SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4913

Algoritam za određivanje ukupnog poravnanja dva grafa poravnanja parcijalnog uređaja

Mislav Bradač

Zagreb, lipanj 2017.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Kako biste uklonili ovu stranicu, obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Pojavom modernih računala razvila se bioinformatika, grana znanosti koja pomoću algoritama pokušava ubrzati i automatizirati rješavanje temeljnih problema biologije i genetike za koje je klasični pristup rješavanja prespor ili preskup.

U središtu genetike, a tako i bioinformatike je DNK lanac kao temeljni nositelj informacija živih bića. U računalu DNK lanac možemo predstaviti kao niz znakova gdje jedan znak predstavlja jednu aminokiselinu, odnosno jedan protein. Tako zapisan dio DNK lanca zovemo slijed. Poravnanje, tj. ocjenjivanja sličnosti više slijedova je zajednički podproblem mnogih drugih problema bioinformatike kao što je sastavljanje DNK lanca, pronalazak konsenzusa, pronalazak srodnih vrsta, detekcija mutacija...

Algoritme za ocjenjivanje sličnosti slijedova dijelimo na heurističke koji jako brzo procjenjuju sličnost dvaju slijedova i analitičke koji su sporiji, ali točniji te uz samu ocjenu sličnosti računaju i točno poravnanje slijedova. U ovom radu će biti razmatrani samo analitički algoritmi.

Najkorišteniji analitički algoritam za poravnanje dvaju slijedova je Smith-Waterman (te njemu srodni algoritmi) koji u složenosti $\mathcal{O}(nm)$, gdje je n duljina jednog, a m drugog slijeda, pronalazi njihovu sličnosti i poravnanje. Algoritam je moguće proširiti i njime poravnati veći broj slijedova, no u praksi takvo proširenje nije primjenjivo zbog eksponencijalnog rasta složenosti s porastom broja slijedova. Unatoč svojoj velikoj složenosti u praksi je često korišten za poravnanje dvaju slijedova te kao dio mnogih drugih algoritama.

Jedan od algoritama za poravnanje slijedova temeljenih na Smith-Waterman algoritmu je i generiranje grafa poravnanja parcijalnog uređeja (POA grafa). Iz poravnanja pronađenog pomoću Smith-Waterman algoritma možemo izgraditi graf koji prikazuje više mogućih optimalnih poravnanja dvaju slijedova. Na taj graf možemo poravnati nove slijedove i njime uspješno prikazati poravnanje većeg broja slijedova.

POA graf je veoma važna struktura jer uz smanjenje složenosti poravnjanja više slijedova daje poravnanja velike točnosti te omogućava daljnu analizu slijedova kao što je generiranje konsenzusa. U ovom radu će biti izložen algoritam za spajanje dvaju već izgrađenih POA grafova kojim se može dodatno ubrzati izgradnja POA grafa većeg broja slijedova uz određeni pad kvalitete poravnanja.

2. Pregled područja

U ovom poglavlju će biti opisani već postojeći algoritmi koji su korišteni kao dijelovi algoritma za spajanje dvaju izgrađenih POA grafova. S obzirom da algoritam koristi mnoge temeljne algoritme u području analitičkog poravnanja slijedova ovo poglavlja služi i kao sam pregled spomenutog područja.

2.1. Levenshteinova udaljenost

Udaljenost dvaju slijedova definiramo kao broj operacija koje je potrebno učiniti nad jednim slijedom da bi on postao jednak drugom slijedu. Alternativno u nekim algoritmima svakoj operaciji ćemo pridružiti težinu te ćemo udaljenost/sličnost definirati kao minimalnu/maksimalnu sumu težina operacija potrebnih da bi jedan slijed pretvorili u drugi. Poravnanje slijedova definiramo kao niz učinjenih operacija pri toj pretvorbi. Jedan od mogućih skupova operacija je:

- brisanje znaka iz slijeda
- umetanje znaka u slijed
- zamjena znaka drugim znakom

Levensteinova udaljenost je upravo minimalan broj operacija kojih moramo načiniti iz gornjeg skupa operacija da bi smo pretvorili jedan slijed u drugi. Formalno ju možemo definirati kao vrijednost funkcije $f(i,j)$ za parametre $i \leftarrow n, j \leftarrow m$ gdje je n duljina niza a, a m niza b. Funkciju f možemo rekurzivno definirati:

$$f(i,j) = \begin{cases} min \begin{cases} f(i-1,j)+1 \\ f(i,j-1)+1 \\ f(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & i > 0 \wedge j > 0 \\ 0 & i = 0 \wedge j = 0 \\ \infty & \text{inače} \end{cases}$$

Za primjer kažimo da je Levenshteinova udaljenosti riječi *svemirci* i *nemiri* jednaka 3 (izbacivanje slova *s* iz riječi *svemirci*, promjena slova *v* u slovo *n*, izbacivanje slova *c*).

Levenshteinova udaljenost se može izračunati u složenosti $\mathcal{O}(nm)$ dinamičkim programiranjem.

2.2. Needleman-Wunsch

Znanstvenici Saul B. Needleman i Christian D. Wunsch su 1970. godine uveli malo drugačiji skup dozvoljenih operacija te svakoj operaciji pridijelili težinu (1):

- izbacivanje ili ubacivanje znaka ima negativnu težinu d
- težina zamjene, tj. podudaranja dvaju znakova definirana je funkcijom $S(x, y)$

Taj skup operacija se pokazao točnijim u poravnanju slijedova jer bolje opisuje pojave koje se događaju DNA-u lancu u stvarnosti. Formalno po Needleman-Wunsch algoritmu je sličnost slijedova a i b, duljina n i m jednaka $f(n, m)$ gdje je funkcija f definirana kao:

$$f(i,j) = \begin{cases} max \begin{cases} f(i-1,j) + d \\ f(i,j-1) + d \\ f(i-1,j-1) + S(a_i, b_j) \end{cases} & i > 0 \wedge j > 0 \\ 0 & i = 0 \wedge j = 0 \\ -\infty & \text{inače} \end{cases}$$

Vrijednosti funkcije S koje se koriste kao težine podudaranja/zamjene znakova su izračunate za svaki par aminokiselina, odnosno proteina temeljene na eksperimentalnim mjerenjima. Tako dobivene vrijednosti su zapisane u razne matrice (PAM, BLOSUM) te se ovisno o primjeni algoritma izabire matrica i vrijednost konstante d koja se koristi pri izračunu poravnanja.
Složenost izračuna sličnosti po Needleman-Wunsch algoritmu je jednaka složenosti izračuna Levenshteinove udaljenosti: $\mathcal{O}(nm)$.

