Poravnanje više nizova

Grubelić, Neven

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Neven Grubelić

PORAVNANJE VIŠE NIZOVA

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Saša Singer

Zagreb, studeni, 2015.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Poravnanje više nizova (engl. Multiple sequence alignment) je problem porijeklom iz biologije i genetike, a najčešće se odnosi na proteine, DNA ili RNA sekvence. Sam problem nastao je zbog potrebe proučavanja podudarnosti među nizovima proteina, DNA ili RNA sekvenci pronađenih u biološkim jedinkama koje su, najčešće, evolucijski povezane, tj. imaju zajedničkog pretka. Istraživačima je želja bila omogućiti brzo i efikasno pronalaženje podudaranja na danim nizovima kako bi mogli, koristeći te informacije, dobiti bolji uvid u funkcije pojedinih proteina u organizmu, ili razloge genetskih mutacija koje su nastale evolucijom, te možda čak i pronaći lijek za neke genetske bolesti izazvane takvim mutacijama. Jedan primjer takve konkretne upotrebe poravnanja više nizova je pronalazak odgovarajućih mjesta u ljudskom genetskom kôdu na kojima se pojavljuju mutacije koje dovode do obolijevanja od cistične fibroze. Upravo je to saznanje jako doprinijelo u osjetnom produljenju životnog vijeka oboljelih osoba.

S obzirom da je ovaj problem poznat, te se istražuje već nekoliko desetaka godina, došlo je do značajnog napretka u tehnikama poravnavanja. Današnji računalni algoritmi u stanju su poravnavati istovremeno na desetke nizova i dati prilično dobra rješenja već u nekoliko sekundi. U ovom radu spomenuti su svi danas najpoznatiji algoritmi za rješavanje ovog problema, a detaljnije je obrađen genetski algoritam za poravnanje više nizova.

Genetski algoritmi su, na neki način, možda i najprikladniji za prezentiranje ovog problema, budući da je motivacija za njihovo nastajanje došla iz prirode, tj. iz načina na koji se događaju genetske mutacije te križanja među jedinkama u procesu razmnožavanja. Ovaj rad podijeljen je u 4 poglavlja, u kojima će se redom dati uvid u samu definiciju problema, opis dosad poznatih algoritama rješavanja, općenitu implementaciju jednog genetskog algoritma, te rezultate dobivene korištenjem spomenutog genetskog algoritma uz različite ulazne parametre.

Poglavlje 1

Problem poravnanja nizova

1.1 Općenito

Kao što je u uvodu i rečeno, poravnavati se mogu nizovi proteina, te DNA ili RNA sekvence, a ovaj se rad isključivo bavi poravnavanjem proteinskih nizova. Proteinski nizovi su nizovi aminokiselina koje, povezane u lanac peptidnim vezama, tvore protein. Poredak aminokiselina u nizovima određen je genetskim zapisom i stoga, poredak aminokiselina u svakom trenutku mora ostati nepromijenjen. Aminokiseline koje tvore proteine mogu se podijeliti u 20 skupina koje označavamo troslovnim ili jednoslovnim kraticama, kao što se može vidjeti u tablici 1.1.

Osim ovdje prikazanih aminokiselina, postoje još dvije nedavno otkrivene i uspješno sintetizirane aminokiseline, a to su selenocistin i pirolizin, s oznakama **U** i **O**, pa je tako ukupan broj danas poznatih aminokiselina 22. Također, zbog nemogućnosti egzaktnog određivanja neke aminokiseline kemijskom analizom, uvode se još i kratice **B**, **Z**, **J** i **X**. One označavaju više mogućih aminokiselina koje se mogu naći na tom mjestu u nizu. Primjeri na kojima je algoritam testiran nisu sadržavali ove dodatne aminokiseline, pa se iz tog razloga ovaj dio rada neće njima previše baviti.

Način reprezentiranja aminokiselina bitan je iz razloga što će proteinski nizovi u algoritmima biti reprezentirani kao nizovi jednoslovnih kratica aminokiselina, te znakom praznine **–**. U takvom sustavu označavanja, usporedba aminokiselina na određenom mjestu u proteinskom nizu svodit će se na uspoređivanje znakova na istoj lokaciji, u svim nizovima, te njihovom bodovanju.

1.2 Ciljne funkcije

Definirajmo sada osnovni problem poravnanja više nizova.

aminokiselina	engleski naziv aminokiseline	troslovna kratica	jednoslovna kratica
alanin	Alanine	Ala	A
arginin	Arginine	Arg	R
asparagin	Asparagine	Asn	N
asparaginska kiselina	Aspartic acid	Asp	D
cistein	Cysteine	Cys	C
glutamin	Glutamine	Gln	Q
glutaminska kiselina	Glutamic acid	Glu	E
glicin	Glycine	Gly	G
histidin	Histidine	His	H
izoleucin	Isoleucine	Ile	I
leucin	Leucine	Leu	L
lizin	Lysine	Lys	K
metionin	Methionine	Met	M
fenilanin	Phenylalanine	Phe	F
prolin	Proline	Pro	P
serin	Serine	Ser	S
treonin	Threonine	Thr	T
triptofan	Tryptophan	Trp	W
tirozin	Tyrosine	Tyr	Y
valin	Valine	Val	V

Tablica 1.1: Tablica dvadeset poznatih aminokiselina i njihove kratice

Definicija 1.2.1. *Neka je* $S = \{S_1, \ldots, S_k\}$ *skup od k proteinskih nizova.* ***Poravnanje*** *M nizova iz S je skup od k nizova jednake duljine* $M = \{S'_1, \ldots, S'_k\}$*, pri čemu je svaki niz* S'_i *dobiven umetanjem praznina u niz* S_i*.*

Definicija 1.2.2. ***Problem poravnanja nizova*** *je problem pronalaženja poravnanja nizova iz skupa S koji maksimiziraju funkciju sličnosti ili minimiziraju funkciju udaljenosti među nizovima.*

Najčešće korištene funkcije sličnosti i udaljenosti (ciljne funkcije) su:

- suma sličnosti u parovima:

$$\sum_{1 \leq i < j \leq k} slicnost_M(S'_i, S'_j),$$

- suma udaljenosti u parovima:

$$\sum_{1 \leq i < j \leq k} udaljenost_M(S'_i, S'_j),$$

pri čemu su $slicnost_M(S'_i, S'_j)$ te $udaljenost_M(S'_i, S'_j)$ funkcije sličnosti i udaljenosti između dva niza.

Samo bodovanje udaljenosti među dvama nizovima duljine n je intuitivno jasno. Dakle,

$$udaljenost_M(S'_i, S'_j) = \sum_{k=0}^{n} (S'_{ik} == S'_{jk}),$$

pri čemu S'_{ik} označava k-tu aminokiselinu u nizu S_i, a operator **==** vraća 1 u slučaju istinitosti, a inače 0.

Što se tiče sličnosti nizova, tu postoji više načina bodovanja. Prvo među njima je najintuitivnije, a to je upravo brojanje identičnih pozicija u nizovima, tj. upravo obrnuto od bodovanja udaljenosti nizova. Kao i kod bodovanja udaljenosti nizova, ovaj način bodovanja ima veliku manu u tome što ne radi distinkciju među pojedinim slučajevima razlikovanja aminokiselina. Naime, svaka aminokiselina u prirodi ima određene vjerojatnosti mutacije u neku drugu aminokiselinu. Upravo iz tog razloga izvedene su supstitucijske matrice za računanje funkcije sličnosti među pojedinim elementima nizova.

1.3 Supstitucijske matrice

PAM (engl. Point Accepted Mutation) matrice prva je uvela Margaret Dayhoff, 1978. godine [2]. Služile su za ocjenjivanje vjerojatnosti mutacije jedne aminokiseline u drugu, a dobivene su ekstrapolacijom informacija o mutacijama između izrazito evolucijski bliskih nizova (99% sličnosti). Zato su se prvotno koristile u poravnanju proteina vrlo bliskih jedinki, tj. proteina koji su poravnati imali barem 85% podudarnosti, što je onda dovodilo do pretpostavke da su razlikovanja u poravnanju posljedice jedne mutacije, a ne više njih, na istom mjestu. Za potrebe poravnanja različitijih nizova, kasnije su iz originalne PAM_1 matrice izvedene PAM_n matrice koje su omogućile ocjenjivanje poravnanja evolucijski različitih proteinskih nizova. Naime, indeks n označavao je prosječni broj mutacija na svakih 100 aminokiselina.

BLOSUM (engl. BLOcks Substitution Matrix) matrice su izveli Henikoff i Henikoff, 1992. godine [3]. Nastale su kao pokušaj poravnavanja evolucijski različitih nizova proteina. Upravo zbog toga su dobivene promatranjem mutacija koje se pojavljuju unutar *motiva* (blokova aminokiselina vrlo sličnih konfiguracija i uloga) evolucijski povezanih proteinskih nizova. Za razliku od PAM matrica, index n u $BLOSUM_n$ matrici označava da je matrica nastala usporedbom nizova koji imaju najviše n% podudarnosti.

Obje skupine matrica su simetrične, reda 20, pri čemu svaki redak i svaki stupac predstavljaju jednu aminokiselinu, a element matrice predstavlja iznos funkcije sličnosti između jedne aminokiseline (redak) i druge (stupac), kao što se može vidjeti na slici 1.1.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	2																			
R	-2	6																		
N	0	0	2																	
D	0	-1	2	4																
C	-2	-4	-4	-5	12															
Q	0	1	1	2	-5	4														
E	0	-1	1	3	-5	2	4													
G	1	-3	0	1	-3	-1	0	5												
H	-1	2	2	1	-3	3	1	-2	6											
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5										
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6									
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5								
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6							
F	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	9						
P	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	6					
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2				
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3			
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17		
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4

(a) PAM250

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

(b) BLOSUM62

Slika 1.1: Primjeri supstitucijskih matrica

1.4 Kažnjavanje praznina

Još nam preostaje definirati ocjenjivanje **praznina** u nizovima. To su mjesta u nizu koja su popunjena uzastopnim znakom **–**. Prema tome, niz **A T – – – C C T G A** ima jednu *prazninu* duljine 3, jer je sastavljena od tri *znaka praznine*, a niz **A – T – A C C T – A** ima tri *praznine* duljine 1, jer svaka praznina odgovara jednom *znaku praznine*.

Konstantno kažnjavanje

Najjednostavnija metoda kažnjavanja praznina u nizu je **konstantno kažnjavanje**. Ono je definirano na način da pridajemo konstantnu vrijednost **c** cijeloj *praznini*, tako da svaka praznina u nizu u funkciji sličnosti sudjeluje s $-c$ bodova, neovisno od koliko *znakova praznine* se sastoji. Detaljnije objašnjenje dano je na primjeru na slici 1.2.

Linearno kažnjavanje

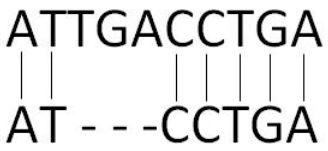
Sljedeća metoda kažnjavanja je **linearno kažnjavanje** koje u obzir uzima duljinu praznine. Kod linearnog kažnjavanja pridaje se konstantna vrijednost **c** na način da svaki *znak praznine* u funkciji sličnosti sudjeluje s $-c$ bodova. Na ovaj način se više vrednuju kraće praznine u nizu.

