SVEUČILISTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Bioinformatika - Projekt

Toni Kukurin

Zagreb, 2018.

Sadržaj

1. Uvod **1**

2. Algoritmi **2**

2.1. MinHash . . . 2

2.2. Winnowing . . . 2

2.3. Postupak . . . 2

2.3.1. Pretvaranje sekvence u k-mere . . . 3

2.3.2. Hashiranje pozicija . . . 3

2.3.3. Winnowing . . . 3

2.3.4. Usporedba . . . 3

3. Formalni opis problema **4**

4. Implementacija **5**

4.1. Korištene knjižnice . . . 5

4.1.1. Project Lombok . . . 5

4.1.2. Google Guava . . . 5

4.2. Postupak . . . 5

4.2.1. Winnowing . . . 5

4.2.2. Pronalazak kandidata . . . 6

4.2.3. Izračun aproksimacija Jaccardovih sličnosti . . . 6

4.3. Pomoćni programi . . . 7

4.4. Parametri . . . 7

5. Rezultati **8**

6. Zaključak **9**

Literatura **10**

1. Uvod

Zadatak je ovog projekta implementirati približni algoritam za mapiranje očitanja sekvenci na dugačke referentne genome. Potreba za postojanjem takvog algoritma uvjetovana je trećom generacijom uređaja za sekvenciranje (*Oxford Nanopores*, *Pacific Biosciences*), koji uz znatno veću duljinu očitanja također rade više pogrešaka. Projekt se temelji na radu Jain et al. [4]

Postojeći radovi izvan sfere bioinformatike koji koriste sličan pristup kao u [4] većinom ga upošljavaju za određivanje sličnosti dokumenata u svrhu otkrivanja plagijata i brze pretrage postojećih baza podataka. Kao primjer ranijih radova navodimo Brodera [1] i Schleimerov Winnowing algoritam [7], koji oboje čine bazu za razvoj postupka korištenog u projektu. Za razliku od prijašnjih radova gdje se samo empirijski dokazuje uspješnost, [4] također razvija matematički okvir pristupa. Objavljuju iznimno optimistične rezultate u prilog predstavljenog algoritma u pogledima točnosti i brzine rada.

2. Algoritmi

Algoritam predstavljen u [4] oslanja se na dvije postojeće metode aproksimacije sličnosti dokumenata, *MinHash* i *Winnowing*. U nastavku kratko opisujemo oba postupka, te dajemo primjer korištenja u algoritmu.

2.1. MinHash

MinHash je efikasan algoritam za računanje nepristrane procjene Jaccardove sličnosti dvaju dokumenata. Broder [1] dokazuje da je

$$\frac{|S(A \cup B_i) \cap S(A) \cup S(B_i)|}{|S(A \cup B_i)|}$$

nepristrana procjena $J(A, B_i)$, gdje J predstavlja Jaccardovu sličnost a A i B_i skupove elemenata. $S(x)$ predstavlja skup najmanjih s *hashiranih* elemenata iz skupa x.

2.2. Winnowing

Winnowing, za razliku od MinHash-a, ograničava najveći razmak izmedu bilo koja dva *hasha*, i to na način da koristi postupak klizećeg prozora, uvijek uzimajući najmanji *hash* iz svakog sljedećeg koraka (ukoliko dva *hasha* imaju istu vrijednost, uzima se onaj s većim indeksom).

Na taj način, ako veličinu prozora označimo s w, uvijek imamo najviše $w - 1$ elemenata između svaka dva otiska. Schleimer et al. [7] dokazuju da je očekivan broj takvih elemenata $\frac{2 \times |A_0|}{w}$ za neki nasumični slijed A_0.

2.3. Postupak

Kratak primjer postupka opisat ćemo na sekvenci $A = (A, A, C, T, C, G)$. Recimo da je veličina k-mera 3, te da je *hash* funkcija h nasumična metoda koja redom

proizvodi vrijednosti $(1, 3, 1, 1)$ neovisno o ulazu. Algoritam radi sljedeće korake.

