označeno je zelenom bojom, odmah je po stvaranju prirodno sortirano prema stvarnim pozicijama u nizu. Asocijativno polje $\mathcal{H}$ označeno je crvenom bojom. Radi što manjeg zauzeća, polje $\mathcal{H}$ referencira indekse polja $W(B)$ preko njihovih pozicija. Budući da sažeci predstavnika nisu nužno jedinstveni, preslikavanje polja $\mathcal{H}$ je $1 : n$. U praksi je ovo jednostavno ostvarivo tablicom raspršenog adresiranja kojoj su ključevi sažeci, a vrijednosti liste pozicija.

Predstavnicima je potrebno moći učinkovito pristupati i temeljem njihove pozicije. Upravo se tako vrši izdvajanje predstavnika promatranog podniza B_i iz cijelog skupa $W(B)$. Budući da algoritam *Winnowing* nizom B prolazi slijedno, polje $W(B)$ sortirano je uzlazno po pozicijama na kojima su predstavnici pronađeni. Prema tome, pronalazak bilo kojeg predstavnika temeljem njegove pozicije ostvariv je binarnim pretraživanjem u logaritamskom vremenu. Uz određenu memorijsku cijenu, moguće je postići dodatno ubrzanje. Konstruiramo li tablicu raspršenog adresiranja s preslikavanjem $pozicija \rightarrow \langle h, pozicija, d\rangle$, složenost pristupa predstavniku preko njegove

pozicije postaje konstantnom.

Računanje poravnanja fragmenta

Kako bismo ustanovili poravnanje niza A s određenim podnizom B_i, potrebno je ispuniti uvjet 3.1. Umjesto eksplicitnog računanja Jaccardovog koeficijenta sličnosti $J(A, B_i)$, računamo njegovu procjenu definiranu izrazom 4.2. Dodatno, Kako bismo izbjegli računanje procjene $\mathcal{J}$ za svaki mogući par nizova, razvijamo heuristiku odabira kandidata. Uz oznake:

$$\tau = \mathcal{G}(\epsilon_{max}, k) - \delta \qquad \text{i} \qquad W_h(A) = \{h \mid \langle h, p, d\rangle \in W(A)\}$$

te uvjet $s \leq |W_h(A)|$, vrijedi:

$$\mathcal{J}(A, B_i) \geq \tau \implies |W_h(A) \cap W_h(B_i)| \geq s \cdot \tau \qquad \forall i, 0 \leq i \leq |B| - |A| \tag{4.3}$$

Dokaz iznesene implikacije dostupan je u literaturi (Jain et al., 2017), a možemo ju koristiti u odabiru parova nizova za koje se računa procjena koeficijenta sličnosti. Procjenu $\mathcal{J}$ računamo samo za parove A i B_i koji ispunjavaju uvjet $|W_h(A) \cap W_h(B_i)| \geq s \cdot \tau$. Kako bismo maksimizirali preciznost procjene i učinkovitost filtra bez kršenja uvjeta $s \leq |W_h(A)|$, veličinu skice s postavljamo da bude jednaka $|W_h(A)|$. Daljnji se algoritam dijeli u dvije faze. Prva faza bira kandidate na temelju uvjeta 4.3, a druga za odabrane kandidate računa procjenu $\mathcal{J}$. Ako je procjena koeficijenta sličnosti $\mathcal{J}(A, B_i)$ veća od praga τ, pozicija i sprema se kao pozicija poravnanja očitanja s referentnim nizom, a mjera njegove kvalitete jednaka je procjeni identiteta. Do procjene identiteta dolazi se korištenjem istog postupka kao i kod izraza 3.2 uz razliku da se ovdje procjena identiteta ne podiže se za prag tolerancije δ.

Izdvajanje kandidata

Izdvajanje potencijalnih poravnanja prikazano je pseudokodom 1. Računaju se sve pozicije i za koje vrijedi $|W_h(A) \cap W_h(B_i)| \geq \lceil s \cdot \tau \rceil = m$. Izlaz algoritma lista je raspona pozicija unutar kojih je spomenuti uvjet ispunjen. Na primjer, raspon $\langle x, y\rangle$ implicira da za svaki podniz B_i gdje je $x \leq i \leq y$ vrijedi $|W_h(A) \cap W_h(B_i)| \geq s \cdot \tau$.

Prvi je korak algoritma računanje predstavnika niza A. Pronađeni se predstavnici zatim pokušavaju pronaći i u referentnom nizu. Pretraga se učinkovito vrši asocijativnim poljem $\mathcal{H}$ objašnjenim u poglavlju 4.1.2. Svaki zajednički predstavnik dodaje se u listu P (reci 3-5). Drugim riječima, lista P sadrži pozicije na referentnom nizu za sve zajedničke predstavnike: $P = \{p \mid h \in W_h(A) \wedge \langle h, p, d\rangle \in W(B)\}$. Nakon

obrade svih predstavnika, lista P se sortira (redak 6). Da bi podniz B_i zadovoljio kriterij odabira, A i B_i moraju dijeliti minimalno m predstavnika, odnosno, nužno je da broj zajedničkih predstavnika s početnim pozicijama unutar segmenta referentnog niza $[i, i + |A|]$ iznosi najmanje m. Također slijedi da lista P tada mora sadržavati barem m susjednih elemenata tako da razlika pozicija u listi između posljednjeg i prvog od njih bude manja od $|A|$. Algoritam bira kandidatne regije upravo prema ovom kriteriju (uvjet u retku 10). Kako ista pozicija ne bi ostala zabilježena unutar više različitih raspona, svi se preklapajući rasponi spajaju (redak 14).

Algoritam 1 Odabir kandidatnih raspona

Ulaz: Očitanje A, asocijativno polje $\mathcal{H}$ ($sazetak \rightarrow$ polje pozicija), s, τ

Izlaz: Lista C kandidatnih raspona oblika $\langle x, y \rangle$

```
1:  m ← ⌈s · τ⌉
2:  P ← [ ]
3:  za h ∈ W_h(A) ponavljaj
4:      P.dodaj(H(h))
5:  kraj za
6:  sortiraj(P)
7:  C ← [ ]                                   ▷ Lista kandidatnih raspona
8:  za i od 1 do |P| − m ponavljaj
9:      j ← i + (m − 1)
10:     ako je P[j] − P[i] < |A| tada
11:         ⟨x_c, y_c⟩ ← ⟨ P[j] − |A| + 1, P[i]⟩
12:         ⟨x_l, y_l⟩ ← posljednji(C)
13:         ako je y_l ≥ x_c tada
14:             kraj raspona posljednjeg člana C postavi na max(y_l, y_c)
15:         inače
16:             C.dodaj(⟨x_c, y_c⟩)
17:         kraj ako je
18:     kraj ako je
19: kraj za
20: vrati C
```

Računanje procjena koeficijenata sličnosti

Drugi dio algoritma zadužen je za obradu kandidatnih raspona odabranih u prvom dijelu. Obrada jednog raspona $\langle x, y \rangle$ podrazumijeva računanje procjene $\mathcal{J}(A, B_i)$ za

$\forall i, x \leq i \leq y$.