Afino kažnjavanje

Najčešće korišteno kažnjavanje praznina je **afino kažnjavanje**. Ono, za zadane konstantne vrijednosti **a** i **b**, svaku prazninu kažnjava s $a + (b \cdot L)$, pri čemu je L duljina praznine. U ovom slučaju se vrijednost a naziva *kazna započinjanja praznine*, a b *kazna produljivanja praznine*. Često je vrlo bitno pitanje odabira vrijednosti a i b, jer njihov odabir značajno utječe na konačni rezultat [1]. Općenito pravilo kojim se treba voditi je da, u slučaju poravnavanja evolucijski bliskih nizova u kojima se ne preferira veliki broj praznina, poželjno je jače kazniti svaku prazninu, dok je kod poravnavanja evolucijski udaljenijih nizova poželjno smanjiti kažnjavanje praznina. Također treba, u ovisnosti o situaciji, odlučiti preferira li se manje duljih ili više kraćih praznina i sukladno tome odrediti parametre a i b. Jedan primjer je prikazan na slici 1.2.

Konveksno kažnjavanje

Smatrajući da afino kažnjavanje ne modelira dovoljno dobro stvarnu situaciju iz prirode, razvijena je fleksibilnija metoda **konveksnog kažnjavanja** ili **logaritamskog kažnjavanja**. Ova metoda je motivirana studijama koje su pokazale da distribucija duljine praznina u prirodi prati zakon potencija (duljina praznine je linearno povezana sa svojom učestalosti



Slika 1.2: a) Konstantno kažnjavanje – Ako uzmemo da vrijedi 1 bod za točno poklapanje, 0 za netočno, a -1 za svaku prazninu, rezultat ovog poravnanja bio bi $7-1=\mathbf{6}$. b) Afino kažnjavanje – Ako uzmemo parametre $a=2$ i $b=1$, a svako točno poravnavanje ocjenjujemo s 1, vrijednost ovog poravnanja bi bila $7-2-(3\cdot 1)=\mathbf{2}$.

na log-log skali). Stoga je samo konveksno kažnjavanje definirano kao $a+c\cdot\ln L$, pri čemu su parametri a i c zadani i fiksni, a L je duljina praznine.

Poglavlje 2

Metode poravnavanja nizova

2.1 Dinamičko programiranje

Prvi algoritam poravnanja nizova temelji se na dinamičkom programiranju. Dinamičko programiranje je način rješavanja složenijih problema svođenjem na manje i lakše rješive potprobleme, kojima se rezultati izračunavaju samo jednom i spremaju u memoriju. Kada se sljedeći put pokaže potreba za izračunavanjem nekog već izračunatog potproblema, rješenje se samo pročita iz memorije i time se uštedi vrijeme koliko bi ponovno računanje trajalo. Da bismo to mogli izvesti, rješenja svih dosad izračunatih potproblema moraju biti korektno pohranjena i indeksirana, kako bi se omogućilo njihovo jednostavno i brzo čitanje.

Jedan algoritam dinamičkog programiranja za poravnanje više nizova je proširenje algoritma za poravnanje dva niza kojeg su predložili Needleman i Wunsch [6].

Needleman–Wunsch algoritam

Označimo nizove koje poravnavamo sa S_1 i S_2. Tada $S_1[i]$ označava i-tu aminokiselinu u nizu S_1. Analogno vrijedi i za S_2. Ako s F označimo dvodimenzionalnu matricu u kojoj se pohranjuju rezultati manjih potproblema, tada element F_{ij} označava najveći mogući iznos funkcije sličnosti poravnanja prvih i elemenata iz S_1 i prvih j elemenata iz S_2. Svaki element F_{ij} može se dobiti na tri različita načina, a to su:

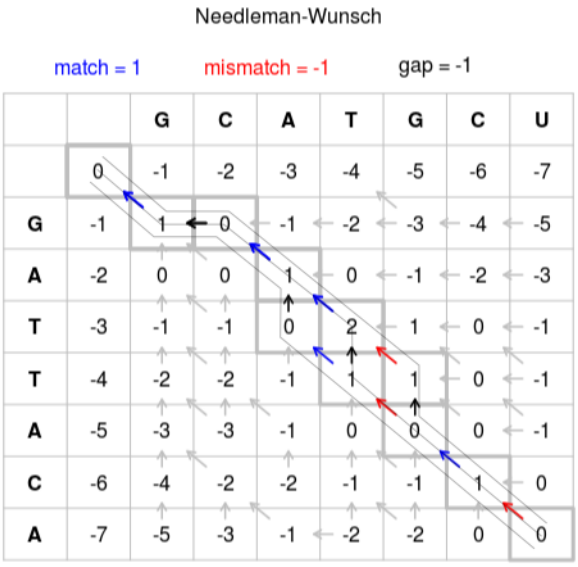
- Iz elementa iznad – označava umetanje znaka praznine u S_1
- Iz elementa lijevo – označava umetanje znaka praznine u S_2
- Iz elementa dijagonalno lijevo-gore – to bi označavalo poravnanje i-tog znaka niza S_1 sa j-tim znakom niza S_2.

Prilikom popunjavanja matrice važno je zapamtiti iz kojeg smjera je rezultat dobiven kako bi se na kraju moglo rekonstruirati najbolje moguće poravnanje. Primjer ovog postupka možemo vidjeti na slici 2.1.

Sada ćemo matematički, koristeći rekurzivne funkcije, definirati algoritam. Neka je kažnjavanje praznina linearno s parametrom $c = 1$. Ako s δ označavamo supstitucijsku matricu, a s $\delta(a, b)$ funkciju sličnosti među aminokiselinom a iz jednog niza i aminokiselinom b iz drugog niza, tada algoritam definiramo:

- $F_{i0} = -c \cdot i$
- $F_{0j} = -c \cdot j$
- $F_{ij} = \max(F_{i-1,j-1} + \delta(S_1[i], S_2[j]), F_{i-1,j} - c, F_{i,j-1} - c)$

Najveći iznos funkcije sličnosti (ocjenu najboljeg poravnanja) cijelih nizova računamo kao F_{n_1,n_2}, pri čemu su n_1 i n_2 duljine nizova S_1, odnosno, S_2.



Slika 2.1: Primjer Needleman–Wunsch algoritma za poravnanje dva niza.

Nakon popunjavanja ove matrice preostaje, iz elementa F_{n_i,n_j}, pronaći smjer iz kojeg smo do njega došli i to ponavljati za svaki element, dok ne dođemo u ishodišni element F_{00}. Tada možemo, praćenjem označenih strelica iz F_{00}, rekonstruirati poravnanje za koje je funkcija sličnosti najveća. Treba obratiti pažnju da, zbog mogućnosti dobivanja jednog elementa iz matrice iz više različitih smjerova, a s istim iznosom funkcije sličnosti,

u konačnici možemo rekonstruirati više poravnanja koja daju maksimalni iznos funkcije sličnosti.

U slučaju da želimo koristiti drugačije kažnjavanje praznina od linearnog, npr. afino, kod popunjavanja elemenata matrice morali bismo u obzir uzeti sve elemente iznad ili lijevo od onog kojeg popunjavamo.

Proširenje N–W

Needleman–Wunsch algoritam može se vrlo lako proširiti na problem poravnanja k nizova. U tom slučaju, matrica F je k-dimenzionalna, a $P(S_1[i_1], S_2[i_2], \ldots, S_k[i_k])$ označava funkciju sličnosti za stupac u poravnanju koji se sastoji, redom odozgo prema dolje, od znakova $S_1[i_1], S_2[i_2], \ldots, S_k[i_k]$. Tada se algoritam definira s:

$$F_{i_1,i_2,\ldots,i_k} = \max \begin{cases} F_{i_1-1,i_2-1,\ldots,i_k-1}, & +P(S_1[i_1], S_2[i_2], \ldots, S_k[i_k]), \\ F_{i_1,i_2-1,\ldots,i_k-1}, & +P(-, S_2[i_2], \ldots, S_k[i_k]), \\ F_{i_1-1,i_2,\ldots,i_k-1}, & +P(S_1[i_1], -, \ldots, S_k[i_k]), \\ \quad\vdots & \quad\vdots \\ F_{i_1,i_2,\ldots,i_k-1}, & +P(-, -, \ldots, S_k[i_k]), \\ \quad\vdots & \quad\vdots \quad . \end{cases}$$

Funkciju sličnosti za jedan stupac poravnanja definiramo sa:

$$P(S_1[i_1], S_2[i_2], \ldots, S_k[i_k]) = \sum_{1 \le p < q \le k} \delta(S_p[i_p], S_q[i_q]).$$

Ocjena složenosti

U osnovnom N–W algoritmu koristimo matricu dimenzija $n_1 \times n_2$ za spremanje rezultata manjih potproblema, a to onda daje prostornu složenost $O(n_1 n_2)$. Istim zaključivanjem dolazimo do prostorne složenosti poopćenog algoritma koja je $O(n_1 n_2 \ldots n_k)$, a ako uzmemo kao pretpostavku da su nizovi koje poravnavamo približno iste duljine n, imamo prostornu složenost $O(n^k)$.

Što se vremenske složenosti tiče, osnovni algoritam N–W ima složenost $O(n^2)$ za popunjavanje matrice, jer za svako od n^2 mjesta u matrici računamo 3 vrijednosti. Složenost za rekonstrukciju poravnanja je $O(n)$, što daje ukupnu vremensku složenost $O(n^2)$. Ova složenost vrijedi u slučajevima korištenja konstantnog ili linearnog kažnjavanja praznina,

dok kod afinog ili logaritamskog kažnjavanja, za svaki element moramo računati 3 vrijednosti, koje mogu ovisiti o još najviše $n - 1$ elemenata matrice. Tako dobivamo vremensku složenost $O(n^3)$.

Prošireni N–W algoritam, uz ponovnu pretpostavku o sličnoj duljini nizova za poravnanje, ima vremensku složenost $O(k^2 2^k n^k)$, jer za popuniti svako od n^k mjesta u matrici trebamo izračunati 2^k različitih vrijednosti, od kojih svako računanje zahtijeva k^2 operacija kako bismo izračunali vrijednost funkcije sličnosti za stupac poravnanja P.

Zaključak

Iako ovaj algoritam u konačnici dobiva globalno optimalno rješenje i daje sva moguća poravnanja koja su globalno optimalna, vremenska složenost je eksponencijalna s obzirom na broj nizova, i zato je vrijeme izvršavanja predugo već za $k > 5$, tj. poravnavanje više od 5 nizova. Štoviše, pokazalo se da je pronalaženje globalno optimalnog rješenja na ovaj način **NP–potpun** problem.

2.2 Aproksimacijski algoritmi

Aproksimacijski algoritmi oslanjaju se na heuristiku pri poravnavanju nizova. Njihova rješenja u pravilu su polinomne složenosti i ne garantiraju globalnu optimalnost te jako ovise o odabiru dobre heuristike, ali i o sličnosti među nizovima koje poravnavaju. Ako se problem poravnanja aproksimacijskim algoritmom gleda kao problem minimizacije funkcije udaljenosti, tada se svakom aproksimacijskom algoritmu pridaje **aproksimacijski koeficijent** α. Ako $D(A)$ označava vrijednost funkcije udaljenosti dobivenog poravnanja A, a $D(A_{opt})$ vrijednost funkcije udaljenosti optimalnog poravnanja, tada se može zapisati:

$$D(A) \leq \alpha D(A_{opt}), \quad \alpha \geq 1$$

.

Odavde je jasno da aproksimacijski koeficijent daje ograničenje na najgori mogući rezultat dobiven tim algoritmom. Također, jasno je da algoritam kojemu je aproksimacijski koeficijent bliži 1, u prosjeku daje kvalitetnija rješenja.