2.3.1. Pretvaranje sekvence u k-mere

K-mere predstavljamo n-torkama: $K = \{(A, A, C), (A, C, T), (C, T, C), (T, C, G)\}$

2.3.2. Hashiranje pozicija

Svaku n-torku iz K zatim mapiramo na njenu *hash* vrijednost $h(x)$, dajući rezultat $H = \{1, 3, 1, 1\}$.

2.3.3. Winnowing

Postupak *winnowinga* provodimo uz parametar $w = 2$. Klizeći prozor redom razmatra n-torke hasheva $\{(1, 3), (3, 1), (1, 1)\}$, te uz ranije objašnjen poredak stvara vrijednosti *minimizatora* $W(A) = \{(0, 1), (2, 1), (3, 1)\}$. Prvi element minimizatora označava indeks na kojem se *hash* nalazi (uz 0-indeksiranje), a drugi samu vrijednost *hasha*.

2.3.4. Usporedba

Koristeći ranije opisani *MinHash* postupak, sada možemo uz znatno manji memorijski otisak napraviti procjenu Jaccardove sličnosti između dva dokumenta. Izrada *sketcha* S nakon što napravimo *winnow* vrijednosti više nije jednostavni nasumični uzorak, no rad [4] empirijski pokazuje da usprkos tome daje jednako dobru procjenu kao originalni *MinHash* postupak. *Winnowed-minhash* procjena stoga je definirana kao:

$$J'(A, B_i) = \frac{|S(W(A) \cup W(B_i)) \cap S(W(A)) \cup S(W(B_i))|}{|S(W(A) \cup W(B_i))|}$$

uz:

$$S(W(A)) = min_s\{h : (h, pos) \in W(A)\}$$

3. Formalni opis problema

Uz sekvencu A i najveću dozvoljenu pogrešku ϵ_{max}, zadatak je pronaći ciljne pozicije u referenci B gdje je greška po bazi manja od ϵ_{max}. Uz algoritam predstavljen u prijašnjoj sekciji možemo efikasno izračunati procjenu Jaccardove sličnosti J, pa se ta vrijednost i koristi kao predstavnik (*proxy*) za traženi ϵ_{max}. Želimo da vrijedi:

$$J(A, B_i) \geq \tau,$$
$$\tau = G(\epsilon_{max}, k) - \delta$$

G predstavlja samo očekivanu vrijednost Jaccardove sličnosti, pa stoga od desne strane nejednakosti oduzimamo neku malu vrijednost δ. G se računa po sljedećoj formuli, koja proizlazi iz pretpostavljenog *Poissonovog* modela distribucije pogrešaka:

$$G(\epsilon_{max}, k) = \frac{1}{2 \times e^{k\epsilon_{max}} - 1}$$

Vrijednost ϵ_{max} proizvoljno je odabrana vrijednost najveće dozvoljene greške po bazi (*per-base error rate*), a parametar k predstavlja veličinu k-mera u postupku algoritma.

4. Implementacija

Program je implementiran u Javi, uz podjelu važnih koraka u zasebne klase. *Unit* testovi pokrivaju neke osnovnije slučajeve; nipošto nisu iscrpni, no pokazali su se vrlo korisnima kao razumna provjera funkcionalnosti.

4.1. Korištene knjižnice

4.1.1. Project Lombok

Kako bi smanjili broj linija kôda uvjetovanih (većinom) Javinom sintaksom, korišten je Project Lombok [5] koji uz pomoć anotacija automatski kreira često korištene programske predloške (*getter* i *setter* funkcije; oblikovni obrazac *builder*; metode *equals* i *hash code*; itd).

4.1.2. Google Guava

Googleova Guava [3] poznata je knjižnica otvorenog kôda pomoćnih metoda za Javu. U implementaciji su konkretno korištene pomoćne metode za *hashiranje* (*Murmur hash*) te *multimap* za jednostavniju izradu inverza indeksa *hasheva* (ranije spomenuta *hash* mapa H).

4.2. Postupak

Algoritam koji su autori koristili opisan je u radu [4] te je općenita ideja iza tog algoritma baza za implementaciju našeg postupka. Slijedi kratak opis.