2.3. Funkcije praznine

Michael S. Waterman je 1976. dodatno prilagodio vredovnanje operacija nad slijedovima vjerojatnostima transformacija DNK lanca u stvarnom svijetu uvodeći funkcije praznina (2). Naime, puno je veća vjerojatnost da će iz jednog DNK lanca nestati (zbog mutacije ili raznih vanjskih faktora) n uzastopnih aminokiselina, nego n nepovezanih aminokiselina. Zato je Waterman predložio skup operacija:

- umetanje k znakova u slijed ima težinu $w(k)$
- brisanje l znakova iz slijeda ima težinu $w(l)$

Tablica 2.1: Matrica BLOSUM50 korištena kao funkcija S u algoritmima poravnanja

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-2	-2	0	-1	-1	0
R	-2	7	0	-1	-3	1	0	-2	0	-3	-2	3	-1	-2	-2	-1	-1	-2	-1	-2	-1	0	-1
N	-1	0	6	2	-2	0	0	0	1	-2	-3	0	-2	-2	-2	1	0	-4	-2	-3	4	0	-1
D	-2	-1	2	7	-3	0	2	-1	0	-4	-3	0	-3	-4	-1	0	-1	-4	-2	-3	5	1	-1
C	-1	-3	-2	-3	12	-3	-3	-3	-3	-3	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1	-2	-3	-2
Q	-1	1	0	0	-3	6	2	-2	1	-2	-2	1	0	-4	-1	0	-1	-2	-1	-3	0	4	-1
E	-1	0	0	2	-3	2	6	-2	0	-3	-2	1	-2	-3	0	0	-1	-3	-2	-3	1	4	-1
G	0	-2	0	-1	-3	-2	-2	7	-2	-4	-3	-2	-2	-3	-2	0	-2	-2	-3	-3	-1	-2	-1
H	-2	0	1	0	-3	1	0	-2	10	-3	-2	-1	0	-2	-2	-1	-2	-3	2	-3	0	0	-1
I	-1	-3	-2	-4	-3	-2	-3	-4	-3	5	2	-3	2	0	-2	-2	-1	-2	0	3	-3	-3	-1
L	-1	-2	-3	-3	-2	-2	-2	-3	-2	2	5	-3	2	1	-3	-3	-1	-2	0	1	-3	-2	-1
K	-1	3	0	0	-3	1	1	-2	-1	-3	-3	5	-1	-3	-1	-1	-1	-2	-1	-2	0	1	-1
M	-1	-1	-2	-3	-2	0	-2	-2	0	2	2	-1	6	0	-2	-2	-1	-2	0	1	-2	-1	-1
F	-2	-2	-2	-4	-2	-4	-3	-3	-2	0	1	-3	0	8	-3	-2	-1	1	3	0	-3	-3	-1
P	-1	-2	-2	-1	-4	-1	0	-2	-2	-2	-3	-1	-2	-3	9	-1	-1	-3	-3	-3	-2	-1	-1
S	1	-1	1	0	-1	0	0	0	-1	-2	-3	-1	-2	-2	-1	4	2	-4	-2	-1	0	0	0
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-1	-1	2	5	-3	-1	0	0	-1	0
W	-2	-2	-4	-4	-5	-2	-3	-2	-3	-2	-2	-2	-2	1	-3	-4	-3	15	3	-3	-4	-2	-2
Y	-2	-1	-2	-2	-3	-1	-2	-3	2	0	0	-1	0	3	-3	-2	-1	3	8	-1	-2	-2	-1
V	0	-2	-3	-3	-1	-3	-3	-3	-3	3	1	-2	1	0	-3	-1	0	-3	-1	5	-3	-3	-1
B	-1	-1	4	5	-2	0	1	-1	0	-3	-3	0	-2	-3	-2	0	0	-4	-2	-3	4	2	-1
Z	-1	0	0	1	-3	4	4	-2	0	-3	-2	1	-1	-3	-1	0	-1	-2	-2	-3	2	4	-1
X	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-2	-1	-1	-1	-1	-1

– podudaranje/zamjena znakova ima težinu $S(x, y)$

S tim skupom operacija je sličnost slijedova definirana funkcijom:

$$f(i,j) = \begin{cases} max \begin{cases} f(i-k,j) + w(k) \\ f(i,j-l) + w(l) \\ f(i-1,j-1) + S(a_i, b_j) \end{cases} & i > 0 \wedge j > 0 \wedge 0 < k \leq i \wedge 0 < l \leq j \\ 0 & i = 0 \wedge j = 0 \\ \infty & \text{inače} \end{cases}$$

U praksi su se najbolje pokazale jednostavne aproksimacije funkcije praznine koje je predložio Gotoh kao što je $w(k) = o + (k-1) \cdot e, k > 0$ za neke odabrane negativne konstante o i e, gdje o predstavlja težinu započinjanja umetanja/brisanja slijeda znakova, a e težinu nastavka umetanja/brisanja (3). Složenost izračuna sličnosti s proizvoljnom funkcijom težina je $\mathcal{O}(nm \cdot max(n,m))$, odnosno općenito $\mathcal{O}(n^3)$, a s već spomenutom aproksimacijom vremenska složenost je $\mathcal{O}(nm)$.

Zbog jednostavnosti će se u ovom radu koristiti težine i operacije kao što su definirane u Needleman-Wunsch algoritmu, no bitno je napomenuti da prelazak na ovako definirane operacije značajnije ne komplicira implementaciju niti jednog od opisanih algoritama poravnanja.

2.4. Smith-Waterman

Uz Needleman-Wunsch algoritam važnu ulogu u poravnanju slijedova ima i algoritam Smith-Waterman (4). Za razliku od Needleman-Wunsch algoritma koji računa sličnost cijelih slijedova, Smith-Waterman algoritam pronalazi težinu najsličnijih uzastopnih podslijedova dvaju slijedovova. Iako zvuči puno složenije zapravo je Smith-Waterman trivijalna transformacija Needleman-Wunsch algoritma. Naime, poravnanje dvaju slijedova a i b, duljina n i m po Smith-Waterman algoritmu ima sličnost jednaku $max_{i,j} f(i,j), 0 < i \leq n \wedge 0 < j \leq m$. Gdje je funkcija f definirana kao:

$$f(i,j) = \begin{cases} max \begin{cases} 0 \\ f(i-1,j) + d \\ f(i,j-1) + 1 \\ f(i-1,j-1) + S(a_i, b_j) \end{cases} & i > 0 \wedge j > 0 \\ 0 & i = 0 \wedge j = 0 \\ -\infty & \text{inače} \end{cases}$$

Tablica 2.2: Levenshteinova funkcija za slijedove **AGGCCTC** i **ACCGGTA**

		1	2	4	4	5	6	7
		A	C	C	G	G	T	A
1	A	**0**	1	2	3	4	5	6
2	G	**1**	1	2	2	3	4	5
3	G	**2**	2	2	2	2	3	4
4	C	3	**2**	2	3	3	3	4
5	C	4	3	**2**	**3**	**4**	4	4
6	T	5	4	3	3	4	**4**	5
7	C	6	5	4	4	4	5	**5**

S obzirom da algoritam uzima najveću vrijednost funkcije, time dopušta da poravnanje završi na bilo kojim pozicijama u slijedovima. Uzimanjem i vrijednosti 0 kao jedan od mogućih prijelaza nikad ne dopušta da težina padne na negativnu vrijednost, tj. dopušta da poravnanje započne na bilo kojoj poziciji u slijedu. Iz tog razloga Smith-Waterman još nazivamo i algoritmom lokalnog poravnanja, a Needleman-Wunsch algoritam globalnog poravnanja.

Primijetimo da možemo lako definirati i poluglobalna poravnanja tako da primijenimo samo jednu od dvije spomenute novosti Smith-Waterman algoritma. Ako sličnost definiramo kao najveću vrijednost funkcije, a koristimo funkciju iz Needleman-Wunsch algoritma fiksirali smo početak poravnanja na početke slijedova, no dopustili smo mu da završi bilo gdje. Ako uzmemo funkciju iz Smith-Waterman algoritma, a uzimamo vrijednost funkcije $f(n, m)$ kao sličnost poravnanja dopustili smo početak na bilo kojoj poziciji, ali smo fiksirali završetak poravnanja na kraj slijedova. Mogučnost računanja poravnavanja na sva četiri načina će nam biti od velike važnosti u spajanju dva POA grafa.

