Univerzitet u Beogradu
Matematički fakultet

Paralelizacija algoritma za predikciju savijanja proteina u HP modelu zasnovanom na koloniji mrava

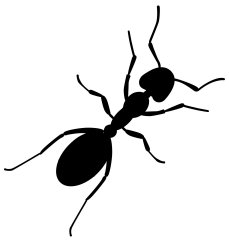
Master rad

Luka Matijević

Beograd, Septembar 2017.

Univerzitet u Beogradu
Matematički fakultet
MASTER RAD

Autor:	**Luka Matijević**
Naslov:	**Paralelizacija algoritma za predikciju savijanja proteina u HP modelu zasnovanom na koloniji mrava**
Mentor:	**dr Saša Malkov, Matematički fakultet**
Komisija:	**dr Miodrag Živković, Matematički fakultet** **dr Aleksandar Kartelj, Matematički fakultet**



Sažetak

Ovaj rad bavi se paralelizacijom algoritma za predikciju savijanja proteina zasnovanog na koloniji mrava, koristeći grafičku karticu kao izvor paralelizacije. U njemu su date osnovne informacije o proteinima, njihovom savijanju, *ACO* metaheuristici i paralelizaciji, dok su u zadnjem delu prikazani rezultati dobijeni testiranjem programa. Rad ulazi u problematiku nešto dublje nego što je potrebno za samu implementaciju, pa se tako u njemu mogu naći i principi koji nisu iskorišćeni u programu. Ovo je ubačeno kako bi se čitaocu pružila potpunija slika problema i potencijalnih rešenja, kao i da bi se imalo u vidu u kom pravcu se mogu vršiti dalja istraživanja.

Ključne reči— Savijanje proteina, *ACO*, paralelizacija, *CUDA*

Sadržaj

Slike

Tabele

1 Uvod

Problem savijanja proteina jedan je od najznačajnijih problema u bioinformatici, čiji rezultati mogu imati primene u raznim oblastima, pogotovo u medicini i industriji. Iako mehanizmi savijanja proteina i dalje nisu u potpunosti poznati, postoje uprošćeni modeli koji nam omogućavaju da predvidimo konformaciju proteina nakon savijanja, uzimajući u obzir dosadašnja znanja o ovom procesu. Model koji je ovde korišćen zasniva se na osobini hidrofobnosti aminokiselina, što ćemo koristiti kako bismo našli konformaciju sa minimalnom slobodnom energijom.

Problem nalaženja konformacije sa minimalnom slobodnom energijom spada u klasu NP-kompletnih problema. Iz tog razloga ima smisla koristiti probabilistički pristup, kao što je metaheuristika zasnovana na koloniji mrava (ACO). Iako vreme konvergencije za dugačke proteine i dalje može biti neprihvatljivo dugo, ovaj algoritam se do sada pokazao bolje od mnogih drugih pristupa rešavanju problema savijanja proteina, kao što su evolucioni algoritmi, Monte Karlo algoritmi, *MSOE* algoritam (eng. *Multi-Self-Overlap Ensemble*) [5] i mnogi drugi.

U ovom radu idemo korak dalje i pokušavamo dodatno ubrzati proces, koristeći grafičku karticu. Hipoteza je da će istovremeno pokretanje većeg broja mrava ubrzati konvergenciju, ispitujući veći deo prostora pretrage istovremeno.