U nastavku ćemo opisati neke od aproksimacijskih algoritama.

Zvjezdasti algoritam

Aproksimacijski algoritam koji za odabrani *središnji niz* poravnava sve ostale nizove samo s njim i tako stvoreno veliko poravnanje daje kao konačno rješenje, naziva se *zvjezdasti algoritam*. Neka je $D(S_1, S_2)$ iznos funkcije udaljenosti optimalnog poravnanja nizova S_1 i S_2.

Algoritam radi u slijedećim koracima:

- Određuje se $D(S_i, S_j), \forall i, j$ (vidi sliku 2.2.b.).
- Odabire se središnji niz S_c iz S takav da S_c minimizira $\sum_{i=1}^{k} D(S_c, S_i)$ (vidi sliku 2.2.c.).
- Za svaki $S_i \in S \setminus S_c$ odabire se optimalno poravnanje nizova S_c i S_i (vidi sliku 2.2.d.).
- Dodavanjem znakova praznina u S_c stvaramo poravnanje svih nizova M takvo da zadovoljava sva pojedinačna poravnanja iz prethodnog koraka (vidi sliku 2.2.e te sliku 2.3.).

Za odabir središnjeg niza ovog algoritma gotovo uvijek se koristi jedan od sljedeća dva kriterija:

- za svaki od danih nizova se algoritam jednom izvrti uz pretpostavku da je baš taj niz središnji, a na kraju se kao rješenje izabere najbolje među dobivenim poravnanjima;
- za svaki par nizova izračuna se njihovo poravnanje (npr. dinamičkim programiranjem), a kao središnji niz S_c se zatim odabere onaj niz koji minimizira

$$\sum_{i \neq c} D(S_i, S_c).$$

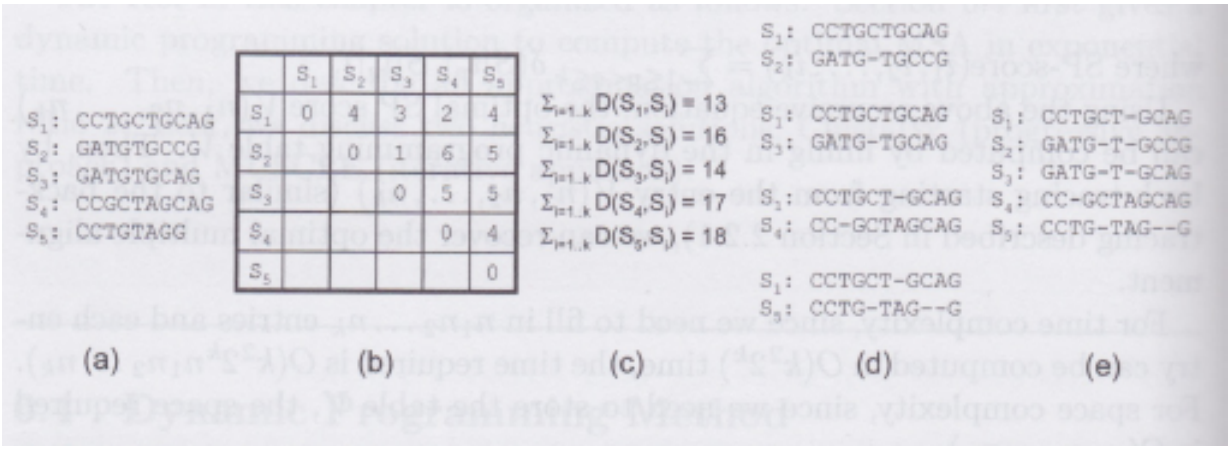
Sljedeći teorem daje aproksimacijski koeficijent zvjezdastog algoritma [5].

Teorem 2.2.1. *Aproksimacijski koeficijent zvjezdastog algoritma za poravnanje k nizova, uz funkciju udaljenosti D je* $2 - \frac{2}{k}$.

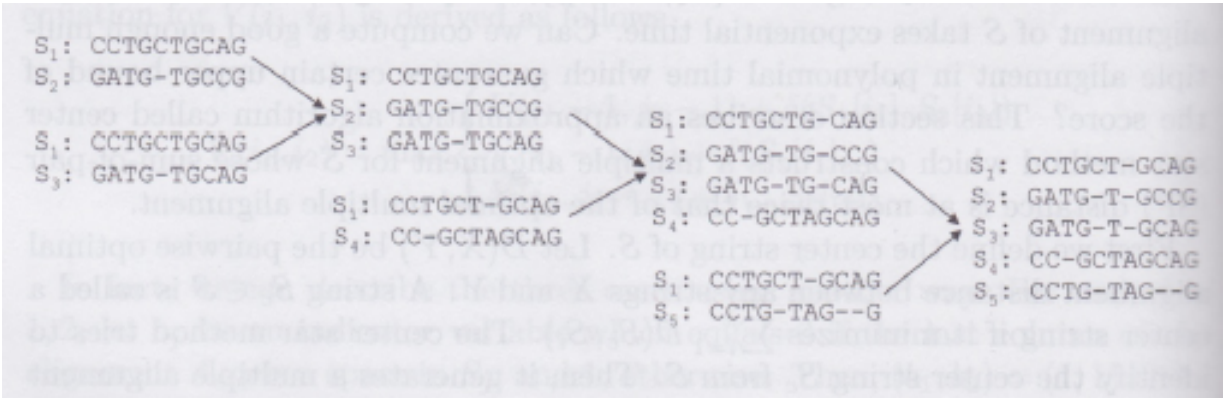
Ako pretpostavimo da su svi početni nizovi približno jednake duljine n, a središnji niz biramo kao onaj koji minimizira gore definiranu funkciju, tada možemo izračunati vremensku složenost algoritma:

- računanje funkcija udaljenosti za svaki par početnih nizova (npr. korištenjem Needleman–Wunschovog algoritma) je složenosti $O(k^2n^2)$,
- određivanje središnjeg niza je složenosti $O(k^2)$,
- generiranje $k - 1$ poravnavanja središnjeg niza S_c sa svim preostalima je složenosti $O(kn^2)$,
- ubacivanje znakova praznina u središnji niz kako bi sva prethodna poravnanja bila zadovoljena je složenosti $O(k^2n)$.

U konačnici nas to dovodi do ukupne složenosti algoritma $O(k^2n^2)$.



Slika 2.2: a) Početni nizovi za poravnanje uz bodovanje: -1 za nepoklapanje aminokiselina i za svaki znak praznine, a 1 za svako poklapanje, b) Matrica udaljenosti svih parova poravnanja, c) Odabir niza S_1 kao središnjeg, d) Poravnanje središnjeg niza sa svim ostalima, e) Dodavanje znakova praznina i stvaranje konačnog poravnanja svih nizova



Slika 2.3: Primjer dobivanja konačnog poravnanja svih nizova, ako znamo poravnanja središnjeg niza sa svim ostalim nizovima

Poopćenje zvjezdastog algoritma

Poopćenje ovog algoritma predložio je Bafna 1996. godine, a naziva se L–zvjezdasti algoritam. Osnovna ideja algoritma je:

1. odabrati središnji niz S_c,
2. razdijeliti k nizova u više grupa, od kojih svaka sadržava L nizova, među kojima je i S_c,

3. poravnati L nizova u svakoj grupi zvjezdastim algoritmom s obzirom na središnji niz S_c,

4. iskoristiti poravnanja svih grupa da bi se dobilo konačno poravnanje svih nizova. U ovom koraku, referentni niz je opet S_c.

Pokazalo se da je najveći problem pronaći način particioniranja k nizova. Ipak, sljedeći teorem pokazuje da je moguće dobiti vremensko poboljšanje u odnosu na obični zvjezdasti algoritam.

Teorem 2.2.2. *Postoji aproksimacijski algoritam aproksimacijskog koeficijenta* $2 - \frac{L}{k}$ *vremenske složenosti* $O(k^{L+1}(2^k + k \cdot g(L, m)))$*, pri čemu* $g(L, m)$ *označava vrijeme potrebno za optimalno poravnanje L nizova duljine m.*

2.3 Heurističke metode

Progresivno poravnavanje

Algoritmi progresivnog poravnavanja su daleko najkorišteniji tipovi algoritama za rješavanje problema poravnanja više nizova. Oni se oslanjaju na heurističko pretraživanje nizova koje se još naziva i *progresivna tehnika*, a prvi su ih razvili Hogeweg i Hasper, 1984. godine [4]. Ideja progresivnih algoritama je izgradnja konačnog poravnanja korištenjem poravnavanja parova nizova i to na način da se prvo poravnaju dva najsličnija niza, a zatim se ostali nizovi redom poravnavaju s tim inicijalnim poravnanjem, od boljeg prema lošijem. Svaki progresivni algoritam se sastoji od 2 dijela:

- reprezentacija međusobne sličnosti računanjem stabla sličnosti
- progresivno dodavanje nizova u poravnanje s obzirom na stablo sličnosti.

Za računanje stabla sličnosti koriste se neki od poznatih algoritama kao što su *algoritam združivanja susjeda (engl. neighbour-joining)* ili *UPGMA algoritam*, koji se oslanjaju na broj podudaranja podnizova od po dvije aminokiseline među proteinskim nizovima kao mjeru sličnosti/udaljenosti. Kao i ostali aproksimacijski algoritmi, ni progresivni algoritmi ne garantiraju pronalazak globalno optimalnog rješenja, a veliki razlog za to leži u činjenici da se potencijalna greška u poravnavanju napravljena na samom početku (stvoreno globalno neoptimalno poravnanje), do kraja postupka više ne može promijeniti, već se propagira skroz do završnog poravnanja. Upravo je to osnovna značajka progresivnih algoritama, *efikasnost ispred kvalitete*.

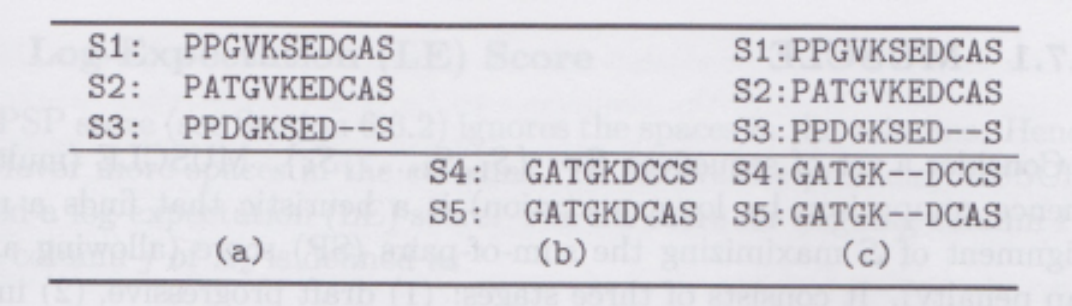
Najpoznatiji primjer jednog algoritma progresivnog poravnavanja je **ClustalW** algoritam [8], koji je možda danas i najkorišteniji algoritam za poravnavanje proteinskih nizova ili genetskih sekvenci proizvoljne duljine.

ClustalW

Uz to što je najkorišteniji algoritam za poravnavanje više nizova, ClustalW je i prilično jasan u svojoj ideji. Bitne promjene u odnosu na dosad obrađene algoritme je uvođenje drugačijeg načina mjerenja udaljenosti među nizovima te profil-profil poravnavanja [10].