2.5. Rekonstrukcija poravnanja

Uzmimo za primjer Levenshteinov algoritam za računanje udaljenosti na slijedovima **AGGCCTC** i **ACCGGTA**. Tada funkcija f poprima vrijednosti kao što je prikazano u tablici 2.2.

Kako bismo iz tablice pronašli poravnanje slijedova potrebno je krenuti iz završne vrijednosti, u slučaju Levenshteinove udaljenost od vrijednosti $f(n, m)$ te pronaći put kojim smo došli do te vrijednosti. To radimo tako da nađemo susjeda iz rekurzivne relacije čija vrijednost pribrojena s težinom odgovarajuće operacija je jednaka završnoj vrijednosti. Postupak ponavljamo dok ne dođemo do početka oba slijeda. Potrebno je primijetiti da je moguće da u nekom koraku postoji više prikladnih susjeda te u tom slučaju postoji više jednako dobrih

Tablica 2.3: Poravnanja iz tablice 2.2

1	2	4	4	5	6	7	8	9
A	-	-	G	G	G	G	T	C
A	C	C	G	G	-	-	T	A

1	2	4	4	5	6	7	8	9
A	G	G	C	C	-	-	T	C
A	-	-	C	C	G	G	T	A

poravnanja dvaju slijedova. U tablici 2.2 podcrtane vrijednosti predstavljaju jedno poravnanje, a *boldane* vrijednosti drugo. Standardni prikaz tih poravnanja je prikazan u tablicima 2.3 pri čemu znak "-" označava umetanje znaka iz drugog slijeda.

2.6. POA graf

Kao što smo vidjeli u prethodnom poglavlju često ne postoji jedinstveno poravnanje dvaju slijedova. U slučaju da nas zanima poravnanje samo dvaju slijedova, možemo izabrati proizvoljno poravnanje i njega proglasiti najboljim, no ako želimo poravnati još neki slijed teško je predvidjeti s kojim poravnanjem dvaju slijedova ćemo poravnavati treći slijed kako bi dobili najbolje ukupno poravnanje. S obzirom da je razmatranje svakog mogućeg poravnanja preskupo, želimo ili smanjiti broj poravnanja ili izgraditi strukturu koja predstavlja veći broj poravnanja te nju koristiti za daljnje poravnanje. POA graf je struktura kojom možemo prikazati više mogućih poravnanja (no ne sva) te je efikasna za izračun poravnanja daljnjih slijedova (5).

Uzmimo za primjer slijedove **ACCCAAC** i **ATTTAAG** te izračunajmo njihovu sličnost algoritmom Needleman-Wunscha s težinom 1 za poravnati znak, težinom -5 za zamjenu znaka, a težinom -1 za umetanja/brisanje znaka. Tada su neka od mnogih mogućih poravnanja prikazana tablicom 2.4. Iz poravnanja gradimo graf tako da svakom znaku iz slijedova pridijelimo jedan čvor graf. Poravnatim znakovima pridjeljujemo zajednički čvor. Svaki čvor grafa spajamo usmjerenom vezom sa čvorom koji je pridjeljen susjednom znaku slijeda. Poravnanju ranije spomenutih slijedova odgovara graf sa slike 2.1. Taj graf osim što prikazuje sva poravnanja iz tablice 2.4 uspješno prikazuje i mnoga druga najbolje poravnanja.

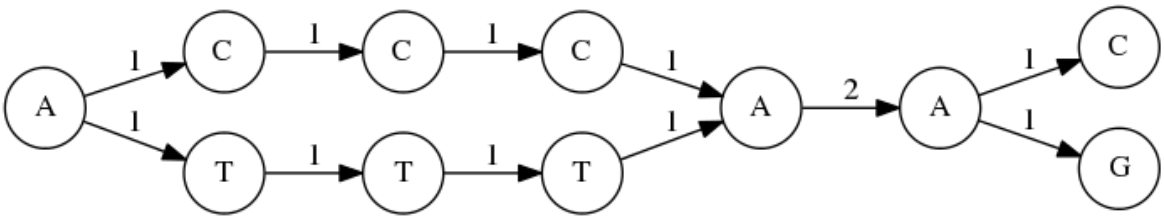
Dodatno svakoj vezi grafa možemo pridijeliti težinu jednaku broju slijedova koji dijele oba čvora pripadna toj vezi. Težine će biti bitne u kasnije opisanom algoritmu pronalaska konsenzusa iz POA grafa.

Nakon što smo izgradili POA graf možemo koristiti modificiranu funkciju za izračun sličnosti grafa i slijeda. Neka je a slijed, a b topološki sortirani niz čvorova grafa, te n

Tablica 2.4: Poravnanja slijedova **ACCCAAC** i **ATTTAAG**

1	2	4	4	5	6	7	8	9	10	11
A	C	C	C	-	-	-	A	A	-	C
A	-	-	-	T	T	T	A	A	G	-
1	2	4	4	5	6	7	8	9	10	11
A	C	-	-	C	C	-	A	A	C	-
A	-	T	T	-	-	T	A	A	-	G
1	2	4	4	5	6	7	8	9	10	11
A	C	C	-	C	-	-	A	A	C	-
A	-	-	T	-	T	T	A	A	-	G

Slika 2.1: POA graf poravnanja slijedova **ACCCAAC** i **ATTTAAG**



duljina slijeda, a m broj čvorova grafa. $prethodnik(x)$ je skup svi čvorova koji imaju vezu koji počinje u njima, a završavaju u čvoru x, a $znak(x)$ je znak koji pripada čvoru x. Tada modificirana funkcija Needleman-Wunscha glasi:

$$f(i,j) = \begin{cases} max \begin{cases} f(i-1,j) + d & \\ f(i,k) + d & \forall k \in prethodnik(b_j) \\ f(i-1,k) + S(a_i, znak(b_j)) & \forall k \in prethodnik(b_j) \end{cases} & i > 0 \wedge j > 0 \\ 0 & i = 0 \wedge j = 0 \\ -\infty & \text{inače} \end{cases}$$

Analogno možemo modificirati i funkciju za izračun Levenstheinove udaljenosti, odnosno funkciju Smith-Waterman algoritma. Koristeći tako modificiranu funkciju pronalazimo poravnanje između grafa i slijeda, te na sami graf dodajemo nove veze i čvorove po potrebi. Takav algoritam ponavljamo dok ne poravnamo sve slijedove.

Kao što je već spomenuto jedan POA graf ne može predstaviti sva moguća poravnanja za neke slijedove. Jedan od primjera je poravnanje iz prethodnog poglavlja te u tom slučaju izabiremo neki od mogućih POA grafova te nastavljamo algoritam s njime.