Izračun procjena može se znatno pojednostaviti. Iz definicije skiciranja slijedi da kardinalni broj skice bilo kojeg skupa od minimalno s članova iznosi upravo s. Kako smo prethodno postavili da je $s = |W(A)|$, zaključujemo da je $|S(W(A) \cup W(B_i))| = s$. Koristeći ovu informaciju, pojednostavljujemo izraz 4.2:

$$\mathcal{J}(A, B_i) = \frac{|S(W(A) \cup W(B_i)) \cap S(W(A)) \cap S(W(B_i))|}{s} \tag{4.4}$$

Budući da nam je parametar s poznat, nazivnik izraza trivijalno je izračunati. Preostaje još pronaći učinkovit način izračuna brojnika. Do vrijednosti brojnika možemo doći tako da unutar najmanjih s elemenata skupa $W(A) \cup W(B_i)$ prebrojimo sve k-torke koje se pojavljuju u oba niza. Postupak je jednostavno ostvariv pomoću mape čiji su ključevi sažeci k-torki $\{h \in W_h(A) \cup W_h(B_i)\}$. Vrijednost pojedinog ključa postavlja se na 1 ako je k-torka prisutna u oba niza, a na 0 inače. Za izračun procjene koeficijenta sličnosti $\mathcal{J}$ sada je dovoljno sumu vrijednosti s najmanjih ključeva podijeliti veličinom skice s. Navedeni način izračuna procjene ostvaren je funkcijom `procijeniJaccard`. Za implementaciju mape koristimo strukturu $\mathcal{L}$ koja podržava sljedeće operacije:

- `dodaj(`H`)` - dodaje u $\mathcal{L}$ sve $h \in H$ tako da:
  - ako ključ h već postoji, njegova se vrijednost postavlja na 1
  - ako ključ h ne postoji, on se dodaje i vrijednost mu se postavlja na 0
- `ukloni(`H`)` - uklanja iz $\mathcal{L}$ sve $h \in H$ tako da:
  - ako vrijednost ključa h iznosi 1, ona se postavlja na 0
  - ako vrijednost ključa h iznosi 0, ključ se uklanja iz mape

U okviru algoritma koristi se i pomoćna funkcija `sažeciURasponu` koja vraća sve predstavnike niza B čije se pozicije nalaze unutar zadanog raspona.

Algoritam je opisan pseudokodom 2. Inicijalno se u mapu $\mathcal{L}$ postavljaju sažeci svih predstavnika niza A (reci 1-2). Početne vrijednosti svih ključeva iznose 0. Zatim slijedi obrada svakog raspona $\langle x, y \rangle$ (reci 4-22). Potrebno je računati $\mathcal{J}(A, B_i)$ za svaki $x \leq i \leq y$. Varijable i i j predstavljaju granice trenutno promatranog podniza B_i. Obje granice u svakoj iteraciji pomičemo za 1 kako bismo obradili sve podnizove s početkom unutar raspona $\langle x, y \rangle$. Obrada pojedinog raspona ostvarena je ažuriranjem trenutnog stanja u mapi i popratnom procjenom koeficijenta sličnosti. Stanje u mapi ažurira se uklanjanjem sažetaka predstavnika s početne pozicije te dodavanjem sažetka predstavnika s krajnje pozicije raspona trenutnog podniza, ako takvi postoje (reci 7, 13,

14). Postiže se učinak *kližuće mape*. Specifikacije funkcija `dodaj` i `ukloni` osiguravaju da se sažeci predstavnika niza A u mapi nalaze uvijek, dok se predstavnici iz niza B klizno mijenjaju. Za svaki B_i, funkcija `procijeniJaccard` računa odgovarajuću procjenu koeficijenta sličnosti. Ako rezultat premašuje zadani minimalni prag τ, u izlaznu se listu dodaje trenutna početna pozicija s izračunatom procjenom identiteta (reci 8-11, 15-18). Smjer poravnanja jednak je smjeru većine predstavnika (radi jednostavnosti nije prikazano pseudokodom). Postupak se ponavlja dok ne obradimo sve podnizove s početkom unutar trenutnog raspona, odnosno, sve dok trenutna pozicija početka i ne premaši granicu y (redak 12).

Algoritam 2 Računanje procjena koeficijenata sličnosti

Ulaz: Polje predstavnika $W(B)$, lista kandidatnih raspona C, s, τ

Izlaz: Lista poravnanja M oblika $\langle p, \mathcal{J} \rangle$

$\mathcal{L} \leftarrow \{\ \}$
$\mathcal{L}$.`insert`$(W_h(A))$
$M \leftarrow [\ ]$
za $\langle x, y \rangle \in T$ **ponavljaj**
    $i \leftarrow x$
    $j \leftarrow x + |A|$
    $\mathcal{L}$.`dodaj`(`saželiURasponu`(i, j))
    $\mathcal{J} \leftarrow$ `procijeniJaccard`$(\mathcal{L})$
    ako je $\mathcal{J} \geq \tau$ **tada**
        M.`dodaj`$(\langle i, 1 - \mathcal{F}(\mathcal{J}, k) \rangle)$
    kraj ako je
    dok je $i \leq y$ **ponavljaj** ▷ Mapa $\mathcal{L}$ klizi po intervalu $\langle x, y \rangle$
        $\mathcal{L}$.`ukloni`(`saželiURasponu`$(i, i + 1)$)
        $\mathcal{L}$.`dodaj`(`saželiURasponu`$(j, j + 1)$)
        $\mathcal{J} \leftarrow$ `procijeniJaccard`$(\mathcal{L})$
        ako je $\mathcal{J} \geq \tau$ **tada**
            M.`dodaj`$(\langle i, 1 - \mathcal{F}(\mathcal{J}, k) \rangle)$
        kraj ako je
        $i \leftarrow i + 1$
        $j \leftarrow j + 1$
    kraj dok je
kraj za
vrati M

funkcija `saželiURasponu`(i, j)
    vrati $\{h \mid \langle h, p, d \rangle \in W(B) \wedge i \leq p < j\}$
kraj funkcije

funkcija `procijeniJaccard`$(\mathcal{L})$
    $zajednicki \leftarrow \sum_{k=0}^{s-1} \mathcal{L}[k]$
    vrati $zajednicki/s$
kraj funkcije

4.1.3. Grupiranje poravnanja

Dobivena lokalna poravnanja spajaju se na temelju redoslijeda i blizine unutar referentnog niza R. Recimo da poravnanja susjednih fragmenata niza Q, q_i, $q_{i+1}, ..., q_n$ odgovaraju redom pozicijama p_0, $p_1, ..., p_{n-i}$ niza R. Navedena ćemo poravnanja grupirati ako ispunjavaju uvjete:

$$p_0 \leq p_1 \leq \cdots \leq p_{n-1} \qquad \text{i} \qquad p_{k+1} - p_k \leq l_0 \text{ za } [0 \leq k < n - i]$$

Drugim riječima, zajedno se grupiraju sva poravnanja međusobno razmaknuta za manje od l_0 s redoslijedom jednakim izvornom redoslijedu fragmenata na nizu Q. Uzima se da su granice poravnanja nastalog grupiranjem jednake najmanjoj početnoj i najvećoj krajnjoj poziciji svih grupiranih poravnanja, dok mu je procjena identiteta izračunata jednostavnim uzimanjem prosjeka. Konačno, *vrijednost poravnanja* umnožak je prosječne procjene identiteta i ukupne duljine. Grupirana poravnanja ulazni su podaci nadolazećem algoritmu filtriranja, a njihova vrijednost predstavlja glavnu mjeru kvalitete po kojoj se filtriranje vrši.

4.2. Filtriranje poravnanja

Kao što je spomenuto u poglavlju 3.1, lokalna poravnanja koja ispunjavaju uvjete minimalne duljine i maksimalne stope pogreške nisu nužno biološki ispravna ili relevantna. U genomima velikih sisavaca i biljaka prisutan je znatan broj ponavljajućih nizova. Kako naš algoritam radi na temelju sličnosti dobivenoj iz lokalno raspršenih uzoraka, kod takvih slučajeva može doći do registracije nevažnih ili lažnih poravnanja.