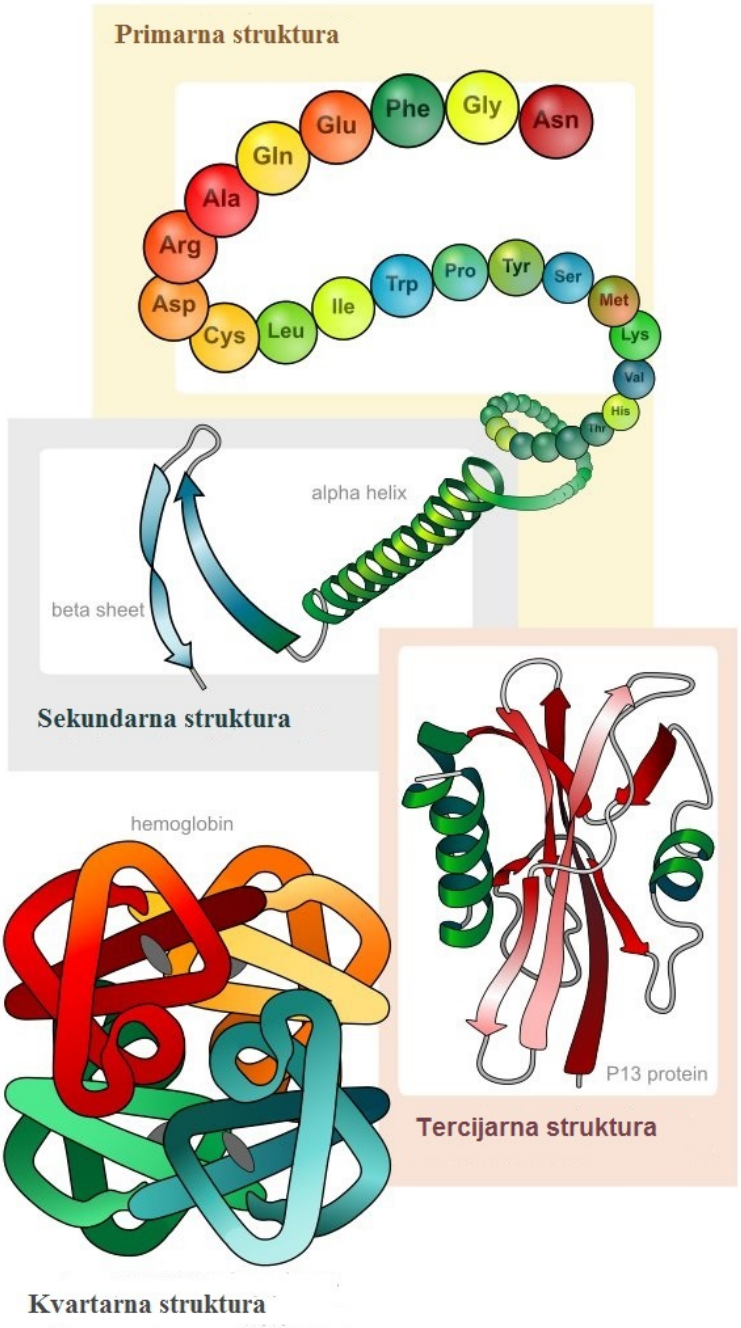
2 Savijanje proteina

2.1 Struktura proteina

Proteini su polimeri dvadeset različitih aminokiselina spojenih peptidnim vezama. Na fiziološkim temperaturama u vodenom rastvoru, polipeptidni lanac savija se u različite oblike, a u nekim slučajevima zauzima globularnu formu. Sekvenca različitih aminokiselina u proteinu, koja je neposredno određena sekvencom nukleotida u genu koji je enkodira, zove se primarna struktura. Prostornu struktura proteina često čine neki prepoznatljivi lokalni segmenti, kao što su α-heliksi, β-trake, obrti i neki drugi, koji se formiraju uspostavljanjem vodoničnih veza između $N - H$ i $C = O$ grupa u invarijantnim delovima aminokiselina glavnog lanca. Uprošćena prostorna strukrura proteina, tj. predstavljanje prostorne strukture pomoću prepoznatih tipova lokalnih segmenata, naziva se sekundarnom strukturom proteina. Tipovi lokalnih segmenata se nazivaju i tipovima sekundarne strukture. Tačan prostorni raspored proteina se naziva i tercijarna struktura. Mnogi proteini formiraju se udruživanjem lanaca više polipeptida, što čini kvartarnu strukturu proteina. Pomenuti nivoi strukture proteina mogu se videti na slici 2.1.

Da bi polipeptid funkcionisao kao protein, uglavnom mora biti u stanju da formira stabilnu tercijarnu strukturu (da se savije), pod određenim fiziološkim uslovima. Sa druge strane, potrebno je da savijeni protein ne bude previše nefleksibilan. Zbog ovih ograničenja, broj načina na koje se protein može saviti, iako je veliki, ipak je ograničen. Da li ograničen broj savijanja odražava fizička ograničenja, ili je posledica evolucije postojećih stabilnih savijanja, još uvek nije poznato. Ukoliko postoji mnogo mogućih stabilnih savijanja proteina koja nisu zastupljena u prirodi, moguće je proizvesti potpuno nove proteine za industrijsku i medicinsku upotrebu.