2.7. Pronalazak konsenzusa iz POA grafa

Jedna od primjena POA grafa je pronalazak konsenzusa (6). Neka smo očitali DNA sekvencu neke jedinke n puta. U idealnom slučaju svaki od tako dobivenih slijedova bi bio jednak, no u stvarnosti zbog mutacija i grešaka u očitavanju slijedovi se razlikuju. Konsenzus je slijed koji iz očitanja ima najveću vjerojatnost da prikazuje stvarnu DNA sekvencu.
Algoritam za izgradnju konsenzusa počinje slijedećim koracima:

- izgradi POA grafa iz očitanih slijedova
- topološki sortiraj čvorove POA grafa
- svakom čvoru postavi težinu na vrijednost 0
- za svaki čvor iz sortiranog niza:
    - izaberi ulaznu vezu najveće težine (u slučaju jednakih težina onu vezu koja dolazi iz čvora veće težine)
    - čvoru pridjeli težinu jednaku sumi težina izabrane veze i početnog čvora te veze
- pronađi čvor najveće težine, neka taj čvor ima oznaku i

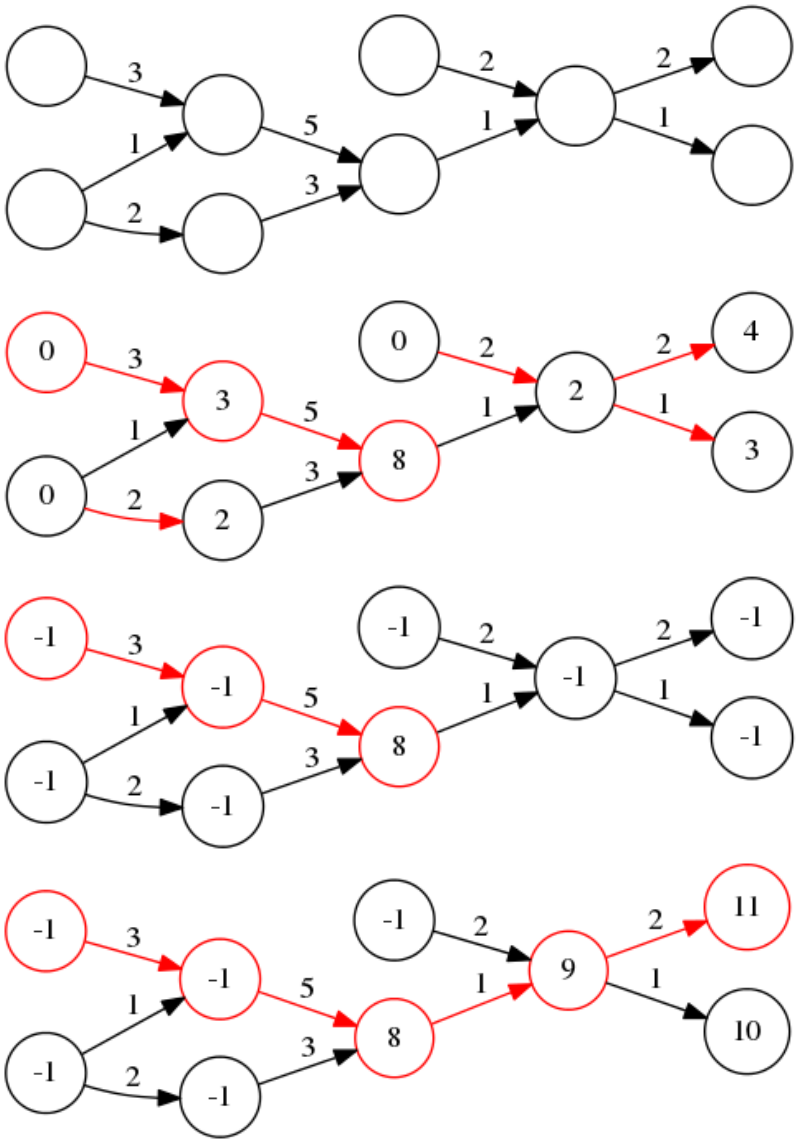
Tim koracima smo pronašli jedan čvor iz konsenzusa. Ostale čvorove pronalazimo tako da iz čvora i vezama odabranim već prije u algoritmu hodamo sve do čvora koji nema ulaznu vezu. Čvorovi posjećeni u toj šetnji pripadaju konsenzusu.
U određenim slučajima se može dogoditi da čvor najveće težine ima izlazne veze. Tada ponavljamo slijedeće korake sve dok nismo došli do čvora bez izlaznih veza:

- svim čvorovima osim čvora i postavi težinu na -1
- za svaki čvor iz sortiranog niza:
    - izaberi ulaznu vezu najveće težine **tako da početni čvor te veze nema negativnu težinu** (u slučaju jednakih težina onu vezu koja dolazi iz čvora veće težine)
    - čvoru pridjeli težinu jednaku sumi težina izabrane veze i početnog čvora te veze
- pronađi čvor najveće težine, neka taj čvor ima oznaku i

Sada od čvora i činimo spomenutu šetnju kako bismo pronašli sve čvorove konsenzusa. Koraci algoritma su prikazani na slici 2.2.

Slika 2.2: Koraci pronalaska konsenzusa

Prvi graf prikazuje jedan mogući POA graf nakon poravnanja nekih slijedova. Na drugom grafu crvenom bojom su označene veze odabrane za svaki čvor. U svakom čvor je upisana njegova težina, a crvenom bojom su označeni čvorovi dobiveni šetnjom iz čvora najveće težine (ti čvorovi su dio konsenzusa). S obzirom da čvor najveće težine ima izlazne veze, ostalim čvorovima težina je postavljena na -1. Algoritam za odabir veza i računanje težina je primjenjen s modifikacijom da veze čiji početni čvorovi imaju negativnu težinu ne mogu biti izabrani. Na četvrtom grafu vidljive su težine nakon provedbe tog koraka, a crvenom bojom je označen konsenzus.



3. Metode

U prošlom poglavlju je opisano kako iz slijedova izgraditi POA graf postupnim poravnavanjem slijeda na graf. U ovom poglavlju će biti predstavljen algoritam za poravnanje dva već izgrađena POA grafa.
Kreiranju algoritma za poravnanje dvaju POA grafova možemo pristupi na nekoliko načina, npr. modificirati neki od postojećih algoritama za poravnanje dvaju slijedova ili grafa i slijeda u algoritam za poravnanje dvaju POA grafova, izmisliti potpuno novi algoritam poravnanja koji može poravnati dva POA grafa ili jedan graf pretvoriti u neku drugu strukturu za koju znamo izračunati poravnanje. Na žalost, niti jedan od uvriježenih analitičkih algoritama za poravnanje slijeda i grafa (Needleman-Wunsch, Smith-Waterman...) nije moguće modificirati u algoritam poravnanja dvaju grafova bez velikog povećanja složenosti. Ti algoritmi se temelje na dinamičkom programiranju s dvodimenzionalnom matricom stanja za poravnanje grafa i slijeda. Za poravnanje dvaju grafova bilo bi potrebno koristiti višedimenzionalno stanje čime bi povećanje složenosti učinilo algoritam neupotrebljivim. S obzirom da je dinamičko programiranje jedini poznati algoritam za učinkovito, analitičko izračunavanje poravnanja, modifikacija postojećeg algoritma kao podloga za razvoj poravnanja dvaju POA grafova nije moguća. Kreiranje potpuno novog algoritma za poravnanje dvaju grafova bi se trebao temeljiti ili na novoj metrici sličnosti dvaju slijedova ili na heurističkom izračunu postojećih metrika. Ti pristupi neće biti istraženi u ovom radu.

3.1. Rastavljanje POA grafa na slijedove

Algoritam za poravnanje dvaju POA grafova opisanim u ovom radu se temelji na rastavu grafa na slijedove. Rastav grafa na slijedove nije jedinstven te način rastava utječe na složenost poravnanja i na njegovu točnost.
Ako modificiramo POA graf tako da u svakom čvoru i u svakoj vezi pamtimo niz slijedova koji ju dijele tada bismo iz grafa mogli u potpunosti rekonstruirati slijedove iz kojih je izgrađen. Te slijedove možemo poravnati jednog po jednog na drugi POA graf. Takav način rastavljanja nudi jednako dobro poravnanje kao da i nikad nismo gradili graf, no vremenska složenost također ostaje ista. Takvim načinom rastava grafa očito nismo ništa postigli.