Definicija 2.3.1. *Neka je* $S = S_1, S_2, \ldots, S_k$ *skup od k nizova koje treba poravnati. Ako je* M_1 *optimalno poravnanje prvih i nizova, a* M_2 *optimalno poravnanje preostalih* $k - i$ *nizova tada proces poravnavanja* M_1 *i* M_2 *nazivamo* ***profi-profil poravnanje****.*

Primjer profil-profil poravnanja prikazan je na slici 2.4.

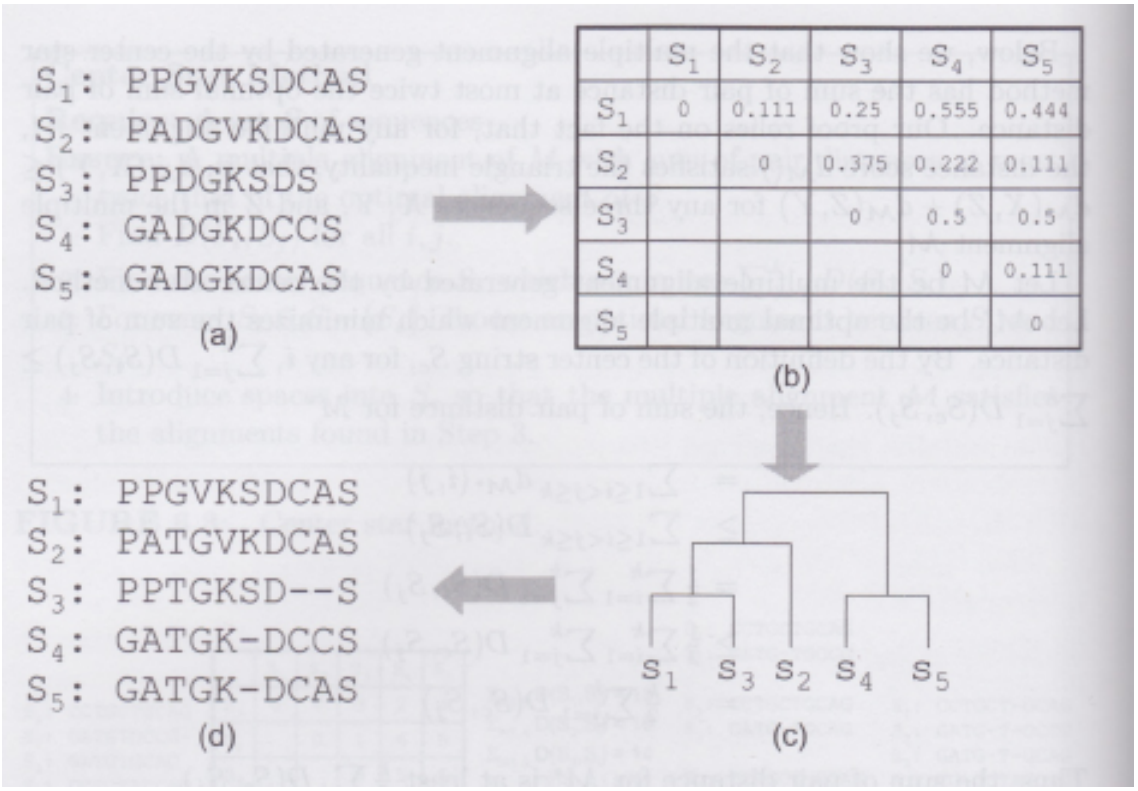


Slika 2.4: Primjer profil-profil poravnanja, tj. poravnanja dva manja, već izračunata, poravnanja

Osnovni koraci ClustalW algoritma grafički su prikazani na primjeru na slici 2.5. Za dani skup nizova $S = S_1, S_2, \ldots, S_k$, svaki duljine $O(n)$,

- izračunavamo globalno optimalno poravnanje za svaka dva niza S_i, S_j; tada postavljamo vrijednost funkcije udaljenosti između S_i i S_j na $1 - \frac{y}{x}$, pri čemu je x broj pozicija bez znakova praznine u oba niza, a y je broj pozicija podudaranja. Vremenska složenost ovog koraka je $O(k^2n^2)$;
- gradimo stablo sličnosti iz matrice udaljenosti, korištenjem algoritma združivanja susjeda. Vremenska složenost ovog koraka je $O(k^3)$;
- izvodimo progresivna poravnanja prateći generirano stablo sličnosti. U ovom koraku se događa najviše k profil-profil poravnanja vremenske složenosti $O(k^2n + n^2)$ pa je ukupna vremenska složenost ovog koraka $O(k^2n^2 + kn^2)$.

Sada vrlo lako zaključujemo da je ukupna vremenska složenost ovog algoritma $O(k^2n^2+k^3)$.



	S_1	S_2	S_3	S_4	S_5
S_1	0	0.111	0.25	0.555	0.444
S_2		0	0.375	0.222	0.111
S_3			0	0.5	0.5
S_4				0	0.111
S_5					0

Slika 2.5: a) Nizovi koje želimo poravnati, b) Matrica udaljenosti svih nizova u parovima, c) Dobiveno stablo sličnosti korištenjem algoritma združivanja susjeda, d) Konačno poravnanje nakon postupka progresivnog poravnavanja nizova prateći stablo sličnosti

Iterativni algoritmi

Iterativni algoritmi su svojevrsno poboljšanje algoritama progresivnog poravnavanja. Naime, u iterativnim algorimima jedan korak algoritma sastoji se od popravljanja trenutnog poravnanja i dodavanja novog niza u dosadašnje poravnanje. Na taj način se smanjuje utjecaj prvog poravnanja na konačni rezultat, koji je kod progresivnih algoritama bio golem. Za razliku od toga, ovdje se greška nastala u ranoj fazi poravnavanja ima mogućnost ispraviti do kraja postupka. Upravo se prema načinu korištenja otprije izračunatih djelomičnih poravnanja razlikuju neki od najpoznatijih iterativnih algoritama.
To su:

PRRN/PRRP, koji koristi algoritam ”uspona na vrh” (engl. Hill climbing) za optimizaciju vrijednosti poravnanja te iterativno popravlja težinske faktore nizova i ”rupičaste” dijelove poravnanja;

DIALIGN, koji se bazira na optimiziranju lokalnih poravnanja bez korištenja kažnjavanja praznina;

MUSCLE, koji se oslanja na precizniju mjeru udaljenosti među dvama nizovima u svrhu preciznijeg određivanja stabla sličnosti.

2.4 Skriveni Markovljevi modeli

Definicija 2.4.1. ***Skriveni Markovljev model*** *je statistički Markovljev model, koji pretpostavlja da je problem koji se modelira Markovljev proces sa skrivenim stanjima.*

Algoritmi koji se koriste skrivenim Markovljevim modelima, pridavanjem vjerojatnosti svim mogućim kombinacijama praznina, podudaranja ili nepodudaranja, određuju najvjerojatnije poravnanje. Ti algoritmi mogu dati globalno, ali i samo lokalno, optimalno rješenje. Također, njima možemo izračunati jedno najbolje poravnanje, ali i izgenerirati familiju mogućih poravnanja.

Algoritmi bazirani na skrivenim Markovljevim modelima reprezentiraju poravnanje više nizova kao usmjereni aciklički graf u kojem jedna konfiguracija stupca u poravnanju predstavlja jedan čvor u grafu. Najpoznatija metoda za poravnanje više nizova, bazirana na skrivenim Markovljevim modelima je *Viterbijev algoritam*. Sam Viterbijev algoritam je, također, i algoritam dinamičkog programiranja. Najčešće se koristi na način da se nizovi dodaju u postojeće poravnanje, jedan po jedan, a zatim se u svakom koraku tako dobiveno rješenje dodatno popravlja koristeći Markovljev vjerojatnosni model. Ovaj algoritam je, kao i svi progresivni algoritmi, dosta osjetljiv na redoslijed dodavanja novih nizova u poravnanje i uglavnom je limitiran na poravnanja 100 ili više nizova. To se u praksi pokazao kao broj za koji korišteni skriveni Markovljev model može sa dovoljnom preciznošću pridavati vjerojatnosti konfiguracijama nizova.

2.5 Genetski algoritmi

Genetski algoritmi su skupina algoritama koja svoju osnovnu ideju vuče iz biologije. Naime, u svakom genetskom algoritmu jedno moguće rješenje problema predstavlja jednu biološku **jedinku**. Svaka takva jedinka pripada skupini rješenja (jedinki) koje tvore trenutnu **populaciju**. Kako bi se pokušalo dobiti optimalno rješenje, početna populacija jedinki podvrgnuta je **evolucijskom procesu**. Spomenuti evolucijski proces sastoji se od iterativnog poboljšavanja trenutne populacije. U svakom koraku evolucije izabiru se jedinke koje nisu dovoljno dobre, te se njih mijenja novim jedinkama, koje se mogu dobiti bilo križanjem nekih, dovoljno dobrih, postojećih jedinki, ili direktnom promjenom na jednoj od, dovoljno dobrih, postojećih jedinki. Upravo je spomenuti proces evolucije motivaciju

našao u biološkoj evoluciji u kojoj, također, najbolje i najspremnije jedinke ostaju i stvaraju potomstvo, a one slabije odumiru. Ovi operatori nad jedinkama su analogni biološkim događajima križanja i mutacije i zaduženi su za stvaranje novih jedinki uz očuvanje svojstava roditeljskih jedinki. Populacija jedinki nakon svake pojedine iteracije čini novu **generaciju**, a početna populacija se još naziva i nultom generacijom.

Kako svi genetski algoritmi u sebi imaju određen faktor slučajnosti, o kojemu ovisi kvaliteta sljedećeg rješenja, oni ne garantiraju pronalazak globalno optimalnog rješenja. Uvjet zaustavljanja daljnjih iteracija i odabira konačnog rješenja, najčešće se definira kao određeni broj iteracija u kojima se populacija nije značajnije popravila. Veličina same populacije, također, predstavlja značajan faktor za uspješnost algoritma, ali i za brzinu izvođenja.

U sljedećem poglavlju bit će detaljno opisan jedan način rješavanja problema poravnanja više nizova genetskim algoritmom.

Poglavlje 3

Poravnanje više nizova genetskim algoritmom

3.1 SAGA

SAGA ili engleski, Sequence Alignment by Genetic Algorithm, je najpoznatiji i najkorišteniji genetski algoritam za poravnavanje više nizova. U njemu jedno poravnanje početnih nizova predstavlja jednu jedinku, a više poravnanja predstavlja populaciju. Tako evolucijsku fazu, tj. križanja i mutacije, predstavljaju operatori koji određene komade poravnanja međusobno križaju, mijenjaju, nadopunjuju... U svojoj srži, ovaj algoritam se oslanja na kvalitetno odabranu ciljnu funkciju poravnanja – u ovom slučaju to je jedna verzija funkcije sličnosti, te na optimizaciju poravnanja korištenjem genetskog algoritma.

Odabir ciljne funkcije

Kao i kod svih ostalih algoritama za poravnanje više nizova, odabir ciljne funkcije ima veliku ulogu u dobivenom konačnom rezultatu. U genetskom algoritmu ta je uloga još i veća, s obzirom na to da se algoritam uvelike oslanja na kvalitetu ciljne funkcije. U ovom primjeru genetskog algoritma, za izračunavanje ciljne funkcije, svakom paru nizova dan je težinski faktor, i izračunata je vrijednost njihova poravnanja za određen način bodovanja poklapanja/nepoklapanja aminokiselina te kažnjavanja praznina. Ovako opisanu ciljnu funkciju D, poravnanja A, može se zapisati u obliku:

$$D(A) = \sum_{i=2}^{N} \sum_{j=1}^{i-1} W_{i,j} \cdot \delta(A_i, A_j),$$

pri čemu δ označava funkciju sličnosti poravnanja dva niza A_i i A_j, a $W_{i,j}$ označava njihov težinski faktor. Kažnjavanje praznina vrši se unutar funkcije δ.