Slika 2.1: Struktura proteina



Funkcija proteina može se često, ali ne i u potpunosti, interpretirati iz njegove strukture. Nervni sistem je mreža ćelija i funkcionalne osobine ovih ćelija se mogu izvesti iz osobina i interakcija njihovih proteina. Proteini su uključeni u sve faze nervih aktivnosti. Oni koji se nalaze u membrani, regulišu transport jona i molekula, kao sredstvo razmene signala između ćelija. Neki od njih imaju enzimske funkcije da katalizuju hemijske procese. Raznovrsna i veoma specifična funkcija proteina je posledica njihove sofisticirane površine, u smislu oblika, napona i hidrofobnosti. Oblik i obrazac spoljašnje površine proteina su posledica jedinstvene trodimenzionalne strukture polipeptidnog lanca. Proteini su linearni polimeri sa (obično) neponavljajućom, kovalentnom strukturom. Kovalentna struktura je određena redosledom aminokiselina. Generalno je prihvaćeno da su savijanje i rezultujuća struktura proteina određeni sekvencom aminokiselina i njihovom sredinom (rastvorom, temperaturom, drugim prisutnim molekulima, itd.).

Jedna klasa proteina nema stalnu prostornu strukturu, već se njihova struktura prilagođava uslovima i okruženju. Takvi proteini nazivaju se neuređeni proteini (engl. disordered). Po savremenim saznanjima, većina proteina ima bar neke neuređene delove, koji često imaju veoma značajnu ulogu u ostvarivanju funkcije proteina.

2.2 Aminokiseline

Aminokiseline se mogu podeliti na više načina. Jedan od načina prikazan je u tabeli 2.1, a zasniva se na polarnosti aminokiselina. Bočni lanci aminokiselina imaju različite tendencije da učestvuju u interakciji jedni sa drugim ili sa vodom. Ove razlike dosta utiču na njihov doprinos stabilnosti i funkciji proteina.

Hidrofobne aminokiseline najčešće učestvuju samo u van der Valsovim vezama. Njihova tendencija da izbegnu kontakt sa vodom i da se slažu jedni do drugih je osnova za hidrofobni efekat. Alanin i leucin su aminokiseline koje favorizuju helikse, dok se prolin retko može pronaći u heliksima, zato što njegov kičmeni azot nije dostupan za vodoničnu vezu, koja je potrebna za nastanak heliksa.

Hidrofilne aminokiseline mogu praviti vodonične veze jedne sa drugim, sa vodom, sa polarnim organskim molekulima ili sa kičmom peptida. Neki od njih mogu promeniti svoje naelektrisanje, u zavisnosti od njihove pH vred-

Tabela 2.1: Klasifikacija aminokiselina

Hidrofobne aminokiseline			Polarne aminokiseline			Naelektrisane aminokiseline		
Alanin	Ala	A	Glutamin	Gln	Q	Arginin	Arg	R
Izoleucin	Ile	I	Asparagin	Asn	N	Lizin	Lys	K
Leucin	Leu	L	Histidin	His	H	Asparaginska kiselina	Asp	D
Fenilalanin	Phe	F	Serin	Ser	S	Glutaminska kiselina	Glu	E
Valin	Val	V	Treonin	Thr	T			
Prolin	Pro	P	Tirozin	Tyr	Y			
Glicin	Gly	G	Cistein	Cys	C			
			Metionin	Met	M			
			Triptofan	Trp	W			

nosti ili mikrookruzenja.

Amfipatske aminokiseline imaju polarne i nepolarne karakteristike, što ih čini idealnim za stupanje u interakcije sa drugim molekulima.

2.3 Važnost sekundarne strukture

Iako su proteini linearni polimeri, strukture mnogih proteina nisu nasumične kao kod sintetičkih polimera. Većina rastvorljivih proteina je globularna i ima jezgro koje se sastoji primarno od hidrofobnih aminokiselina. Ovo se objašnjava tendencijom hidrofobnih grupa da izbegnu kontakt sa vodom i intereaguju jedna sa drugom. Još jedna karakteristika savijenih polipeptidnih lanaca je da segmenti lanca u skoro svim proteinima zauzimaju konformaciju u kojoj se Φ i Ψ uglovi u kičmi ponavljaju po regularnom šablonu. Ovi regularni segmenti formiraju elemente sekundarne strukture proteina. Definisana su tri osnovna tipa sekundarnih struktura: heliksi, među kojima je najčešći α-heliks, β-ravni, koje mogu biti paralelne ili antiparalelne, i β-zaokret (eng. β-*turn*)u kojima je lanac primoran da promeni smer i koji omogućavaju kompaktno savijanje proteina.

Sekundarna struktura dosta utiče na stabilizaciju savijenog proteina. Heliksi i β-ravni se sastoje od mreže vodoničnih veza u kojima učestvuju mnoge uzastopne aminokiseline. Vodonične veze u ovim elementima obezbeđuju dosta stabilizacione entalpije koja omogućava polarnim grupama kičme da budu u hidrofobnom jezgru savijenog proteina.