Kako bismo smanjili složenost potrebno je graf rastaviti na manji broj slijedova, nego od što je bio izgrađen. Također je dobro minimizirati i duljinu tih slijedova jer složenost poravnanja ovisi o duljini i broju slijedova te o broju čvorova i veza grafa. U ovom radu korišten je modificirani algoritam za pronalazak konsenzuza kako bi se rastavio graf na slijedove. Kako bismo graf rastavili u slijedove ponavljamo slijedeće korake sve dok nismo sve veze izdvojili u neki slijed:

- pronađi konsenzus u grafu
- sve veze iz konsenzusa makni iz grafa; te veze i pripadni čvorovi čine jedan slijed u rastavu
- svaki čvor iz konsenzusa zamijeni s dva čvora: prvi ima sve veze koje su ulazile u originalni čvor, drugi sve koje su izlazile iz njega

Drugi opisani korak činimo kako bi svaka veza u samo jednom slijedu nakon rastava, a važnost trećeg koraka će biti objašnjena u idućem odjeljku. Koraci algoritma su prikazani u primjeru na slici 3.1.

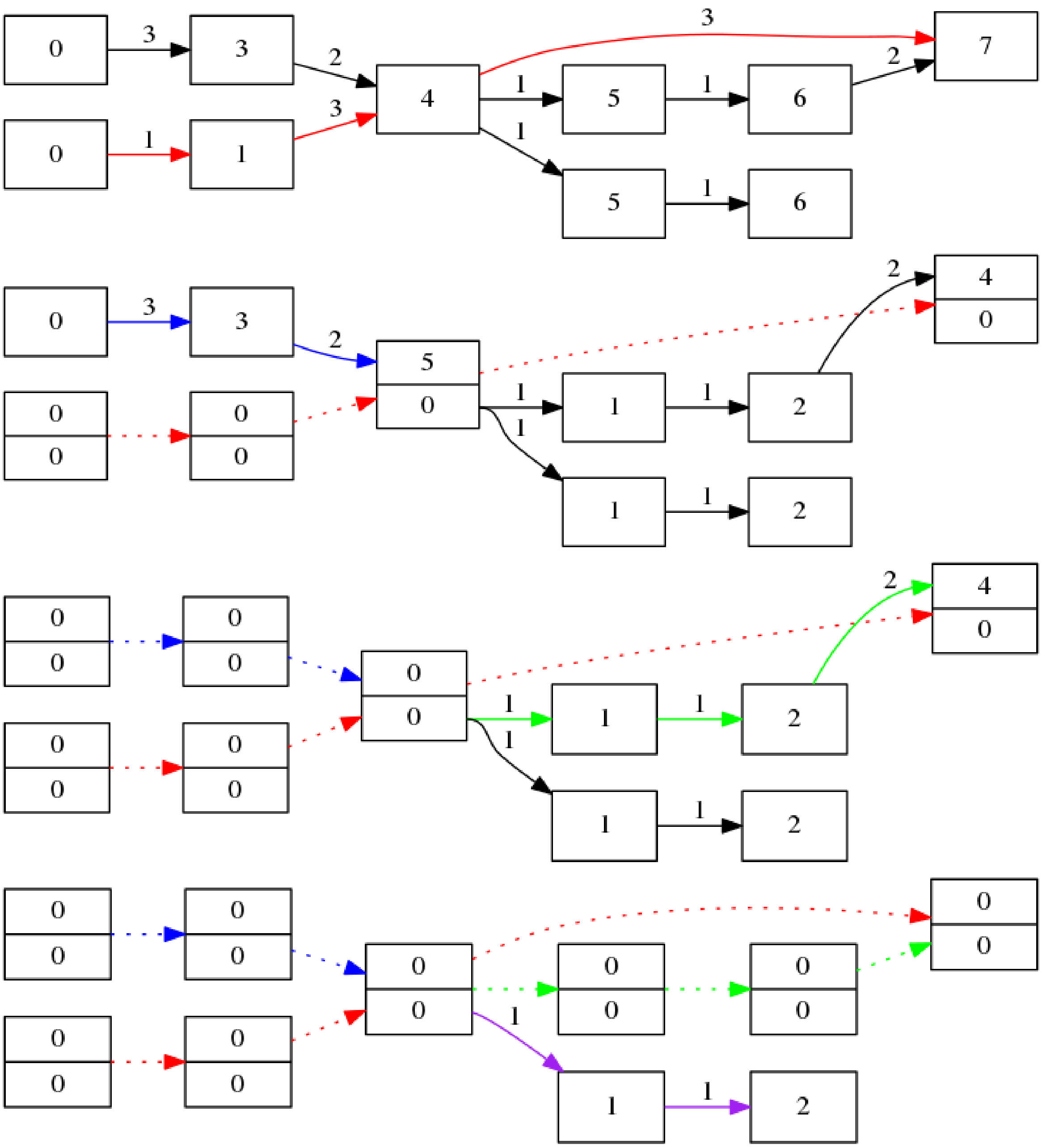
3.2. Poravnanje slijedova iz rastava s POA grafom

Nakon što smo rastavili graf na slijedove potrebno je svaki slijed poravnati na POA graf. Za poravnanje prvog slijeda u ovom radu je korišten algoritam Smith-Watermana. Ako neki od budućih slijedova dijeli početni i/ili završni čvor s nekim od već poravnatih slijedova ne smijemo ih poravnati na bilo koju poziciju u grafu jer dijeljeni čvorovi prvog grafa moraju biti poravnati na isti čvor drugog grafa. Uvedimo mapu mp u kojoj za svaki čvor grafa kojeg smo rastavili na slijedove pamtimo na koji čvor drugog grafa je poravnat. Sada primjećujemo 4 slučaja s obzirom na čvorove koje slijed dijeli s prethodno poravnatim slijedovima te za svaki od njih na malo drugačiji način računamo poravnanje:

- slijed ne dijeli niti početni niti završni čvor
    - koristimo Smith-Waterman algoritam
- slijed dijeli samo završni čvor
    - koristimo Smith-Watermanovu funkciju sličnosti
    - kao rezultat ne uzimamo maksimalnu vrijednost funkcije, nego vrijednost $f(mp(a_n), n)$, gdje je a slijed kojeg poravnavamo, a n njegova duljina
- slijed dijeli samo početni čvor
    - koristimo Needleman-Wunsch funkciju sličnosti
    - kao rezultat uzimamo maksimalnu vrijednost funkcije
- slijed dijeli i početni i završni čvor

Slika 3.1: Koraci rastava POA grafa na slijedove

Na prvom grafu prikazan je prvi pronađeni konsenzus crvenom bojom. Prije nego istim algoritmom potražimo idući slijed, iz grafa mičemo veze iz prvog konsenzusa. Maknute veze prikazane su isprekidanim linijama. Čvorovi prvog konsenzusa su podijeljeni na dva čvora. Jedan čvor ima sve ulazne veze, a drugi sve izlazne veze originalnog čvora. Svaki čvor ima svoj težinu u izračunu novog konsenzusa. Nakon načinjenih transformacija grafa pronalazimo drugi konsenzus čije su veze prikazane plavom bojom. Primijetimo da prvi i drugi konsenzus dijele jedan čvor, no imaju jedistvene veze. Nakon što smo ponovili korake algoritma još dvaput našli smo konsenzuse prikazane zelenom i ljubičastom bojom.