Ovakva definicija dovodi do velikog broja mogućih ciljnih funkcija zbog različitih načina određivanja svake od uključenih komponenti.

Težinski faktori $W_{i,j}$ mogu se zadati kao konstantna vrijednost za sve parove i, j, ali se mogu i posebno zadati iz stabla sličnosti, u svrhu smanjivanja količine redundantnih informacija s obzirom na evolucijsku povezanost nekih nizova. Još jedna mogućnost je izračunavanje kao u ClustalW algoritmu. Nakon izračunavanja stabla sličnosti, svakom se nizu pridaje zasebna težina, a težinski faktor para nizova onda je dobiven kao umnožak tih zasebnih težina.

Način bodovanja ovisi o odabiru supstitucijske matrice koja će se koristiti u samom bodovanju. Od najpoznatijih kategorija supstitucijskih matrica, izabrati se može između PAM i BLOSUM matrica.

Za kažnjavanje praznina uglavnom se odabire afino kažnjavanje, iako se može koristiti i kvazi-afino kažnjavanje. Ove dvije mogućnosti razlikuju se u načinu kažnjavanja "ugniježđenih" praznina, tj. situacija u kojima je praznina u jednom nizu potpuno sadržana u praznini drugog niza. Iako nijedno od navedenih kažnjavanja ne ubraja pozicije na kojima u oba niza stoji znak praznine, kvazi-afino kažnjavanje još dodatno kažnjava manji niz kaznom započinjanja praznine u slučaju ugniježđenih praznina.

Verzija genetskog algoritma, obrađena ovom radu, koristi ciljnu funkciju koja se oslanja na afino kažnjavanje praznina, ClustalW način određivanja težina, te koristi PAM250 kao supstitucijsku matricu.

3.2 Koraci algoritma

Postupak algoritma svodi se na generiranje nasumične nulte generacije G_0, te na iterativno stvaranje sljedećih generacija, sve dok se uvjet zaustavljanja ne postigne. Tokom iteracija koriste se unaprijed definirani operatori za stvaranje novih jedinki. Ti operatori se odabiru prema dinamički dobivenim vjerojatnostima. Dio operatora predstavlja operatore križanja, a dio operatore mutacija i time omogućuje stvaranje svih mogućih konfiguracija poravnanja nizova. Slijedi opis procesa po koracima u pseudo-kodu.

Inicijalizacija:

1. kreiranje nulte generacije G_0

Evaluacija:

2. izračunavanje ciljne funkcije za sve jedinke n-te generacije G_n

3. ako je uvjet zaustavljanja postignut, GOTO 13.
4. odabir jedinki koje se uklanjaju
5. izračunavanje mjere očekivanog potomstva

Evolucija:

6. odabir roditelja iz G_n za nove jedinke u G_{n+1}
7. odabir operatora
8. stvaranje nove jedinke (potomka)
9. odluka zadržati ili odbaciti novostvorenu jedinku
10. GOTO 6. dok se nova generacija ne popuni
11. $n = n + 1$
12. GOTO 2.

Završetak:

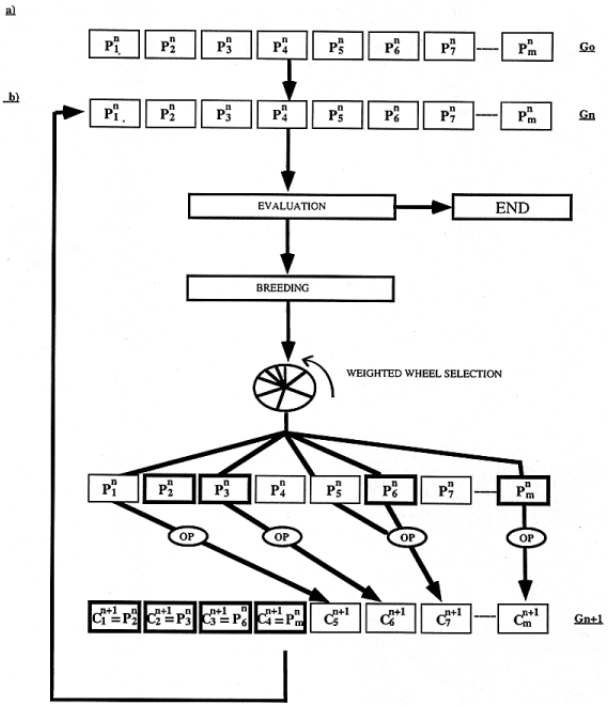
13. kraj.

Upravo opisani postupak može se vidjeti na slici 3.1.

Inicijalizacija

U ovom koraku se na nasumični način stvara nulta generacija veličine k jedinki. U implementaciji ovog algoritma, nulta, i svaka slijedeća generacija, bit će konstantne veličine i činiti će ih populacija od točno **10** jedinki. Kako bi se generirala jedna jedinka za nultu generaciju, za svaki niz koji se poravnava, na slučajan način se određuje veličina posmaka (engl. offseta) koja označava za koliko mjesta će se niz pomaknuti udesno u poravnanju. Posmak se bira u rasponu između 0 mjesta i $\frac{1}{4}$ ukupne veličine najduljeg niza u poravnanju.

Konačno se ispred svih nizova doda onoliko znakova praznina, koliki im je određeni posmak, a kako bi u konačnici svi nizovi bili jednake duljine, krajevi nizova se dopunjavaju znakovima praznina do konačne duljine. Na ovaj način dobivenih 10 jedinki, čini nultu generaciju iz koje će se u narednim koracima iterativno pokušati dobiti optimalno poravnanje.



Slika 3.1: Grafički prikaz genetskog algoritma: a) Nulta generacija, b) Jedna iteracija algoritma sa odabiranjem raditelja novih jedinki te operatora kojim ih se dobiva

Evaluacija

Prvi korak evaluacije je izračunavanje ciljne funkcije za svako poravnanje u populaciji. Nakon toga izračunavaju se mjere očekivanog potomstva za svaku jedinku. **Mjera očekivanog potomstva** je, najčešće, mali prirodni broj, a u ovom radu uzet je kao prirodni broj između 0 i 2. Njegovo određivanje vezano je uz iznos ciljne funkcije svakog poravnanja, a služi kasnije u svrhu odabira roditelja novih jedinki. Dobiva se izračunavanjem prosječne vrijednosti ciljne funkcije za sve jedinke trenutne populacije. Ako je s $D(A_i)$ označena ciljna funkcija i-te jedinke, a s D_{avg} prosječna vrijednost ciljne funkcije, tada se mjera očekivanog potomstva E, za jedinku i zadaje kao:

$$E(A_i) = \begin{cases} 0, & \dfrac{D(A_i)}{D_{avg}} < 1, \\ 1, & 1 \leq \dfrac{D(A_i)}{D_{avg}} < 2, \\ 2, & \dfrac{D(A_i)}{D_{avg}} \geq 2. \end{cases}$$

U sljedećem koraku određuju se one jedinke koje će se ukloniti i biti zamijenjene novima. Zamijeniti se mogu sve jedinke, ali u ovoj verziji algoritma, čuva se 50% populacije koja će postati dijelom nove generacije, a drugu polovinu nadopunjuje se novim jedinkama. Ova metoda zove se *metoda preklapajućih generacija*.

Evolucija

Nova generacija jedinki dobiva se na način, da se prvo najbolje jedinke trenutne generacije dodaju u novu, a zatim se, koristeći istih tih 50% najboljih jedinki, operatorima križanja i mutacija dobiju nove jedinke koje popunjavaju novu generaciju. Samom generiranju jedinki prethodi odabir operatora koji se koristi za kreiranje nove jedinke, te odabir jednog ili dva preostala poravnanja (ovisno o operatoru), nad kojima se izvršava odabrani operator i generira jedna nova jedinka.

Odabir roditeljskih jedinki (jedinki nad kojima se izvršava operacija) odvija se korištenjem *jednostavne (engl. roulette wheel) selekcije* s obzirom na izračunate mjere očekivanog potomstva. Jednostavna selekcija odvija se procesom vrtnje kotača ruleta s poljima koja označavaju jedinke, a svaka jedinka se nalazi na onoliko polja koliko iznosi njezina mjera očekivanog potomstva. Kada je jedna jedinka odabrana za roditelja, njezina polja se uklone s kotača. Postupak se ponavlja dok se ne odaberu sve roditeljske jedinke potrebne za izvođenje odabrane operacije, koja je, također, izabrana jednostavnom selekcijom s obzirom na dotadašnju uspješnost dotičnog operatora.

Završetak

Kako za genetske algoritme ne postoji dokaz da će u beskonačnom vremenu dati optimalno rješenje, potrebno je odrediti neki razumni uvjet zaustavljanja izvođenja algoritma. U ovom radu, za uvjet se uzima *stabilizacija* algoritma, tj. situacija u kojoj se određeni broj generacija dotad najbolje rješenje nije promijenilo. Najčešće se, za broj generacija potrebnih za stabilizaciju, uzima 100.

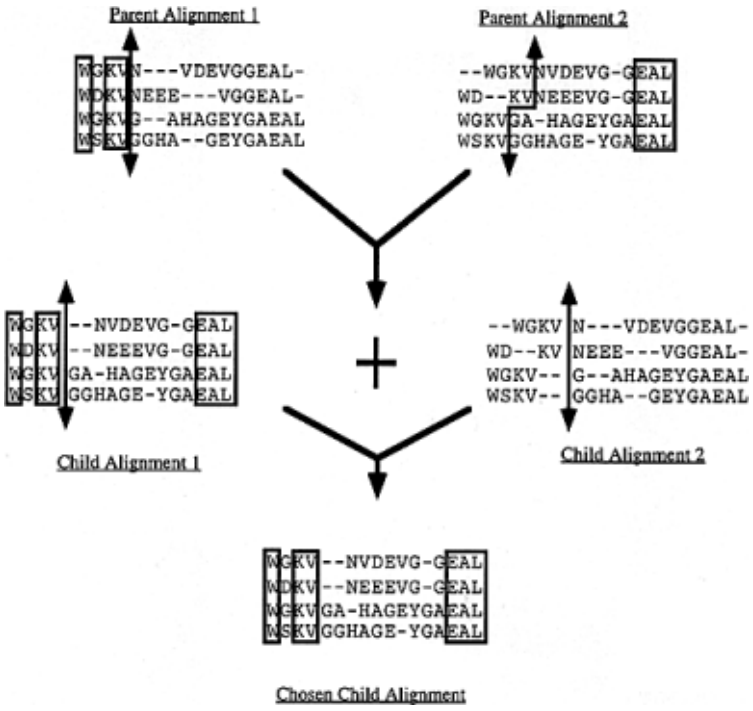
3.3 Operatori

Operatori križanja

Operatori križanja su zaduženi za stvaranje nove jedinke iz dvije roditeljske jedinke. U ovom radu koriste se dvije vrste operatora križanja, a to su **mjesno križanje** i **uniformno križanje**.