4.2.1. Winnowing

Najprije izračunamo $W(B)$, *winnow* referentnog čitanja. Na taj način možemo efikasno pronaći $W(B_i)$ (*winnow* iz reference B koji počinje na indeksu i) za

proizvoljni i. *Winnow* spremamo kao sortiranu listu parova $(indeks, hash)$, a prema ranijoj primjedbi iz [7], očekivano memorijsko zauzeće čitave strukture podataka jest $\frac{2\times|B|}{w}$.

Dodatno, inverz sekvence (mapiranje s *hash* vrijednosti na listu indeksa) spremamo u mapu H radi efikasnog dohvaćanja potrebnog u prvom koraku algoritma. Za upite nam ova mapa nije potrebna, te za njih samo računamo *winnow* vrijednosti.

Kôd koji računa ove vrijednosti nalazi se u *FastaKmerBufferedReader* i *Minimizer* klasama. Prva klasa čita k-mere iz datoteka *streamingom*, a druga izrađuje *winnow* tehnikom klizećeg prozora.

4.2.2. Pronalazak kandidata

Nakon što izradi *minimizatore*, algoritam radi u dva koraka: najprije izračuna sve potencijalne kandidate u B, dobivajući za rezultat rangove za koje vrijedi željena Jaccardova sličnost (veća od τ). To napravi preko *hash* mape H i poznatih očitanih *hasheva* k-mera upitne sekvence A.

U prvom koraku algoritma prolazimo kroz jedinstvene *hashirane* vrijednosti minimizatora upita, preko mape H provjeravamo sadrži li referenca traženu vrijednost te, ukoliko da, sve pozicije iz reference B na kojima je taj *hash* minimizator, dodajemo u listu L. Listu zatim sortiramo te iz takve strukture jednostavno možemo izvući *kandidatne regije* gdje se barem m *hasheva* podudara između referentne i upitne sekvence.

ReadMapper sadrži metodu *collectCandidateRegions*, unutar koje se izvršavaju koraci opisani u ovoj sekciji. Povratna vrijednost metode lista je *kandidatnih regija*.

4.2.3. Izračun aproksimacija Jaccardovih sličnosti

Iz prethodnog koraka algoritma dobivamo regije za koje smatramo da sadrže tražene vrijednosti. Za svaku regiju pronađemo *minimizatore* te u rezultatnu listu spremimo indeks kada je *winnowed-minhash* procjena Jaccardove sličnosti barem τ.

Iz *winnowed-minhasha*, procjenu za ϵ možemo jednostavno dobiti koristeći:

$$\epsilon = F(J,k) = \frac{1}{k} \times \log(\frac{2J}{1+J})$$

gdje je k veličina k-mera, a J *winnowed-minhash* procjena Jaccardove sličnosti. Konačan rezultat kao izlaz programa jest procjena identiteta, to jest vrijednost $(1 - \epsilon)$ te pripadni indeks na kojem je ta vrijednost očitana. Ove se vrijednosti računaju u klasi *ReadMapper*, metoda *findMostLikelyMatch*. Kao pomoćna mapa koja u logaritamskoj strukturi (AVL stablo) pamti brojnik za *winnowed-minhash* procjenu koristi se klasa *SketchMap*. Metoda *getSharedMinimizers* vraća broj dijeljenih minimizatora između upita i referentne sekvence.

4.3. Pomoćni programi

Za izradu testnih podataka skinut je referentni slijed DNK-a bakterije *Clostridium Cellulosi* s *web* stranice [2]. Program WGSIM [8] korišten je za izradu upita, to jest za simulaciju podslijedova iz DNK-a referentne bakterije. U direktoriju *helpers* nalazi se *bash* program *simulate.sh* koji je korišten za izradu upita. Na svakom mjestu koje to dozvoljava, iz očitih razloga vrijednost 42 je korištena kao *seed* za generator nasumičnih brojeva.

S obzirom da je izlaz WGSIM programa FQ datoteka, direktorij *helpers* također sadrži kratku *bash* naredbu koja pretvara FQ format u FASTA datoteku.