- koristimo Needleman-Wunsch funkciju sličnosti
- izračun funkcije ne počinjemo na početnom čvoru grafa, nego na čvoru $mp(a_1)$, gdje je a slijed za koji računamo poravnanje
- kao rezultat uzimamo vrijednost $f(mp(a_n), n)$

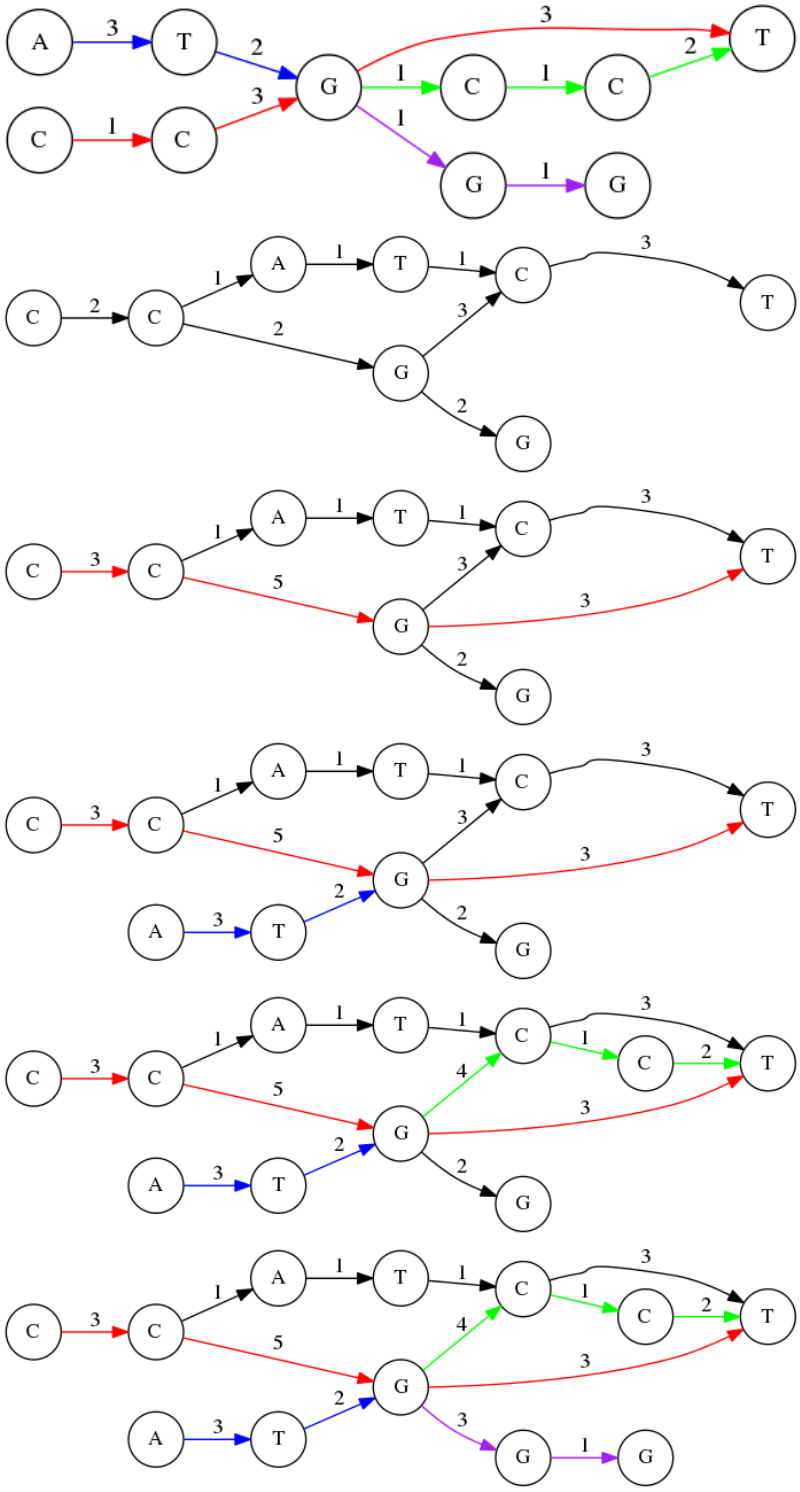
Na slici 3.2 je prikazan primjer poravnanja rastavljenih sljedova iz grafa sa slike 3.1 na drugi POA graf.

3.3. Nemoguća poravnanja

Može se dogoditi da je početni i završni čvor nekog slijeda dijeljen, ali u grafu nad kojim se vrši poravnanje ne postoji put iz čvora na koji je mapiran početni čvor u čvor na koji je mapiran završni čvor. U praksi se pokazalo da se to događa relativno rijetko na kratkim slijedovima koji ne utječu bitno na strukturu grafa. U ovoj implementaciji takvi slijedovi su jednostavno zanemareni, tj. nisu poravnati nego ih se preskočilo.

Slika 3.2: Prikaz poravnanja rastavljenih slijedova na POA graf

Prvi graf prikazuje jedan mogući POA graf nakon poravnanja nekih slijedova. Na Prvi graf na slici je podijeljen na slijedove, a nad drugim grafom sa slike se vrše poravnanja. Treći graf prikazuje drugi graf nakon poravnanja s prvim slijedom, koji je označen crvenom bojom. Na četvtom grafu je poravnat i drugi slijed. Slijed ima poravnanje veće sličnosti u gornjoj grani grafa (čvorovi sa znakovima A i T), no tamo ne smije biti poravnat jer mora završiti u čvoru kojeg dijeli s već poravnatim slijedom. Iduća dva grafa prikazuju poravnanja ostala dva slijeda.



4. Implementacija

Za implementaciju algoritma korišten je programski jezik C++, točnije standard C++11. Program je razvijen na operativnom sustavu Linux, no moguće ga je pokrenuti na svakom operativnom sustavu na kojem je dostupan moderni C++ prevedioc te alati *CMake* i *make*. U kodu nije bio naglasak na brzinu programa, nego kod služi kao primjer implementacije novorazvijenog algoritma. Kao takav može poslužiti kao prototip za moguću efikasniju implementaciju u budućnosti. Kod je javno dostupan na stranici `https://github.com/mbradac/poa_alignment`.
U nastavku ovog poglavlja će biti prikazane implementacije nekih od ključnih dijelova algoritma.