Mjesno križanje generira novu jedinku kroz točno jednu razmjenu dijelova niza između dvije roditeljske jedinke. U prvom poravnanju (roditelju) se na slučajan način odabere po-

zicija na kojoj će se poravnanje vertikalno razdvojiti, a zatim se drugo poravnanje "kroji" prema potrebi spajanja s dijelovima prvog. Naime, redoslijed svih aminokiselina u nizovima mora ostati nepromijenjen, ali se isto tako nijedna aminokiselina ne smije izgubiti križanjem. Kada su oba poravnanja razdijeljena, tada se križanjem lijevih i desnih dijelova dobivaju dvije nove jedinke koje zadovoljavaju uvjet očuvanja svih aminokiselina i njihovog redoslijeda. S obzirom da se iste aminokiseline u istim nizovima, ali različitim poravnanjima, ne moraju nalaziti na istom mjestu, to može dovesti do razlikovanja u duljinama nizova u dobivenim poravnanjima. Kako bi se to ispravilo, na spojnoj točki dvaju dijelova poravnanja, svaki niz se dopunjava s onoliko znakova praznina koliko je potrebno da se duljine svih nizova u jednom poravnanju izjednače. Primjer rada opisanog operatora može se vidjeti na slici 3.2.

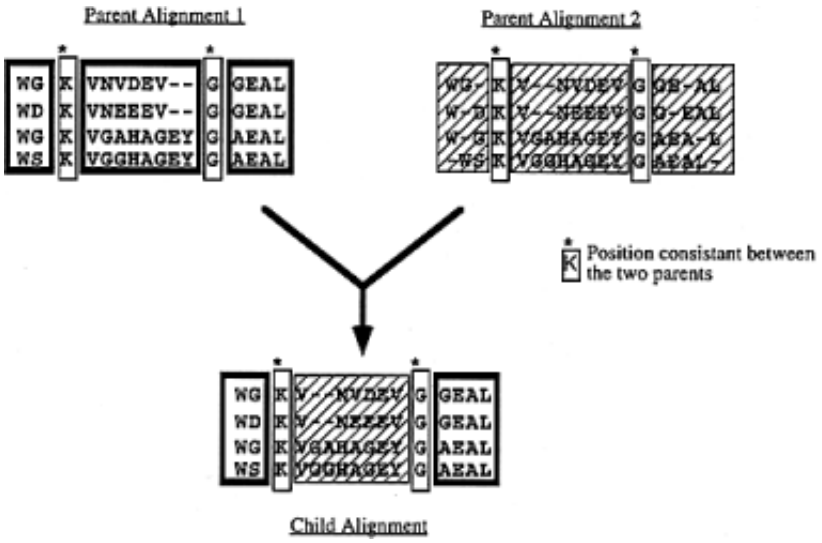


Slika 3.2: Grafički prikaz mjesnog križanja i odabira bolje od dvije dobivene jedinke

Upravo navedeno dopunjavanje nizova prazninama predstavlja najveći problem mjesnog križanja, jer time kvari strukturu poravnanja i, zbog kažnjavanja praznina, kvari vrijednost dobivenog rješenja. Iz tog razloga koristi se uniformno križanje, koje omogućuje dijeljenje roditeljskih jedinki na više mjesta i višestruku međusobnu razmjenu dijelova u svrhu stvaranja novih jedinki. Prvo se u oba roditeljska poravnanja mapiraju sve **konzistentne pozicije**. Konzistentne pozicije podrazumijevaju stupce u poravnanjima koji se u oba roditelja podudaraju po konfiguraciji.

Ako stupac jednog poravnanja ima konfiguraciju **ALA125**, a ista pozicija u drugom poravnanju ima konfiguraciju stupca **ALA101**, tada te pozicije nisu konzistentne. Kada bi pozicija u drugom poravnanju imala istu konfiguraciju (ALA125) kao i konfiguracija prvog poravnanja, tada bi se za te pozicije moglo reći da su konzistentne. To, naravno, vrijedi ako i samo ako su aminokiseline iz prve konfiguracije i druge konfiguracije na istim relativnim pozicijama u svojim nizovima, svaki u svom poravnanju (npr. ako je aminokiselina L iz prve konfiguracije 6. aminokiselina po redu u svome nizu u prvom poravnanju, tada aminokiselina L iz druge konfiguracije mora biti 6. po redu aminokiselina u svome nizu u drugom poravnanju). Primjer konzistentnih pozicija su uokvireni stupci na slici 3.3.

Nakon mapiranja konzistentnih pozicija u roditeljskim poravnanjima, blokovi među njima mogu se nesmetano zamijeniti između roditeljskih poravnanja i time se stvaraju dvije nove jedinke. Odabir blokova koji će se zamijeniti među jedinkama, a koji će ostati, može biti potpuno slučajan, ali se može i isprobati sve kombinacije te tako dobiti najbolje moguće rješenje. Metoda isprobavanja svih kombinacija i odabira najbolje naziva se metoda uspona na vrh (engl. Hill climbing).

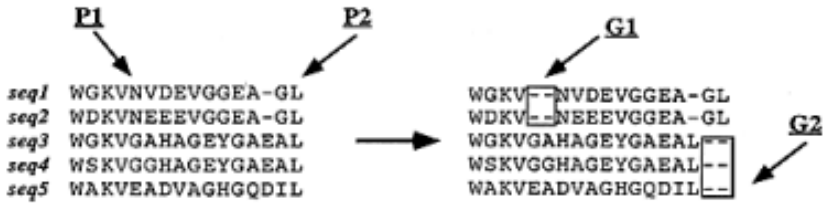


Slika 3.3: Grafički prikaz uniformnog križanja. Uokvireni stupci predstavljaju konzistentne pozicije.

Operator umetanja praznine

Iako operatori križanja mogu, kroz zamijene dijelova poravnanja, prenijeti razne uzorke poravnatih aminokiselina, operatori mutacija koriste se za generiranje tih uzoraka.

Prvi operator mutacije je operator umetanja praznine koji, za jedno dano poravnanje, proširuje to poravnanje prazninama. U svaki niz u poravnanju ubacuje se jedna praznina slučajno odabrane duljine. Kako ne bi svi nizovi u poravnanju imali prazninu na istom mjestu, nizovi su podijeljeni u dvije skupine i odabrane su dvije slučajne pozicije, prva (P1) potpuno slučajno, a druga (P2) slučajno, ali unutar najveće zadane udaljenosti od prve. Svim nizovima prve skupine umeće se praznina na prvu odabranu poziciju, a svim nizovima druge skupine na drugu odabranu poziciju, kao što je vidljivo na primjeru na slici 3.4.



Slika 3.4: Nizovi *seq1* i *seq2* čine prvu, a *seq3*, *seq4* i *seq5* drugu skupinu nizova, pa su im zato praznine ubačene na različita mjesta $P1$ i $P2$.

Postoji mogućnost odabira i umetanja praznine metodom uspona na vrh, pri čemu se sve varijable odabiru i dalje na slučajan način, osim pozicije P1, za koju se isprobaju sve mogućnosti i odabere ona za koju dobivena jedinka ima najbolju ciljnu funkciju.

Operatori repozicioniranja praznina

Kako operator umetanja praznine u sebi sadrži dosta varijabli koje ovise o čistoj slučajnosti, ponekad može doći do situacije u kojoj se praznina umetne na neko mjesto u poravnanju, a malim, čovjeku očiglednim, modifikacijama bi se uvelike mogao popraviti rezultat poravnavanja. Tada se koriste operatori repozicioniranja praznina.

Blok praznina je neki skup praznina u nizovima jednog poravnanja koji imaju barem jedno stupčano preklapanje, tj. sve praznine koje u nekom slučajno odabranom stupcu poravnanja imaju znak praznine. Navedeni operatori koriste se nad nekim blokom praznina.

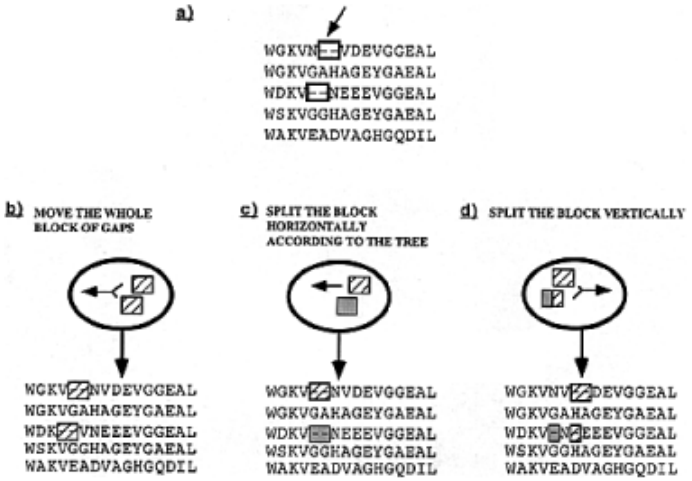
Od operatora repozicioniranja koriste se:

operator posmaka bloka praznina, koji odabrani blok praznina pomiče udesno ili ulijevo, koristeći metodu uspona na vrh, kako bi generirao novu jedinku

operator horizontalne razdiobe bloka praznina, koji za odabrani blok praznina odabire nizove čije praznine će tvoriti podblok praznina koji će se pomicati ulijevo ili udesno, korištenjem metode uspona na vrh, i tako generirati novu jedniku

operator vertikalne razdiobe bloka praznina, koji za odabrani blok praznina odabire poziciju unutar dosega bloka i na tom mjestu vertikalno odsijeca blok u dva podbloka, od kojih se jedan odabire i pomiče u svom smjeru, koristeći metodu uspona na vrh, kako bi se generirala nova jedinka.

Svi spomenuti operatori mogu se vidjeti na primjeru na slici 3.5.



Slika 3.5: a) Primjer jednog bloka praznina, b) pomak cijelog bloka praznina za jedno mjesto ulijevo, c) horizontalno dijeljenje bloka praznina i pomak donjeg dijela za jedno mjesto udesno, d) vertikalno dijeljenje bloka praznina i pomak lijevog dijela bloka za jedno mjesto ulijevo

Pretraživanje po blokovima

Pretraživanje po blokovima je možda i najvažniji operator mutacije u genetskom algoritmu za poravnanje nizova. Iako operatori umetanja i repozicioniranja praznina nude mogućnost stvaranja raznih uzoraka, a operatori križanja nude mogućnost prenošenja tih uzoraka kroz generacije jedinki, zbog velikog broja varijabli koje se koriste u tim operatorima, a dobivene su na slučajan način, ovi operatori trebaju veliku količinu vremena i pokušaja kako bi uspjeli stvoriti uzorak koji je možda lako uočljiv iz nekog već stvorenog

poravnanja. Iz tog razloga koristi se ova heuristička metoda, koja u postojećem poravnanju lokalno pretražuje zajedničke motive u nizovima i na taj način značajno poboljšava i ubrzava proces rješavanja algoritma.

U ovom kontekstu, blok se definira kao kraći komad poravnanja koji ne sadrži znakove praznina. Operatoru se prvo zadaje podniz nekog od nizova iz roditeljskog poravnanja koji ne sadrži praznine. Duljina podniza je odabrana na slučajan način, iako se najčešće početni podniz ograničava na manji broj aminokiselina (najčešće, 5–15). Početna pozicija podniza je, također, odabrana na slučajan način. Nakon toga se, unutar unaprijed definiranih granica oko rubova odabranog podniza, on uspoređuje sa svim podnizovima preostalih nizova unutar tih granica i odabire se podniz koji daje najveću ciljnu funkciju za to poravnanje. Nakon toga se blok od dva podniza uspoređuje sa svim podnizovima preostalih nizova unutar granica te mu se pridodaje novi podniz, i tako redom, sve dok se u tom bloku ne nalazi po jedan podniz svakog niza iz roditeljskog poravnanja. Tada se cjelokupni nizovi pomaknu ulijevo ili udesno kako bi se reproducirao dobiveni blok, a na početak i na kraj se doda onoliko znakova praznina koliko je potrebno da svi nizovi u novonastalom poravnanju budu jednake duljine.