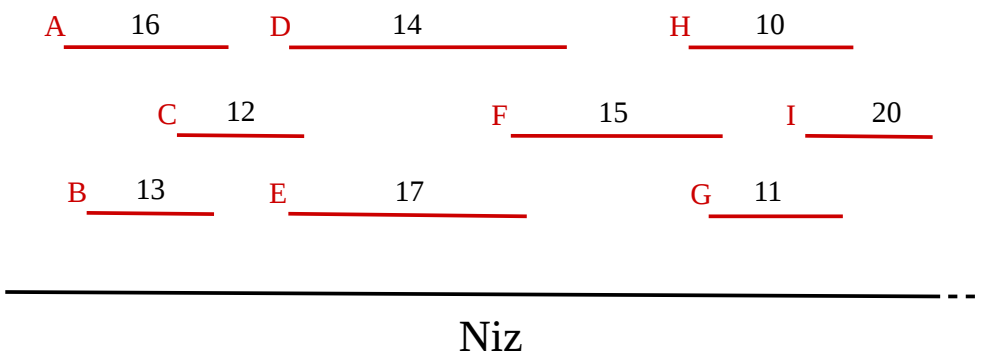
Potrebno je razviti heuristiku kojom ćemo u kontekstu svih pronađenih poravnanja odlučiti koja su od njih suvišna te ih treba odbaciti, a zatim osmisliti i algoritam koji će filtriranje obaviti u prihvatljivom vremenu.

4.2.1. Heuristika filtriranja

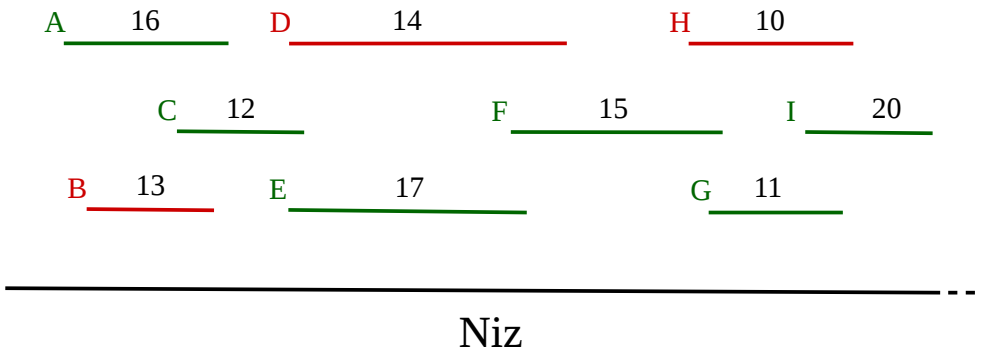
U okviru filtriranja, razmatrat ćemo vrijednosti i granice poravnanja izračunate u poglavlju 4.1.3. Svako poravnanje sadrži podatke o svojim pozicijama na oba niza. Prema tome, poravnanja možemo promatrati kao intervale nad nizovima Q ili R. Granice jednog takvog intervala određene su izravno pozicijama njegovog početka i kraja na promatranom nizu.

Ako se pozicija i nalazi unutar granica intervala poravnanja m, tada kažemo da je pozicija i pokrivena poravnanjem m, odnosno, da je m pokrivatelj pozicije i. Budući

da se počeci i krajevi poravnanja preklapaju, jednu poziciju može pokrivati više različitih poravnanja. Poravnanje proglašavamo redundantnim ako i samo ako je svaka njegova pozicija pokrivena nekim drugim poravnanjem veće vrijednosti. Sva se redundantna poravnanja odbacuju. Djelovanje heuristike prikazano je slikama 4.2 i 4.3.



Slika 4.2: Proizvoljni ulazni skup podataka za algoritam filtriranja. Nemamo informacije o korisnosti poravnanja, za sva se pretpostavlja da su redundantna. Iznad svakog poravnanja napisana je pripadajuća vrijednost.



Slika 4.3: Poravnanja nakon algoritma filtriranja. Korisna su poravnanja označena zelenom, a redundantna crvenom bojom. Poravnanje B redundantno je jer je poravnanje A bolji pokrivatelj svim pozicijama unutar njegovog intervala. Poravnanje D redundantno je jer su sve njegove pozicije bolje pokrivene poravnanjem E ili poravnanjem F. Slično vrijedi za poravnanje H - svaku njegovu poziciju bolje pokriva barem jedno od poravnanja F, G ili I.

4.2.2. Geometrijski algoritam filtriranja

Filtriranje po predloženoj heuristici mogli bismo ostvariti međusobnom usporedbom svih poravnanja. Složenost postupka filtriranja tada bi bila $\mathcal{O}(n^2)$. Za tipične veličine ulaznih podataka kvadratna vremenska složenost nije zadovoljavajuća.

Problem filtriranja poravnanja sličan je problemu pronalaska sjecišta dužina u dvodimenzionalnom prostoru. Navedeni problem također ima trivijalno rješenje s kvadratnom složenošću, ali poznato je i njegovo optimalno rješenje čija je vremenska složenost $\mathcal{O}(n \log(n))$ (Shamos i Hoey, 1976). U nastavku je opisana modifikacija poznatog algoritma iz 1976. godine pomoću koje možemo filtrirati poravnanja u vremenu $\mathcal{O}(n \log(n))$ (Jain et al., 2018).

Preduvjeti

Algoritam zahtijeva da nad skupom poravnanja bude definiran potpuni poredak. Jedno poravnanje smatra se većim od drugog ako mu je vrijednost veća. Ako su vrijednosti dva poravnanja jednake, tada se većim smatra poravnanje čiji se početak nalazi na kasnijoj poziciji niza.

Nadalje, potrebna nam je podatkovna struktura koja će čuvati poravnanja sortirana po prethodno definiranom poretku. Struktura mora podržavati operacije dodavanja i brisanja elemenata (uz očuvanje poretka). Također, potrebno je podržati operaciju koja će sva najvrjednija poravnanja sadržana u strukturi označiti korisnima (neredundantnima). Učinkovitu implementaciju tražene strukture možemo dobiti balansiranim binarnim stablom za pretraživanje (BSP). Binarna stabla za pretraživanje po definiciji čuvaju poredak svih svojih članova pri dodavanju i brisanju. Budući da je stablo balansirano, vremenska složenost spomenutih operacija nužno iznosi $\mathcal{O}(\log n)$. Konačno, sva najveća poravnanja možemo označiti korisnima u složenosti $\mathcal{O}(k \log n)$ gdje je k broj najvrjednijih poravnanja sadržanih u stablu.

Tijek algoritma

Tijek algoritma filtriranja na konceptualnoj razini prikazan je pseudokodom 3. Neka je n broj poravnanja prije filtriranja. Kako svako poravnanje predstavlja jedan interval, ukupan broj krajnjih pozicija iznosi $2n$ (nisu nužno jedinstvene). Radi jednostavnijeg i implementacijski neovisnog opisa algoritma, pseudokod prolazi po jedinstvenom skupu krajnjih pozicija. U praksi će vjerojatno biti povoljnije koristiti nejedinstvenu listu uz preskakanje obrađenih elemenata.