4.1. Topološko sortiranje

Prvi korak poravnanja slijeda i grafa te pronalaska konsenzusa u POA grafu je topološko sortiranje čvorova. Za niz čvorova usmjerenog grafa kažemo da je topološki sortiran ako za svaki čvor x vrijedi da svi čvorovi od kojih se može doći do čvora x se nalaze prije njega u nizu. Za grafove s ciklusom ne postoji topološki poredak, no niti jedan ispravno izgrađeni POA graf nema ciklus. U nastavku je prikazana jedna moguća implementacija topološkog sorta.

```
auto TopologicalSort(
    const std::vector<Node *> &start_nodes) {
  // Push start_nodes on stack.
  std::stack<Node *> s;
  for (auto *start_node : start_nodes) {
    s.push(start_node);
  }

  std::vector<Node *> sorted;
  while (!s.empty()) {
```

```
    auto *node = s.top();
    s.pop();
    sorted.push_back(node);
    for (auto next : node->edges) {
      if (!--next->indegree) {
        s.push(next);
      }
    }
  }
  return sorted;
}
```

Jedini parametar funkcije je `start_nodes` – niz čvorova grafa koji nemaju niti jednu ulaznu vezu. Algoritam u svakom koraku **`while`** petlje uzima čvor s vrha stoga, stavlja ga na kraj liste u kojem se nalazi topološki sortirani niz obrađenih čvorova, te prolazi po svim njegovim izlaznim vezama. Za čvor svake izlazne veze smanjuje brojač `indegree` u kojem je na početku zapisano koliko postoji ulaznih veza u taj čvor. Kada brojač padne na 0 sigurni smo da se svi čvorovi koji se trebaju nalaziti ispred, već nalaze u sortiranoj listi te tada taj čvor stavljamo na stog. Nakon što stog ostane prazan ili su svi čvorovi sortirani ili postoji ciklus u grafu, no kao što je već spomenuto u POA grafovima nikad nema ciklusa te smo sigurni da smo sortirali sve čvorove.

4.2. Izračun funkcije sličnosti slijeda i grafa

Vrijednosti funkcije sličnosti slijeda i grafa računamo iteravnim dinamičkim programiranjem, tj. u tablici `f` ćemo zapamtiti sve moguće vrijednosti funkcije sličnosti f. Na početku unosimo u tablicu sve početne vrijednosti funkcije – one vrijednosti koje nije potrebno izračunavati rekurzivno. U svakom idućem koraku uzimamo jednu vrijednost za koju smo sigurni da je izračunata te iz nje pomoću prijelaza definiranih rekurzivnom relacijom u poglavlju 2.6 računamo iduće vrijednosti funkcije. U nastavku je dan kod koji računa sličnosti po funkciji Smith-Waterman (primijetimo da su na početku implicitno sve vrijednosti funkije postavljene na 0 zbog korištenja `std::unordered_map`-e):

```
auto CalculateAlignment(
    const std::vector<Node *> start_nodes,
    const std::string &sequence,
    const std::vector<std::vector<int>> &score,
    int d) {
```

```
  auto nodes = TopologicalSort(start_nodes);
  std::unordered_map<std::pair<int, Node *>, int> f;

  // Init values for start nodes.
  for (Node *start_node : start_nodes) {
    f[{0, start_node}] = std::max(
        0, score[sequence[0]][start_node->letter]);
  }

  // Calculate values of function f.
  for (int i = 0; i < sequence.size(); ++i) {
    for (auto *node : nodes) {
      for (auto next : node->edges) {
        f[{i, next}] = std::max(
            f[{i, next}], f[{i - 1, next}] + d);
        f[{i, next}] = std::max(
            f[{i, next}], f[{i, node}] + d);
        f[{i, next}] = std::max(
            f[{i, next}], f[{i - 1, node}] +
                          s[sequence[i]][next->letter]);
      }
    }
  }
  return f;
}
```

Funkcija prima 4 parametra: početne čvorove grafa, niz znakova slijeda za koji računamo sličnost, matricu `score` koja predstavlja funkciju $S(x, y)$ (težina zamjene/podudaranja znakova te težinu d (težina umetanja/brisanja znaka). Prilagodba funkcije da pronalazi i najveću vrijednost u tablici ili da računa sličnost po Needleman-Wunsch algortimu je jednostavna te se u praksi lagano može modificirati funkciju i koristitu ja za sve četiri vrste poravnanja.

4.3. Glavna petlja

U nastavku je prikazana malo pojednostavljena vršna funkcija za poravnanje dvaju POA grafova:

```
void AlignGraphToGraph(
    Graph &graph1, Graph &graph2,
```

```
    const std::vector<std::vector<int>> &score,
    int d) {
  std::unordered_map<Node *, Node *> mp;

  while (true) {
    auto concensus = FindConcensus(graph2.start_nodes);
    if (concensus.size() <= 1U) break; // No edges left.
    graph2.RemovePathAndSplitNodes(concensus);
    auto start_node = mp.find(concensus[0]);
    auto end_node = mp.find(concensus.back());
    auto subgraph = graph1.Subgraph(start_node, end_node);
    auto aligned_path =
        AlignSequenceToGraphByType(subgraph, concensus, score, d,
                                   Type(start_node, end_node));
    for (int i = 0; i < path.size(); ++i) {
      mp[concensus[i]] = path[i];
    }
  }
}
```

Kao što je vidljivo iz priloženog koda vršna logika poravnanja dvaju grafova se odvija u jednoj petlji. U toj petlji pronalazimo slijed pomoću funkcije za nalazak konsenzusa. U slučaju da više ne postoji postoji veza koja nije ni u jednom slijedu prekidamo petlju i završili smo s poravnanjem. Zatim funkcija `RemovePathAndSplitNodes` briše iz grafa veze iz konsenzusa te čvorove koncenzusa dijeli u dva čvora kao što je opisano u poglavlju 3.1. Iz mape `mp` pronalazimo čvor grafa `graph1` koji je poravnat s početnim, odnosno završnim čvorom slijeda te iz grafa `graph1` izdvajamo samo podgraf nad kojim računamo sličnost. Naime, kao što je opisano u poglavlju 3.2, ako je početni ili završni čvor dijeljen onda ne računamo lokalno poravnanje, nego moramo osigurati da poravnanje počinje odnosno/završava u dijeljenom čvoru. To činimo na dva načina:

- izdvajanjem podgrafa tako da je dijeljeni čvor početni, tj. završni čvor podgrafa
- u ovisnosti o tome postoje li mapiranja `start_node` i `end_node` funkcija `Type` računa vrstu poravnanja koju funkcija `AlignSequenceToGraphByType` mora izračunati

Nakon što je slijed uspješno poravnat na graf preostaje još samo obnoviti mapu `mp` novim vrijednostima.

5. Rezultati

Implementacija je testirana na skupovima slijedova duljine oko 500 znakova. U svakom mjerenju je iz skupa od 54 slijeda slučajno odabran podskup zadane veličine. Za svaki podskup je traženje konsensuza bilo računato na dva različita načina:

- POA graf je izgrađen dodavajući jedan po jedan slijed te je iz konačnog grafa pronađen konsenzus
- POA graf je izgađen spajanjem postojećih POA grafova, tj. prvo su izgrađeni POA grafovi od samo dva slijeda, zatim su parovi tih grafova spojeni u grafove od četiri slijeda, pa u grafove od osam slijedova,... sve dok svi grafovi nisu spojeni u jedan te je na tom grafu izračunat konsenzus

U nastavku je kod funkcije koja prikazuje strategiju spajanja grafova iz drugog opisanog načina računanja konsenzusa:

```
Graph GraphTournamentBuildPoa(
    std::vector<Sequence> sequences,
    const ScoreMatrix &matrix, int penalty,
    NodeStorage &storage) {
  int n = sequences.size();
  if (n == 1) {
    TranslateSequence(&sequences[0], matrix);
    return Graph(&storage, sequences[0]);
  }

  int mid = (n + 1) / 2;
  auto g1 = GraphTournamentBuildPoa(
      std::vector<Sequence>(sequences.begin(),
                            sequences.begin() + mid),
      matrix, penalty, storage);
  auto g2 = GraphTournamentBuildPoa(
      std::vector<Sequence>(sequences.begin() + mid,
      sequences.end()),
```