Odabir operatora

U samoj implementaciji algoritma bitan faktor je odabir operatora, a zbog velike količine iteracija potrebnih za dobivanje dobrog rješenja, moguć je slučaj da su određeni operatori dobri u početnim iteracijama, a u kasnijima postanu gotovo beskorisni, i obrnuto. Kako bi se ta mogućnost minimizirala, osim implementacije potpuno nasumičnog odabira sljedećeg operatora, implementiran je i slučaj kada se vjerojatnost odabira nekog operatora dinamički mijenja kroz iteracije, na sličan način kao što se mijenja odabir roditeljskih jedinki za generiranje novih jedinki. Prema tome, svaki operator ima šansu biti izabran proporcionalnu s kvalitetom zadnje jedinke koju je generirao, no svaki operator zadržava barem minimalnu šansu biti izabran, kako se neki operatori, koji bi mogli biti korisni u kasnijim fazama algoritma, ne bi izgubili na samom početku. Ovaj način odabira operatora se još naziva dinamički odabir operatora.

3.4 Implementacija

Svi opisani operatori križanja i mutacija implementirani su u programskom rješenju priloženom uz ovaj rad. Svi oni implementirani su uz korištenje metode uspona na vrh, ako to podržavaju, kako bi se potencijalno poboljšali rezultati algoritma. Na taj se način, naravno, dodatno izgubilo na brzini izvršavanja.

Programsko rješenje izvedeno je u potpunosti u programskom jeziku Java, uz korištenje standardnih biblioteka.

Poglavlje 4

Rezultati i zaključak

4.1 Rezultati

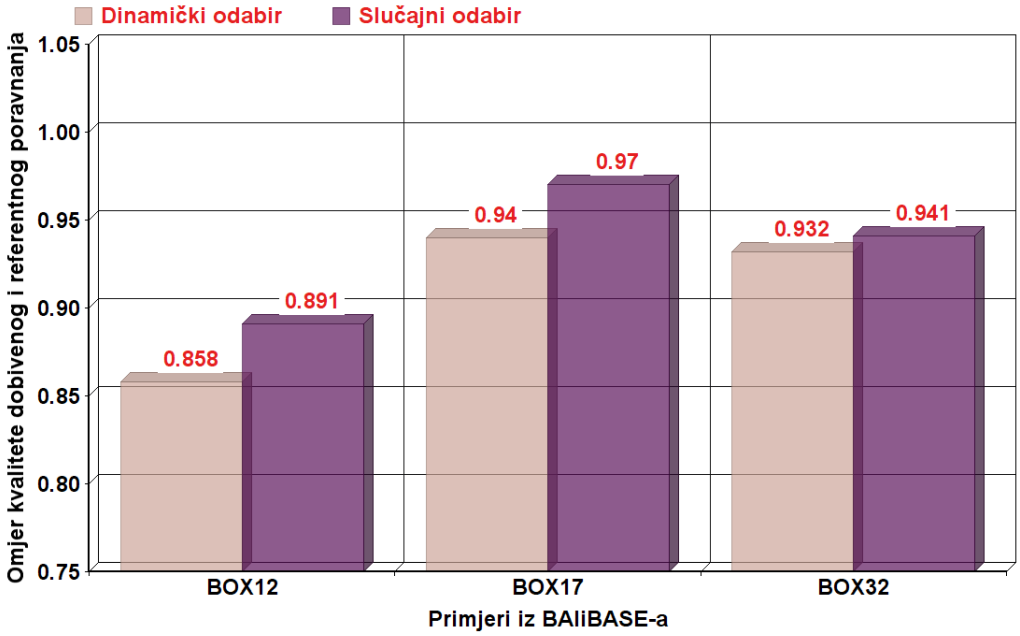
Kao baza podataka, korištena je **BAliBASE** (**B**enchmark **Ali**gnment Data**BASE**) [9], koja sadrži razne grupe proteinskih nizova sa različitim stupnjevima sličnosti. U ovom radu korištena je isključivo grupa **RV913**, tj. grupa proteinskih nizova sa 40–80% sličnosti. Ta grupa je korištena iz razloga što je poravnavanje manje sličnih nizova izuzetno osjetljivo na način odabira svih parametara koji se koriste u algoritmu, a određeni su unaprijed, na neku fiksnu vrijednost, pa se uz nedovoljno precizno odabrane vrijednosti parametara, dobiju rješenja koja su potpuno neupotrebljiva.

Upravo su načini odabira operatora, parametri kažnjavanja, te duljine i broj nizova koje treba poravnati, zanimljivi za analizu njihovog utjecaja na duljinu izvođenja algoritma, kao i na samu kvalitetu dobivenog rješenja.

Prva usporedba odnosi se na uspješnost algoritma s obzirom na način odabira operatora, bilo potpuno nasumičnim odabirom, ili dinamičkim odabirom. Iako su očekivanja bila drugačija, pokazalo se da bitnih razlika u samoj kvaliteti dobivenih rješenja nema, a duljina izvođenja algoritma čak je blago rasla u slučajevima dinamičkog odabira operatora. Dobivene rezultate iz grafa 4.1 može se opravdati činjenicom da je implementiran relativno mali broj operatora (njih 7). Pretpostavka je da pristranost pri izboru operatora koji će se koristiti nema velik utjecaj, budući da se vjerojatnost odabira operatora u nekoliko iteracija algoritma može u potpunosti izmijeniti, zbog malog ukupnog broja operatora.

Kako se u kasnijim fazama algoritma ukupna populacija počne stabilizirati, te se velike promjene više gotovo ne pojavljuju, operatori koji uspijevaju poboljšati rezultat, uglavnom, su operatori koji bitno ne mijenjaju strukturu (repozicioniranje praznina) pa algoritam "zaglavi" u lokalnom minimumu i jedini napredak koji se ostvaruje je potencijalno poboljšanje poravnanja do samog lokalnog minimuma. Ova tendencija algoritma koji koristi dinamički

odabir operatora, daje dobro opravdanje za sporiju konvergenciju algoritma u, ne nužno, najbolje rješenje.



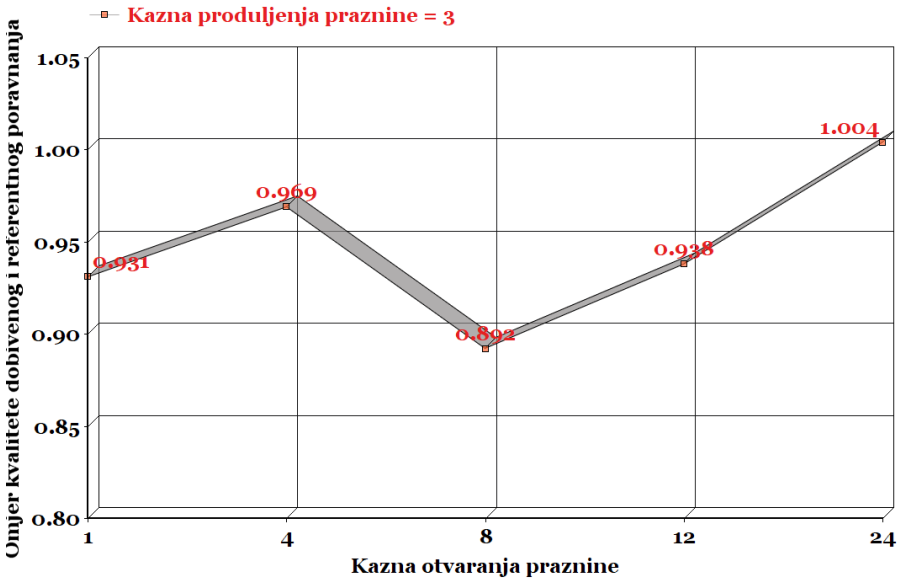
Slika 4.1: Usporedba rada algoritama s dinamičkim i slučajnim odabirom operatora

Zanimljivost koja se pojavila prilikom pokretanja algoritma na testnim slučajevima je ta, da se algoritam pokazao iznimno uspješan pri poravnanju manjeg broja (≤ 6) relativno kratkih nizova (≤ 250 znakova). U takvim uvjetima događale su se situacije kada bi ovako implementiran genetski algoritam ponudio rješenje za koje je iznos ciljne funkcije veći od vrijednosti za referentno poravnanje priloženo uz bazu podataka BAliBASE. To referentno rješenje dobiveno je ClustalW algoritmom, čiji izvorni kod je priložen uz bazu. Ova pojava je u skladu s rezultatima koje su, u originalnom članku, opisali autori ovog algoritma. Na sličnim poravnanjima, njihovi su rezultati u ponekim slučajevima bili bolji od onih dobivenih ClustalW algoritmom.

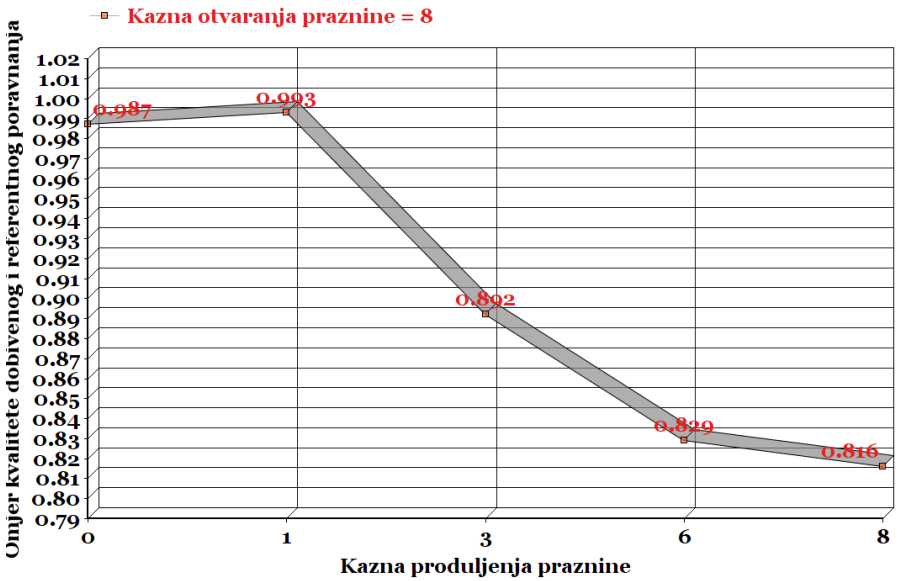
Kako je objašnjeno, sam način odabira operatora ne predstavlja važan faktor u izvođenju ovog algoritma, pa se za potrebe daljnje analize koristio samo nasumični odabir operatora.

Osim određivanja najboljeg načina za odabir operatora, analiziran je i način na koji različite vrijednosti parametara u afiinom kažnjavanju mogu utjecati na kvalitetu dobivenog rješenja. Standardno se u algoritmu koristilo 8 kao kaznu otvaranja praznine, te 3 kao kaznu

produljenja praznine. Na slikama 4.2a i 4.2b može se vrlo lako vidjeti koje su posljedice povećanja ili smanjenja bilo kojeg od ta dva parametra.