Skup krajnjih pozicija najprije je potrebno sortirati, a sva poravnanja označiti redundantnima (reci 1-6). Algoritam redom prolazi sve krajnje pozicije (redak 8) i za svaku od njih ažurira stanje u stablu. Poravnanja koja na trenutnoj poziciji počinju dodaju se u stablo (reci 9-11), a ona za koja trenutna pozicija predstavlja kraj iz stabla se uklanjaju (reci 12-14). Nakon ažuriranja stabla, sva se najvrjednija trenutno prisutna poravnanja označe korisnima (redak 15). Algoritam se prvo provodi nad nizom Q, a zatim nad nizom R.

Algoritam 3 Određivanje redundantnih poravnanja

Ulaz: Lista poravnanja $\mathcal{M}$

Izlaz: Lista poravnanja $\mathcal{M}$ uz označene redundantne članove

1: $\mathcal{E} \leftarrow \varnothing$ ▷ Skup jedinstvenih krajnjih pozicija
2: **za** $m \in \mathcal{M}$ **ponavljaj**
3: $\quad m.redundantan \leftarrow$ true
4: $\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{m.pocetak, m.kraj\}$
5: **kraj za**
6: sortiraj($\mathcal{E}$)
7: $\mathcal{T} \leftarrow \varnothing$ ▷ BSP za očuvanje poretka
8: **za** $pozicija \in \mathcal{E}$ **ponavljaj**
9: $\quad$ **za** $m \in \{m \mid m \in \mathcal{M} \wedge m.pocetak = pozicija\}$ **ponavljaj**
10: $\quad\quad \mathcal{T}$.dodaj(m)
11: $\quad$ **kraj za**
12: $\quad$ **za** $m \in \{m \mid m \in \mathcal{M} \wedge m.kraj = pozicija\}$ **ponavljaj**
13: $\quad\quad \mathcal{T}$.ukloni(m)
14: $\quad$ **kraj za**
15: $\quad \mathcal{T}$.označi_korisne()
16: **kraj za**

Dokaz točnosti

Prema definiciji redundancije iz poglavlja 4.2.1, svako poravnanje koje je najbolji pokrivatelj barem jednoj poziciji nije redundantno. Budući da je skup krajnjih pozicija uzlazno sortiran i algoritam ga prolazi slijedno, promjene trenutno aktivnih intervala moguće su jedino na krajnjim pozicijama, bilo dodavanjem ili uklanjanjem. Nakon ažuriranja sadržaja stabla u krajnjoj poziciji k, sigurni smo da će sve do krajnje pozicije $k + 1$ ono ostati isto. Iz navedenog slijedi kako sva u tom trenutku najvrjednija

poravnanja sigurno nisu redundantna jer niti jedna pozicija između k i $k + 1$ neće imati boljih pokrivatelja. Dokazali smo da svako poravnanje koje označimo korisnim uistinu to i jest. Još je potrebno dokazati da ne postoji korisno poravnanje koje algoritam ne prepoznaje. Budući da algoritam ažurira stanje stabla na svakoj krajnjoj poziciji, nemoguće je da mu promakne bilo kakva promjena najvrjednijih poravnanja. Prema tome, svako poravnanje koje je barem jednoj poziciji najbolji pokrivatelj bit će označeno korisnim.

Složenost

Složenost razmatramo u okviru broja poravnanja n. Ukupan broj krajnjih pozicija tada iznosi $2n$. Zaključujemo kako je složenost obilaska svih krajnjih pozicija $\mathcal{O}(n)$. U svakoj krajnjoj poziciji ažurira se stanje stabla dodavanjem ili uklanjanjem određenog broja poravnanja. Složenosti navedenih operacija za balansirano binarno stablo iznose $\mathcal{O}(\log n)$. Osiguramo li da funkcija za označavanje korisnosti pri nailasku na prvo otprije označeno poravnanje prestane s obilaskom stabla, tijekom cijelog algoritma korisnim se označi najviše n poravnanja čime se složenost ove operacije ograničava na $\mathcal{O}(n \log n)$ bez gubljenja točnosti (Jain et al., 2018). Uz složenost sortiranja $\mathcal{O}(n \log n)$ i ukupne složenosti ažuriranja stabla i funkcije označavanja korisnosti $\mathcal{O}(n \log n)$, zaključujemo kako je opisani algoritma filtriranja odozgo ograničen s $\mathcal{O}(n \log n)$.

5. Detalji implementacije

5.1. Ulazni i izlazni formati

Program nizove čita iz datoteka koje moraju biti u formatu FASTA. Dozvoljeno je da pojedina datoteka sadrži više nizova različitih naziva. Oni se tada redom spajaju i cijela datoteka interpretira se kao dugačak niz razdijeljen u više segmenata.

Program pronađena poravnanja ispisuje u datoteku organiziranu po uzoru na format PAF. Stupci izlazne datoteke odvojeni su razmacima i redom sadrže podatke popisane u tablici 5.1:

Stupac	Tip podatka	Opis
1	niz znakova	Ime niza Q
2	cijeli broj	Duljina niza Q
3	cijeli broj	Početak poravnanja na nizu Q
4	cijeli broj	Kraj poravnanja na nizu Q
5	znak	Smjer ("+" za isti smjer, "-" za reverzni komplement)
6	niz znakova	Ime niza R
7	cijeli broj	Duljina niza R
8	cijeli broj	Početak poravnanja na nizu R
9	cijeli broj	Kraj poravnanja na nizu R
10	decimalan broj	Procjena identiteta poravnanja

Tablica 5.1: Specifikacija formata izlazne datoteke

Ovako organiziranu datoteku možemo predati programu za generiranje grafičkog prikaza poravnanja nizova. Program je napisan u programskom jeziku Perl. Preuzet je iz repozitorija originalne implementacije algoritma (Jain et al., 2018).

5.2. Parametri programa

Algoritam traži predstavnike, računa skice i donosi odluke o klasifikaciji poravnanja u ovisnosti o nizu parametara. Konstantni parametri pobrojeni su tablicom 5.2.

Ime parametra	Iznos	Opis
k	16	Veličina jedne k-torke
Σ	4	Veličina abecede ulaznih nizova
I	0.85	Minimalni zahtijevani identitet (definiran umjesto ϵ_{max})
CI	0.75	Interval povjerenja u procjeni poravnanja
l_0	5000	Duljina fragmenata koji se poravnavaju lokalno

Tablica 5.2: Popis imena i vrijednosti konstantnih parametara

Iako fiksna veličina abecede sugerira da program radi isključivo za molekule DNA, moguće ga je jednostavno proširiti na nizove definirane nad proizvoljnim abecedama. Fiksna veličina k-torke odabrana je empirijski i po uzoru na originalnu implementaciju algoritma. Veličina prozora w nije fiksirana, bira se ovisno o duljini referentnog niza R (Jain et al., 2017).

5.3. Programska implementacija

Implementacija algoritma ostvarena je programskim jezikom C++. Standardna biblioteka jezika nudi nekoliko struktura koje znatno olakšavaju implementaciju. Spomenute strukture navedene su u tablici 5.3

Ime	Opis	Tip	Dio algoritma
$\mathcal{H}$	Asocijativno polje za pristup predstavnicima preko sažetaka	`std::unordered_map`	1
$\mathcal{L}$	Kližuća mapa za procjenu koeficijenta sličnosti	`std::map`	2
$\mathcal{T}$	BSP korišteno za filtriranje poravnanja	`std::set`	3

Tablica 5.3: Popis implementacija struktura korištenih u algoritmu

Potpuna implementacija algoritma dostupna je ovdje.