Tablica 5.1: Rezultati poravnanja slijedova veličine 500 znakova

	Metoda 1					Metoda 2				
br. slijedova	br. čvorova	br. veza	suma težina	sličnost	vrijeme/s	br. čvorova	br. veza	suma težina	sličnost	vrijeme/s
4	616	764	1953	3634	2.18	611	766	1953	3595	2.11
8	752	1083	3823	3530	5.46	720	1024	3741	3476	5.62
12	853	1348	5682	3464	10.56	755	1176	5473	3459	10.80
16	912	1541	7631	3502	16.29	858	1405	7428	3498	13.25
20	983	1755	9546	3474	23.22	929	1578	9227	3476	17.93
24	1066	1980	11487	3488	31.75	984	1703	11010	3486	21.65
28	1100	2109	13383	3482	43.88	1020	1864	12864	3488	25.89
32	1132	2205	14973	3498	41.42	1036	1941	14336	3520	29.75

```
      matrix, penalty, storage);
  AlignGraphToGraph(g1, g2, matrix, penalty);
  return g1;
}
```

Prvi način računanja konsenzusa za svaki podsup je ponovljen još jednom s drugačijim redoslijedom poravnavanja slijedova. Taj konsenzus je korišten za usporedbu s prva dva konsenzusa da bi se vidjelo postoje li drastična odstupanja u prethodno navedenim metodama za računanje konsenzusa. Za usporedbu sličnosti konsenzusa korišten je algoritam Smith-Waterman. U svim poravnanjima je korištena matrica BLOSUM45 i težina -1 za umetanje/brisanje znakova. U svakom mjerenju osim konsenzusa je izbrojan broj čvorova, odnosno veza u završnim POA grafovima te ukupan zbroj težina svih veza grafa (kako bi se utvrdilo koliko veza nije moglo biti poravnato u drugoj metodi zbog razloga opisanih na kraju poglavalja 3.2). Rezultati su prikazani u tablici 5.1.

Kao što je vidljivo iz tablice sličnost konsenzusa dobivenih iz obje metode je otprilike ista u usporedbi s baznim konsenzusom. Mjerenje bi trebalo ponoviti na većim slijedovima da bismo mogli zaključiti koja metoda daje točniji konsenzus. Ukupna suma težina malo je pala u drugoj metodi zbog nemogućnosti poravnanja nekih slijedova. Najveći pad je bio u mjerenju podskupa s 28 slijedova gdje je izgubljeno oko 4% ukupne težine veza. Vrijeme izvršavanja je raslo s brojem slijedova, no u drugoj metodi je vidljiv sporiji rast zbog rekurzivnog spajanja grafova koji smanjuje složenost cijelog algoritma. Bitno je napomenuti da je implementacija algoritma daleko od optimalne te da bi se pojedini dijelovi mogli višestruko ubrazati.

6. Zaključak

Poravnanje i ocjenjivanje sličnosti slijedova je jedan od temeljnih problema bioinformatike. Analitički algoritmi za izračunavanje poravnanja poput Smith-Watermana i Needleman-Wunscha se koriste kao temeljnji dio mnogih drugih algoritama. Iako su konceptualno jednostavni nisu direktno upotrebljivi za poravnanje više od dva slijeda zbog svoje velike memorijske i vremenske složenosti.

Kao praktična struktura za obradu više slijedova se pokazao POA graf koji na prirodan način prikazuje poravnjanje više slijedova, a kao temeljnu logiku za poravnanje iskorištava već poznate analitičke algoritme. Osim poravnanja POA graf je prikladan za računanje konsenzusa.

U ovom radu je predstavljen algoritam za poravnanje dva već postojeća POA grafa. Algoritam se temelji na algoritmu za pronalazak konsenzusa nad grafom te nad klasičnim algoritmom poravnanja slijeda i grafa. Uz rad je implementiran i prototip u jeziku C++ te je predstavljena i njegova moguća praktična primjena za problem pronalaska konsenzusa.

Iako tijekom implementacije nije bio naglasak na brzini u mjerenjima se vidjelo ubrzanje koje je donio algoritam naspram klasičnog algoritma za pronalazak konsensuza. Algoritam se može dalje ubrzati paralelizmom: SIMD poravnanje slijeda i grafa, višedretvena raspodjela poslova spajanja grafova... Također se algoritam može unaprijediti iskorištavanjem već izračunatih težina tijekom izračuna konsenzusa u postupku rastavljanja grafa na slijedove u slučaju da nije bilo promjena u tom podgrafu. U ovom radu je izložen i test točnosti po kojem izgleda da izloženi algoritam nije lošiji od već postojećeg. Na žalost, test nije dovoljno dobar te treba detaljnije istražiti točnost algoritma kako bi se ustvrdilo isplati li se daljni rad nad njime.

LITERATURA

[1] Christian D. Wunsch Saul B. Needleman. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[2] M. S. Smith. Some biological sequence metrics. *Advances in Mathematics*, 20(3):367–387, 1976.

[3] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982.

[4] M. S. Waterman T. F. Smith. Identification of common molecular subsequences. *J Mol Biol.*, 147:195–197, 1981.

[5] Mark F. Sharlow Chritopher Lee, Catherine Grasso. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

[6] Chritopher Lee. Generating concsensus sequences from partial order multiple sequence alignment graphs. *Bioinformatics*, 19(8):999–1008, 2003.

[7] Martin Šošić. An simd dynamic programming c/c++ library, 2015.

[8] Tobjørn Rognes. Faster smith-waterman database searches with inter sequence simd parallelisation. *BMC Bioinformatics*, 12:221, 2011.

Algoritam za određivanje ukupnog poravnanja dva grafa poravnanja parcijalnog uređaja

Sažetak

Poravnanja više slijedova je jedan od temeljnih problema bioinformatike. Za analitično poravnanje dva slijeda već se desecima godina koristi Smith-Waterman obitelj algoritama, no zbog svoje velike prostorne i vremenske složenosti ti algoritmi nisu pogodni za poravnanje većeg broja slijedova. Kao prvi korak poravnanja većeg broja slijedova izgrađuje se graf poravnanja parcijalnog uređaja koristeći modificirani Smith-Waterman algoritam. Tako izgrađen graf pogodan je za daljnu analizu slijedova: generiranje poravnanja većeg broja slijedova te pronalaženja konsenzusa. U ovom radu predstavljen je algoritam za spajanje dva već postojeća grafa poravnanja parcijalnog uređenja koji zbog korištenja već izgrađenog grafa smanjuje broj koraka potrebnih za igradnju većeg grafa.

Ključne riječi: bioinformatika, POA, graf poravnanja parcijalnog uređaja, konsenzus, Smith-Waterman, poravnanje slijedova

Algorithm for the Alignment of Two Partial Order Alignment Graphs

Abstract

Sequence alignment is one of fundemental problems in bionformatics. Alignment of two sequences is done using Smith-Waterman family of algorithms. These algorithms are not useable for alignment of multiple sequences because of theirs big time and memory complexity. Partial order alignment (POA) graph is a first step of calculating multiple sequence alignment efficiently. POA graph is suitable for further analysis of sequences: calculation of multiple sequence alignment and concensus generation. In this thesis algorithm for merging two already existing POA graph is presented. Algorithm reduces number of steps needed to build bigger POA graphs by reusing parts of existing ones.

Keywords: bioinformatics, POA, partial order alignment graph, concensus, Smith-Waterman, sequence alignment