(a) Različite kazne otvaranja

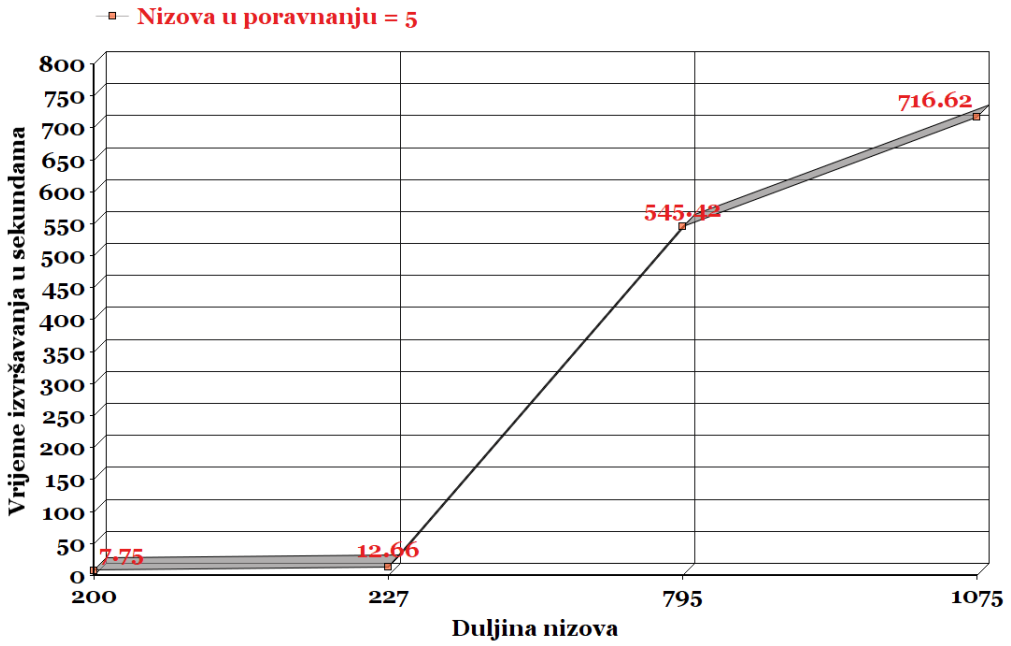


(b) Različite kazne produljenja

Slika 4.2: Analiza kvalitete alogritma za različite parametre afinog kažnjavanja praznina

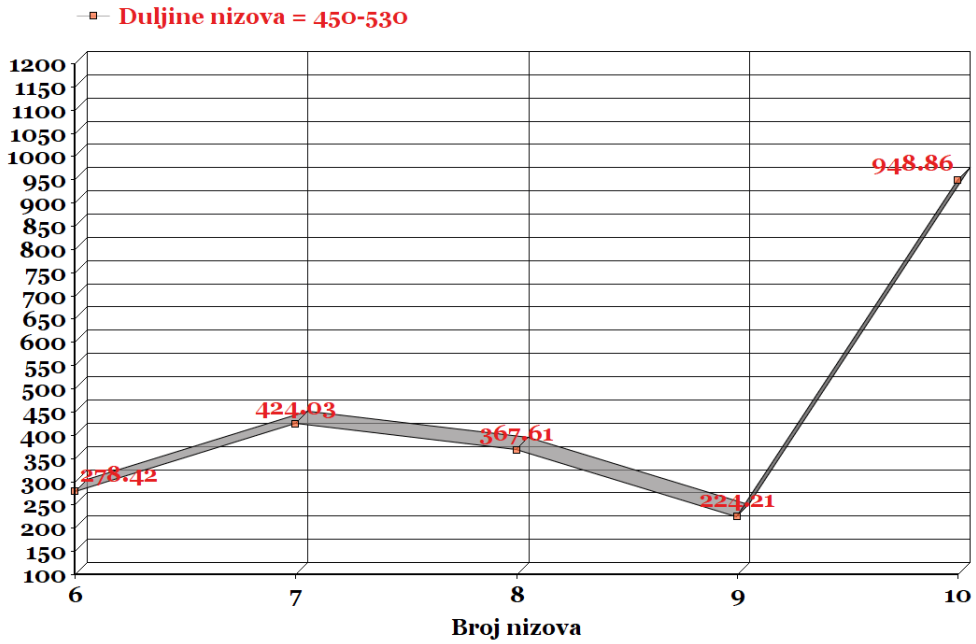
Iako je sam algoritam prilično stabilan za razne iznose kazne otvaranja praznine, pokazuje se da se povećanjem kazne produljenja niza osjetno gubi na kvaliteti konačnog rješenja. Ova situacija je očekivana, ako se u obzir uzme da se veća kazna produljenja niza obično povezuje s poravnavanjem nizova koji imaju manju sličnost, a u ovom radu koriste se nizovi koji imaju i do 80% sličnosti.

Analizirani su još i odnosi brzine izvođenja algoritma s duljinom nizova koje treba poravnati, te njihovim brojem. Dobiveni podaci mogu se vidjeti na slikama 4.3 i 4.4. U svrhu korektnosti analiziranja, pri analizi ovisnosti ukupnog vremena izvođenja o duljini nizova, korišteni su primjeri u kojima su duljine nizova različite, ali su svi primjeri poravnanja 5 nizova. Analogno, za analizu ukupnog vremena izvođenja o količini nizova za poravnavanje, odabrani su primjeri u kojima su nizovi za poravnanje približno jednakih duljina (450–530 aminokiselina). Iz grafa je očigledno da se algoritam značajno usporava kako broj nizova u poravnanju raste, a s obzirom da operatori koriste metodu uspona na vrh, koja je kvadratne složenosti, može se reći da je taj rezultat očekivan. Isti argument vrijedi i za usporavanje algoritma kod poravnavanja duljih nizova.



Slika 4.3: Trajanje algoritma za različite duljine nizova

Naposlijetku treba reći kako se testiranjem implementacije na raznim primjerima, s raznim zadanim parametrima, pokazalo kako je ponašanje algoritma dosta očekivano i za većinu parametara dosta stabilno. Međutim, ono što se još pokazalo je da generiranje



Slika 4.4: Trajanje algoritma za različit broj nizova

kvalitetnih jedinki u nultoj generaciji osjetno ubrzava konvergenciju algoritma, te, također, popravlja kvalitetu konačnog poravnanja. Stoga je očito da je to jedno bitno područje koje zahtijeva dodatnu analizu.

4.2 Zaključak

U analizi metoda poravnanja više nizova pokazalo se kako je često potrebno jasno definirati uvjete na raspoložive resurse (vremenske i prostorne) prije odabira metode rješavanja ovog problema. Tako se moglo vidjeti nekoliko algoritama koji žrtvuju kvalitetu poravnanja, kako bi dobili na brzini izračunavanja. Genetski algoritmi su u tom pogledu posve drugačiji. Njima se pokušava dobiti kompromis između kvalitete rješenja i vremena izvođenja. Kako je sama motivacija za razvoj genetskih algoritama dobivena iz postupka prirodne selekcije (evolucije), prilično se jasno nameće potreba pokušati pristupiti jednom ovakvom problemu iz prirode na taj način.

Genetski algoritam za poravnanje više nizova, predložen od strane Notredamea i Higginsa [7], a implementiran u ovome radu, pokazao se u svim testovima kao prilično uspješan algoritam s gledišta kvalitete poravnanja koje daje kao rezultat. Međutim, sitne heuristike, koje se koriste u operatorima mutacije i križanja u implementaciji algoritma, polinomne su

vremenske složenosti, pa vrijeme izvođenja algoritma ima tendenciju jako porasti za osjetnija povećanja zadanog skupa nizova. Ipak, postoji razlog za korištenje ovog algoritma u praksi, a taj je što predloženi algoritam u slučajevima manjih poravnanja može ponuditi rješenja koja su za nijansu bolja od rješenja dobivenih algoritmom ClustalW, danas najkorištenijim algoritmom za poravnanje više nizova.

S gledišta implementacije spomenutog algoritma, pokazalo se da je algoritam dosta robustan, i može ponuditi gotovo ista rješenja kakva su opisali autori u svome originalnom radu. Nakon analiziranja utjecaja većine varijabilnih elemenata implementacije na konačni uspjeh algoritma, pokazalo se da ima još prostora za njegovo poboljšanje, posebno u području odabira nulte generacije koja dosta utječe na ukupnu uspješnost algoritma. Također, ne treba zanemariti niti mogućnost razvijanja boljih ciljnih funkcija koje bi lakše vodile do rješenja, niti mogućnost korištenja ovog algoritma u kombinaciji s drugim, već postojećim, rješenjima poravnanja više nizova.

Bibliografija

[1] Hyrum Carroll, Mark J. Clement, Perry Ridge i Quinn O. Snell, *Effects of gap open and gap extension penalties*, (2006).

[2] Margaret O. Dayhoff i Robert M. Schwartz, *A model of evolutionary change in proteins*, In Atlas of protein sequence and structure, Citeseer, 1978.

[3] Steven Henikoff i Jorja G. Henikoff, *Amino acid substitution matrices from protein blocks*, Proceedings of the National Academy of Sciences **89** (1992), br. 22, 10915–10919.

[4] P. Hogeweg i B. Hesper, *The alignment of sets of sequences and the construction of phyletic trees: an integrated method*, Journal of molecular evolution **20** (1984), br. 2, 175–186.

[5] Tao Jiang, Ying Xu i Michael Q. Zhang, *Current topics in computational molecular biology*, MIT Press, 2002.

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

[7] Cédric Notredame i Desmond G. Higgins, *SAGA: sequence alignment by genetic algorithm*, Nucleic acids research **24** (1996), br. 8, 1515–1524.

[8] Julie D. Thompson, Desmond G. Higgins i Toby J. Gibson, *CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice*, Nucleic acids research **22** (1994), br. 22, 4673–4680.

[9] Julie D. Thompson, Frédéric Plewniak i Olivier Poch, *BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs.*, Bioinformatics **15** (1999), br. 1, 87–88.

[10] Guoli Wang i Roland L. Dunbrack, *Scoring profile-to-profile sequence alignments*, Protein Science **13** (2004), br. 6, 1612–1626.

Sažetak

Problem poravnanja više nizova svoju svrhu nalazi u svakodnevoj upotrebi i stoga je od velikog interesa imati na raspolaganju dovoljno dobre metode rješavanja tog problema. Kako je sam problem poravnanja više nizova prilično složen i egzaktno rješavanje u praksi ne dolazi u obzir, u ovom radu dan je pregled poznatih metoda rješavanja, a poseban naglasak stavljen je na metodu rješavanja problema genetskim algoritmom. Provedena je analiza kvalitete dobivenih rješenja s obzirom na određivanje nekih varijabli u samom algoritmu, te je na kraju opisan smjer u kojem bi se potencijalno moglo krenuti u daljnjem istraživanju ove problematike.

Summary

Multiple sequence analysis is a problem which is encountered daily, therefore, it is very important to have good methods for solving this problem. Because solving multiple sequence alignment problem using dynamic programming would have no practical application, due to complexity of the problem itself, this paper gives some of the most popular methods for solving this problem with emphasis on genetic algorithms. Result quality analysis was made with respect to different methods for assessing various parameters included in the algorithm. Finally, some possibilities for future improvements were discussed.

Životopis

Neven Grubelić rođen je 24. kolovoza 1991. u Zagrebu, gdje je odličnim uspjehom završio osnovnu školu Tituša Brezovačkog i V. gimnaziju. Preddiplomski sveučilišni studij matematike na Matematičkom odsjeku Prirodoslovno–matematičkog fakulteta Sveučilišta u Zagrebu upisao je 2010. godine. Na istom fakultetu 2013. godine dobio je titulu sveučilišnog prvostupnika matematike, te je upisao diplomski studij Računarstva i matematike.