6. Rezultati

Svi su ispitivanja provedena na računalu čija je specifikacija dana tablicom 6.1

Operacijski sustav	Linux Ubuntu 16.04 LTS
Procesor	Intel Core i5-4200H @ 2.80GHz
Broj fizičkih jezgri	2
Maksimalan broj dretvi	4
Radna memorija	DDR3 8 GiB 1600 MHz (0,6 ns)

Tablica 6.1: Specifikacija računala na kojem je provedeno ispitivanje

Popis organizama nad kojima su ispitivanja provedena dan je tablicom 6.2.

Identifikator uzorka	Ime organizma	Veličina datoteke
NC_000913.3	*Escherichia coli*	4.48 MB
NC_016845.1	*Klebsiella pneumoniae*	5.15 MB
NC_002942.5	*Legionella pneumophila*	3.28 MB

Tablica 6.2: Organizmi nad kojima je provedeno ispitavanje

Implementacija algoritma uspoređivana je sa široko korištenim programom za poravnanje *Minimap*. Kako bismo računalnu učinkovitost algoritma opisanog u radu mogli usporediti s učinkovitošću onog koji koristi *Minimap*, ispitivanja se provode i na njegovoj originalnoj implementaciji, programu *MashMap*. Na ovaj smo način uklonili utjecaj implementacijskih detalja, neoptimalnosti i pogrešaka kojima bi nova implementacija algoritma mogla biti podložna. Razumno je očekivati kako su kod široko korištenih i profesionalno razvijenih programa poput *Minimapa* i *MashMapa* glavne

razlike u brzini i utrošku memorije izravna posljedica algoritama, a ne detalja ili nedostataka implementacije.

6.1. Očekivani rezultati

6.1.1. Točnost algoritma

Budući da je metoda na kojoj se temelji *Minimap* pouzdanija od metoda korištenih u ovom radu, očekuje se da će najviše poravnanja pronaći *Minimap*. *Minimap* nema ugrađenu heuristiku zanemarivanja redundantnih i lažnih poravnanja. Iz navedenog se razloga očekuje da će put optimalnog poravnanja kvalitetnije rekonstruirati *MashMap* i implementacija razvijena u okviru ovog rada.

6.1.2. Računalna učinkovitost algoritma

Algoritam opisan u ovome radu prvenstveno je osmišljen radi brzine i memorijske učinkovitosti. Iz tog je razloga smisleno očekivati da obje njegove implementacije (i originalna i nova) budu brže i memorijski učinkovitije nego što je *Minimap*. Ipak, novoj implementaciji nedostaju brojne optimizacije koje *MashMap* i *Minimap* kao široko korišteni programi imaju. Prema tome, vrlo je vjerojatno da implementacija razvijena u okviru rada bude i sporija i memorijski manje učinkovita od ostalih opcija. Upravo se iz tog razloga i promatraju rezultati programa *MashMap*.

6.2. Stvarni rezultati

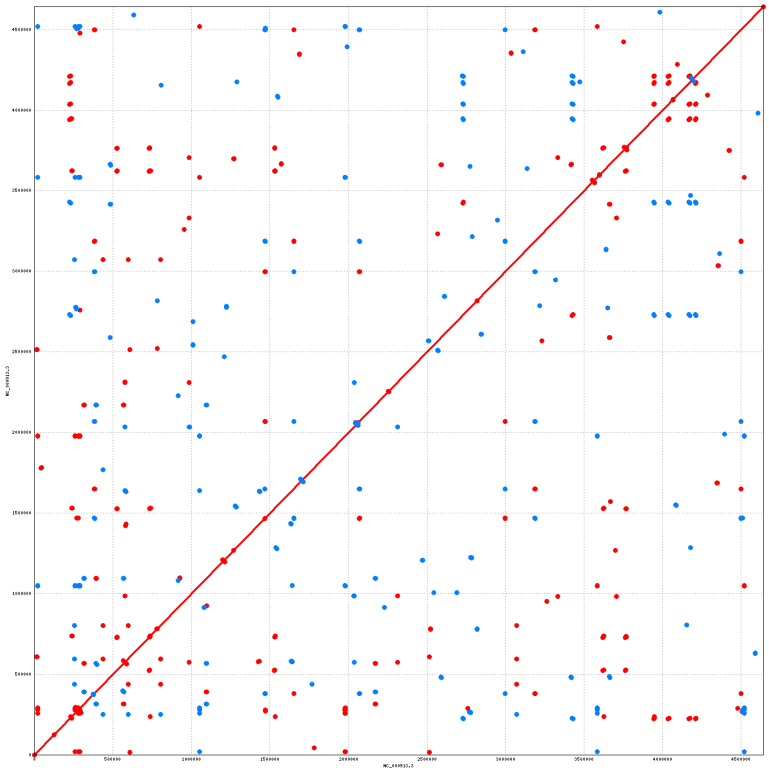
6.2.1. Točnost algoritama

Pronađeni optimalni putevi poravnanja prikazani su točkastim grafovima (engl. *dot plot*). Svaka os predstavlja pozicije na genomu jednog od uspoređivanih organizama. Crvena točka označava preklapanje istog smjera, a plava reverzno preklapanje. Grafovi su generirani skriptom posuđenom iz repozitorija programa *MashMap*.

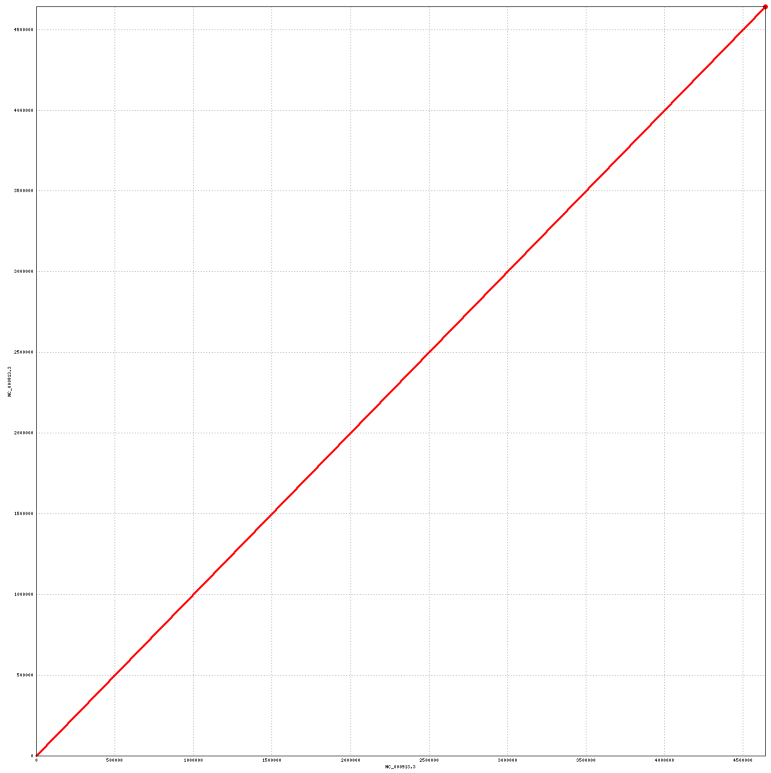
Usporedba identičnih genoma

Na slikama 6.1, 6.2 i 6.3 prikazani su rezultati usporedbe dva ista genoma vrste *Escherichia coli*. Očekivano, *Minimap* je ukupno pronašao najviše poravnanja, ali i najviše