4.4. Parametri

Korišteni su parametri $\epsilon = 0.15$ (identitet od 85%), veličina k-mera od 16 baza (dakle *hashiramo* po 16 baza odjednom), te veličina prozora $w = 90$. Uz te parametre, očekivana vrijednost Jaccardove sličnosti G je oko 0.0475. Kao δ uzimamo 1.5%, tj. 0.015 (empirijski odabrano).

5. Rezultati

Testirali smo algoritam na 10 različitih upita generiranih iz bakterije *Clostridium Cellulosi* putem programa WGSIM (korištena je skripta *helpers/simulate.sh* za generiranje slijedova). Sva su mjerenja (3 pokretanja) obavljena na računalu s 16 GB RAM memorije i Intelovim i7 CPU-om takta 2.2 Ghz, a ukupan broj upita jest 50 po datoteci. Stupac *pronađenih položaja* označava koliko je od tih 50 upita točno mapirano na referencu (u zagradi se nalazi broj upita pronađenih korištenjem originalne implementacije algoritma [6]).

ID upita	Trajanje	Prosječan utrošak memorije	Pronađenih položaja
0	1.19 s	64.5 MB	32 (50)
1	1.03 s	7.56 MB	32 (50)
2	1.12 s	103.5 MB	25 (50)
3	1.26 s	95.5 MB	27 (50)
4	1.52 s	90.5 MB	24 (49)
5	2.14 s	136.67 MB	18 (47)
6	2.61 s	110.3 MB	28 (25)
7	2.66 s	268.5 MB	1 (5)
8	2.33 s	78.5 MB	0 (1)
9	2.94 s	65.0 MB	0 (2)

Prosječno vrijeme po upitu je 1.88 sekundi, uz oko 0.9 sekundi potrebnih za čitanje i minimizaciju referentne sekvence. S druge strane, trenutna implementacija *MashMapa* u prosjeku treba oko 0.23 sekunde za mapiranje istih sekvenci, uz oko 0.1 sekundu utrošenu na referentnu sekvencu, što je iznimno poražavajuć rezultat u okviru projekta. Algoritamski implementacija *ne bi trebala* biti toliko sporija, pa se nadamo da je izbor programskog jezika usko grlo u ovom slučaju.

6. Zaključak

Predstavljeni algoritam zanimljivo pristupa problemu određivanja položaja dugih slijedova DNK iz referentnih baza podataka. Iako za poravnavanje slijedova koristi *proxy* Jaccardove sličnosti, dobiveni rezultati pokazuju uspješnost sumjerljivu postojećim metodama, uz daleko bolju skalabilnost i manje memorijsko zauzeće. S obzirom da moderne tehnologije sekvenciranja dozvoljavaju čitanje sve duljih slijedova genoma uz rastuću propunost, *mashmap* predstavlja važan korak pri parsiranju tih slijedova u realnom vremenu.

Naša implementacija nažalost ne uspijeva biti jednako uspješna kao što je ona predstavljena u originalnom radu, no za jednostavnije slučajeve pronalazi odgovarajuća rješenja u razumnom vremenu. Potencijalno bi interesantno bilo u budućnosti isprobati pokrenuti isti program uz primitivnije strukture podataka (puno niži nivo apstrakcije) te usporediti tako dobivene rezultate s onima dobivenim u ovom projektu.

Literatura

[1] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[2] Clostridium Cellulosi. `http://bacteria.ensembl.org/_clostridium_cellulosi/Info/Index`.

[3] Guava. `https://github.com/google/guava`.

[4] Chirag Jain, Alexander Dilthey, Sergey Koren, Srinivas Aluru, and Adam M Phillippy. A fast approximate algorithm for mapping long reads to large reference databases. In *International Conference on Research in Computational Molecular Biology*, pages 66–81. Springer, 2017.

[5] Project Lombok. `https://projectlombok.org/`.

[6] MashMap. `https://github.com/marbl/MashMap`.

[7] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85. ACM, 2003.

[8] WGSIM. `https://github.com/lh3/wgsim`.