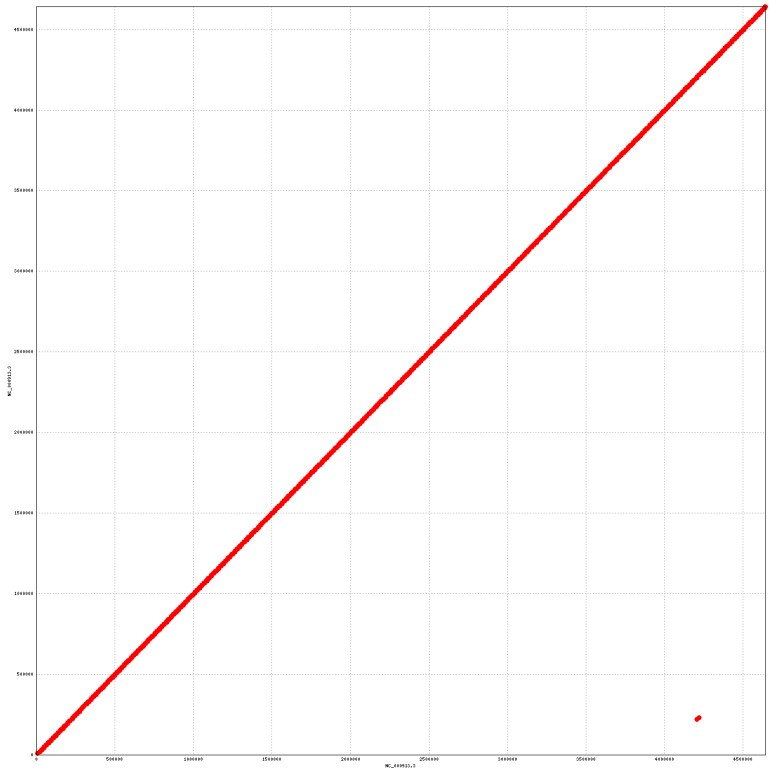
biološki nevažnih poravnanja. *MashMap* i nova implementacija zahvaljujući geometrijskom algoritmu filtriranja jasno određuju optimalan put poravnanja. Radi boljih algoritama grupiranja, *MashMap* i *Minimap* grupiraju sva poravnanja duž optimalnog puta u jedno.



Slika 6.1: Minimap: Usporedba dva ista genoma vrste *Escherichia coli*



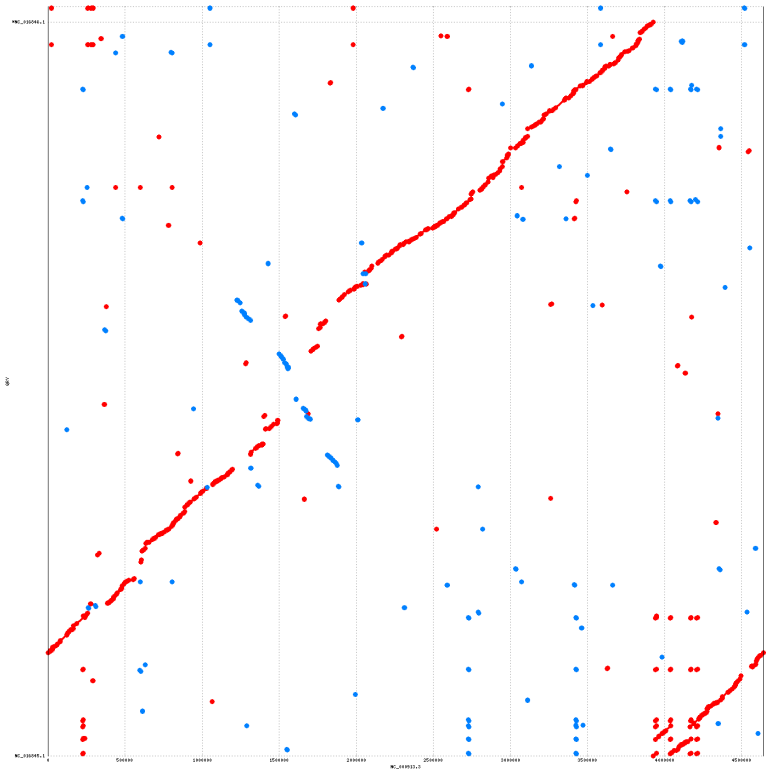
Slika 6.2: Mashmap: Usporedba dva ista genoma vrste *Escherichia coli*



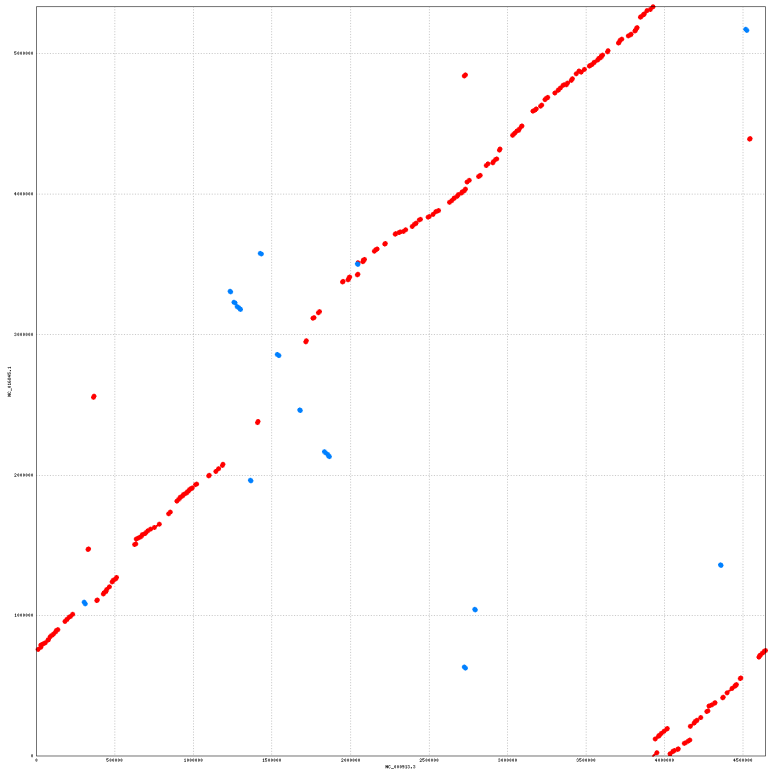
Slika 6.3: Implementacija rada: Usporedba dva ista genoma vrste *Escherichia coli*

Usporedba sličnih genoma

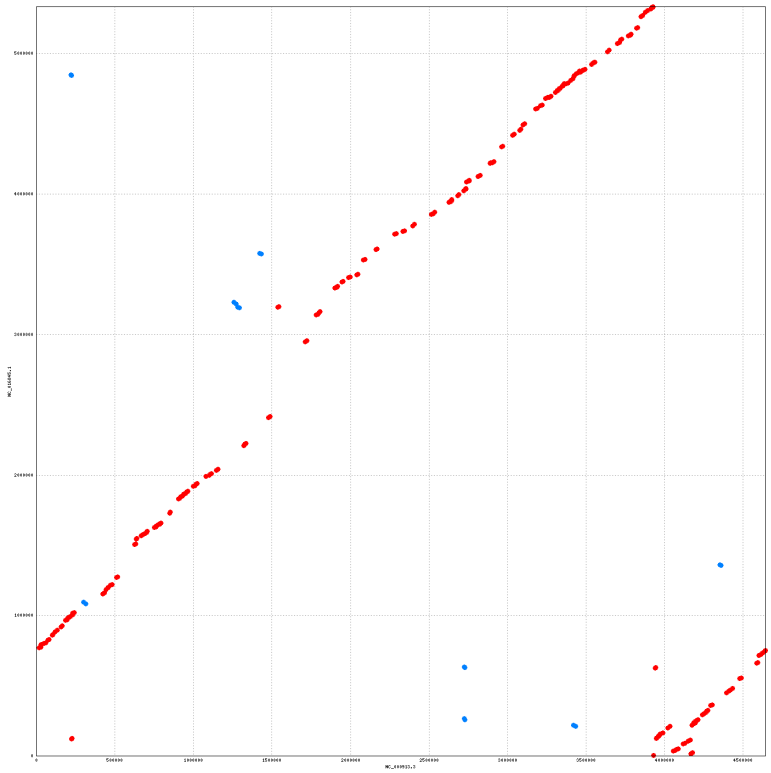
Na slikama 6.4, 6.5 i 6.6 prikazani su rezultati usporedbe genoma vrste *Klebsiella pneumoniae* s genomom vrste *Escherichia coli*. Najviše poravnanja ponovno prijavljuje *Minimap*. Najviše biološki nevažnih poravnanja također ponovno prijavljuje *Minimap*. Iako *MashMap* i nova implementacija put poravnanja ponovno prepoznaju puno čišće, zbog nepreciznosti algoritma ne uspijevaju jasno uočiti reverzno preklapanje na području prekida optimalnog puta.



Slika 6.4: Minimap: Usporedba genoma vrsta *Escherichia coli* i *Klebsiella pneumoniae*



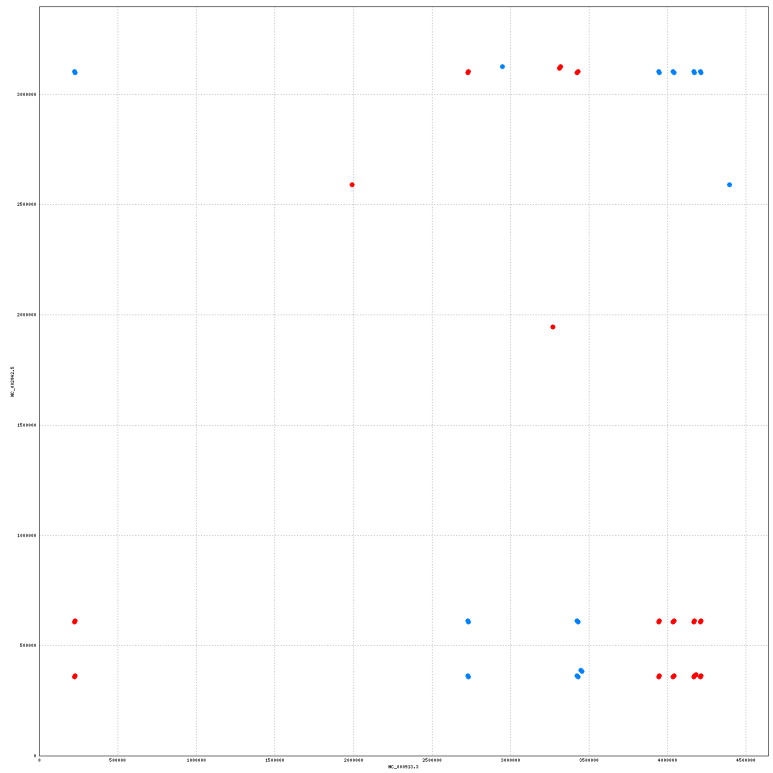
Slika 6.5: Mashmap: Usporedba genoma vrsta *Escherichia coli* i *Klebsiella pneumoniae*



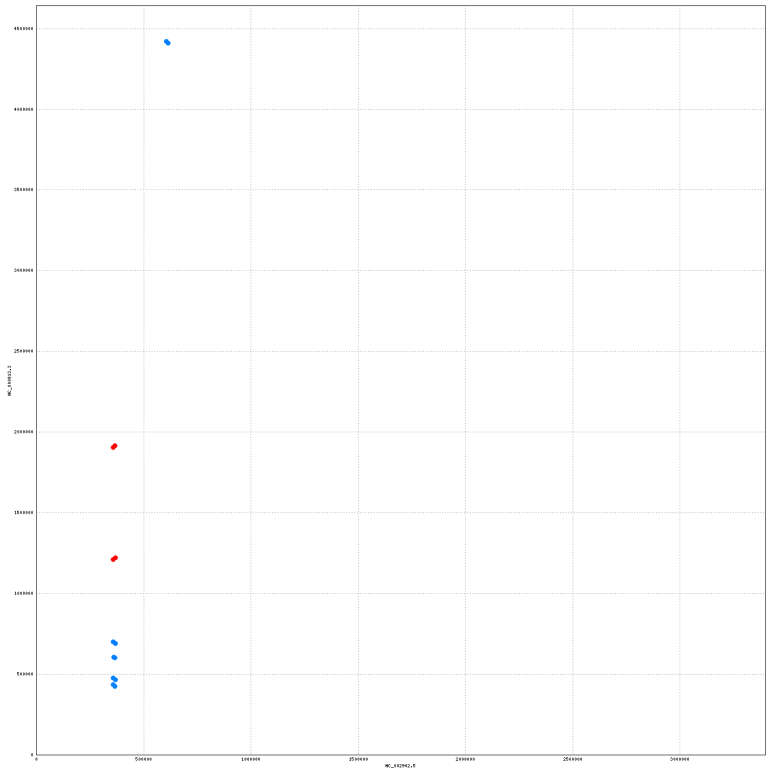
Slika 6.6: Implementacija rada: Usporedba genoma vrsta *Escherichia coli* i *Klebsiella pneumoniae*

Usporedba različitih genoma

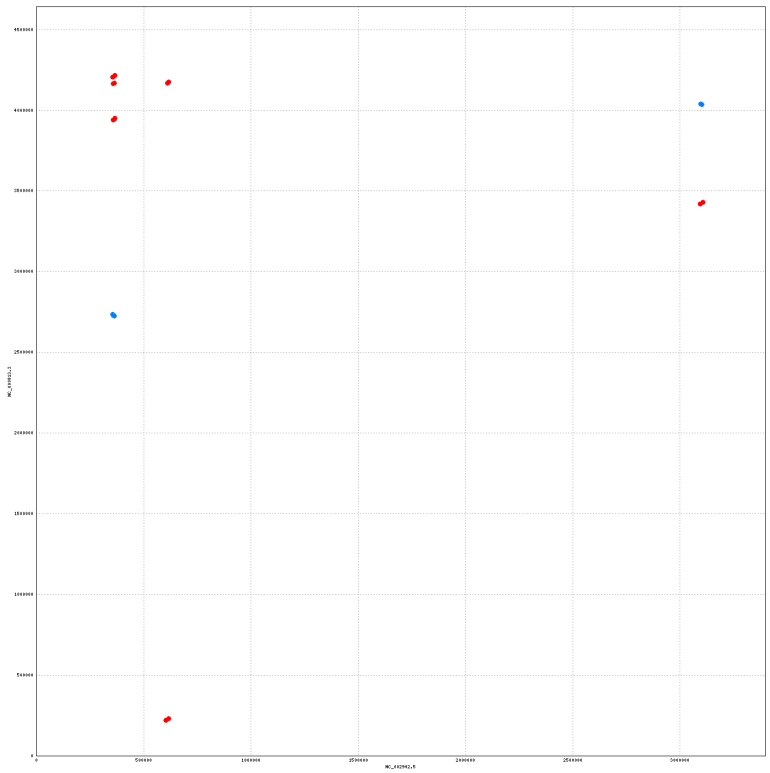
Preostalo je još usporediti ponašanje programa kada je riječ o genomima bez puno preklapanja. Taj je slučaj vidljiv is slika 6.7, 6.8 i 6.9. Na njima su prikazani rezultati usporedbe genoma vrste *Escherichia coli* s genomom vrste *Legionella pneumophila*. Iz slika je jasno kako svaki od programa ima podjednakih problema s pronalaskom poravnanja. Budući da genomi nisu slični, takvi su rezultati i očekivani.



Slika 6.7: Minimap: Usporedba genoma vrsta *Escherichia coli* i *Legionella pneumophila*



Slika 6.8: Mashmap: Usporedba genoma vrsta *Escherichia coli* i *Legionella pneumophila*



Slika 6.9: Implementacija rada: Usporedba genoma vrsta *Escherichia coli* i *Legionella pneumophila*

6.2.2. Računalna učinkovitost algoritama

U ovom su odjeljku istražene vremenska i memorijska učinkovitost algoritma. Svi su podaci dobiveni standardnim alatom na Linux sustavima, pokretanjem programa uz naredbu `/usr/bin/time -v`. Podaci o vremenima izvođenja i zauzeću memorije za sve ulazne podatke na kojima je promatrana točnost dani su tablicama 6.3, 6.4 i 6.5.

Program	Vrijeme izvođenja	Maksimalna utrošena memoija
MashMap	0.65 sekundi	34752 KB
Minimap	0.75 sekundi	102112 KB
Nova implementacija	624.17 sekundi	1044728 KB

Tablica 6.3: Podaci za usporedbu dva ista genoma vrste *Escherichia coli*

Program	Vrijeme izvođenja	Maksimalna utrošena memoija
MashMap	0.57 sekundi	35480 KB
Minimap	0.50 sekundi	70216 KB
Nova implementacija	103.67 sekundi	191392 KB

Tablica 6.4: Podaci za usporedbu genoma vrsta *Escherichia coli* i *Klebsiella pneumoniae*

Program	Vrijeme izvođenja	Maksimalna utrošena memoija
MashMap	0.49 sekundi	33580 KB
Minimap	0.47 sekundi	57452 KB
Nova implementacija	10.55 sekundi	64724 KB

Tablica 6.5: Podaci za usporedbu genoma vrsta *Escherichia coli* i *Legionella pneumophila*

Rezultati potvrđuju većinu naših pretpostavki. Iako za male duljine ulaznih podataka *Minimap* poravnanja pronađe brže nego *MashMap*, s porastom sličnosti nizova raste i učinkovitost programa *MashMap*. Kod unosa veličine ljudskog genoma, *MashMap* poravnanja pronađe znatno brže nego *Minimap* (Jain et al., 2018). Iako je nova

implementacija izrađena u sklopu ovog rada po točnosti istovjetna originalnoj implementaciji, ona se po performansama ipak ne da mjeriti s profesionalnim alatima kao što su *MashMap* i *Minimap*.

7. Zaključak

Kroz ovaj je rad predstavljen heuristički pristup problemu globalnog poravnanja temeljen na računanju koeficijenata sličnosti skupova k-torki. Učinkovitost većine korištenih tehnika dokazana je i formalno i empirijski. Važniji su dokazi ponovno izneseni i u ovom radu, a svi ostali dostupni su u literaturi. Iako je implementirani algoritam specijaliziran za poravnavanje cjelovitih genoma, uz određene modifikacije primjenjiv je i na niz drugih problema. Na primjer, prvi se dio algoritma može samostalno koristiti za traženje preklapanja dugih očitanja s referentnim nizom.

Predložena je i heuristika odbacivanja nevažnih i lažnih poravnanja koja se može primijeniti u nizu drugih algoritama. Uz heuristiku je predložen i prikladan algoritam filtriranja čija je optimalnost formalno dokazana. Algoritam je neovisan o korištenoj heuristici što otvara vrata njegovoj širokoj primjeni.

Konačno, kroz rezultate je demonstrirana učinkovitost algoritma te preciznost usporediva s točnijim i sporijim metodama globalnog poravnanja.

LITERATURA

A.Z. Broder. On the resemblance and containment of documents. 1997. URL https://dl.acm.org/citation.cfm?id=830043.

H. Fan, A.R. Ives, Y. Surgot-Groba, C.H. Cannon, i S. Aluru. An assembly and alignment-free method of phylogeny reconstruction from next-generation sequencing data. 2015. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4501066/.

P. Jaccard. Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines. 1901. URL https://www.researchgate.net/publication/243457811_Distribution_de_la_Flore_Alpine_dans_le_Bassin_des_Dranses_et_dans_quelques_regions_voisines.

C. Jain, A. Dilthey, S. Koren, A.M. Phillippy, i S. Aluru. A fast approximate algorithm for mapping long reads to large reference databases. 2017. URL https://www.biorxiv.org/content/early/2017/01/27/103812.full.pdf.

C. Jain, S. Koren, A. Dilthey, A.M. Phillippy, i S. Aluru. A fast adaptive algorithm for computing whole-genome homology maps. 2018. URL https://www.biorxiv.org/content/biorxiv/early/2018/02/18/259986.1.full.pdf.

S. Schleimer, D.S Wilkerson, i A. Aiken. Winnowing: Local algorithms for document fingerprinting. 2003. URL https://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf.

M. I. Shamos i D. Hoey. GEOMETRIC INTERSECTION PROBLEMS. 1976. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.366.9983&rep=rep1&type=pdf.

Približni algoritam za brzo računanje poravnanja dvaju dugačkih nizova

Sažetak

Globalno poravnanje genoma važan je problem genomike. Zbog duljine genoma, računanje optimalnog puta poravnanja točnim algoritmima dinamičkog programiranja računalno je neizvedivo. Cilj ovog rada predstavljanje je i implementacija heurističkog algoritma poravnanja temeljenog na tehnikama traženja predstavnika i procjene Jaccardovog koeficijenta sličnosti skiciranjem.

U poglavlju 2 dan je popis pretpostavki neophodnih za rad algoritma te popis pojmova za pomoć pri njegovom razumijevanju. Poglavlje 3 iznosi formalnu definiciju problema, a konkretan algoritam razvijen za njegovo rješavanje opisan je poglavljem 4. Implementacijski specifični detalji nevažni za opis algoritma na visokoj razini navedeni su u poglavlju 5. Rezultati i učinkovitost implementiranog algoritma istraženi su u poglavlju 6. Konačno, poglavlje 7 ukratko izdvaja najbitnije ideje rada.

Ključne riječi: MinHash, Jaccard, Winnowing, Predstavnik, Skiciranje, Poravnanja dugih očitanja

Approximate Algorithm for Computing Alignment Between Two Long Sequences

Abstract

Global sequence alignment is an important problem in the field of genomics. At the whole genome scale, computing the optimal alignment path by using an exact dynamic programming algorithm is computationally impossible. This paper looks to present and implement a heuristic alignment algorithm which relies on computing sequence minimizers and sketch-based estimations of the Jaccard index.

The assumptions and preliminaries necessary to understand the algorithm are given by chapter 2. Chapter 3 formally defines the studied problem, while chapter 4 presents and explains the proposed solution. Implementation details not directly connected to the algorithm itself are listed in chapter 5. Performance and results yielded by the implementation are explored in chapter 6. Chapter 7 provides a quick overview of the main ideas presented throughout the paper.

Keywords: MinHash, Jaccard, Winnowing, Minimizer, Sketching, long read mapping