Predikcija lokacije elemenata sekundarne strukture samo iz sekvence aminokiselina je precizna samo oko 70%. Ovakva predikcija je nekada korisna jer šablon elemenata sekundarne strukture u lancu može biti karakteristika određenog savijenog proteina. Na primer, β-ravan praćena α-heliksom, ponovljeno osam puta, uglavnom ukazuje na tip savijanja proteina koji se zove TIM-burence (eng. *TIM-barrel*). Sva TIM-burenca pronađeni do sada su enzimi, pa prepoznavanje TIM-burence savijanja u sekvenci sugeriše da protein ima katalističku funkciju. Ipak, opšte pravilo je da iako klasifikacija proteina može sugerisati funkciju, ona je ne može definisati; TIM-burenca mogu biti katalizatori u mnogim reakcijama, tako da se predikcija određenije funkcije ne može dobiti prepoznavanjem savijanja. Šta više, relativno malo savijanja mogu se prepoznati na ovaj način. Individualne sekundarne strukture se retko povezuju sa specifičnim funkcijama, iako za ovo postoje neki izuzeci.

2.4 Predikcija sekundarne strukture

Analiza frekvencija različitih aminokiselina u različitim tipovima struktura pokazuje neke opšte preferencije. Na primer, dugi bočni lanci se često mogu naći u heliksima, verovatno zbog toga što se ovi bočni lanci prostiru u suprotnom pravcu od gustih centralnih regija heliksnog cilindra. Nasuprot tome, aminokiseline čiji se bočni lanac račva u $\beta-$ugljeniku često se nalaze u $\beta-$ravnima, zbog toga što se svaki drugi bočni lanac u ravni prostire u suprotnom pravcu, što ostavlja mesta za $\beta-$razgranate bočne lance. Ovakve tendencije su u osnovi empirijskih pravila za predikciju sekundarne strukture iz sekvence.

U Chou-Fasman i drugim statističkim metodama za predikciju sekundarne strukture, pretpostavka je da lokalni efekti preovlađuju u odlučivanju da li će deo sekvence biti heliks, $\beta-$ravan, formirati zaokret ili poprimiti iregularnu konformaciju. Ova pretpostavka je verovatno samo delimično ispravna, što može delom objasniti nemogućnost ovakvih metoda da postignu stoprocentni uspeh u predviđanju sekundarne strukture proteina. Jedan od osnovnih razloga za nemogućnost 100% predikcije strukrure je u to što je tačnost skeniranja prostornog rasporeda proteina tek delimična, po nekim (ne baš novim) procenama 80-90%. Ove metode uzimaju proteine sa poznatim trodimenzionalnim strukturama i računaju preference pojedinacnih aminokiselina za različite strukturne elemente. Da bi se primenile ove preference na sekvencu nepoznate strukture, pokretni prozor od pet aminokiselina pomera se duž sekvence i prosečne preference se sabiraju. Zatim se primenjuju empirijska pravila za dodeljivanje sekundarnih struktura na osnovu

prosečnih preferenci.

U današnje vreme, sa razvojem oblasti istraživanja podataka (eng. *Data mining*), imamo mogućnost procesiranja velikog broja podataka. Proučavanjem $\alpha-$heliksa i $\beta-$ravni na ovaj način, možemo primetiti da se određene grupe aminokiselina često nalaze zajedno u ovim strukturama. Uočavanje ovih grupa može nagoveštavati da na tom mestu postoji određeni element sekundarne strukture. Na primer, može se uočiti da se u $\alpha-$heliksima Alanin, Leucin i Glutaminska kiselina često nalaze zajedno, u čak 34.1% slučajeva [30].

Ipak, ove tendencije nisu precizne i postoje mnogi izuzeci. Verovatno je čak korisnije posmatrati koji bočni lanci ne ulaze u određene tipove sekundarnih struktura. Iako prediktivne sheme bazirane na preferencijama aminokiselina imaju neku vrednost, ni jedna nije potpuno precizna.

2.5 Savijanje

Trodimenzionalna ili tercijarna struktura proteina određena je sekvencom aminokiselina. Translacija mRNA proizvodi linearni polimer aminokiselina koji se zatim spontano savija u kompaktniju, stabilniju strukturu. Nekada je savijanje potpomognuto drugim proteinima zvanim šaperoni (eng. *chaperon*), ali mnogi proteini mogu biti razvijeni i ponovo savijeni u rastvoru, što pokazuje da primarna struktura sadrži sve potrebne informacije da odredi način savijanja. Savijanje proteina odvija se prilično brzo, ali postoje dokazi da često postoji jedno ili više prelaznih stanja, pre nego što se dođe do finalne strukture, kao što se može videti na slici 2.2

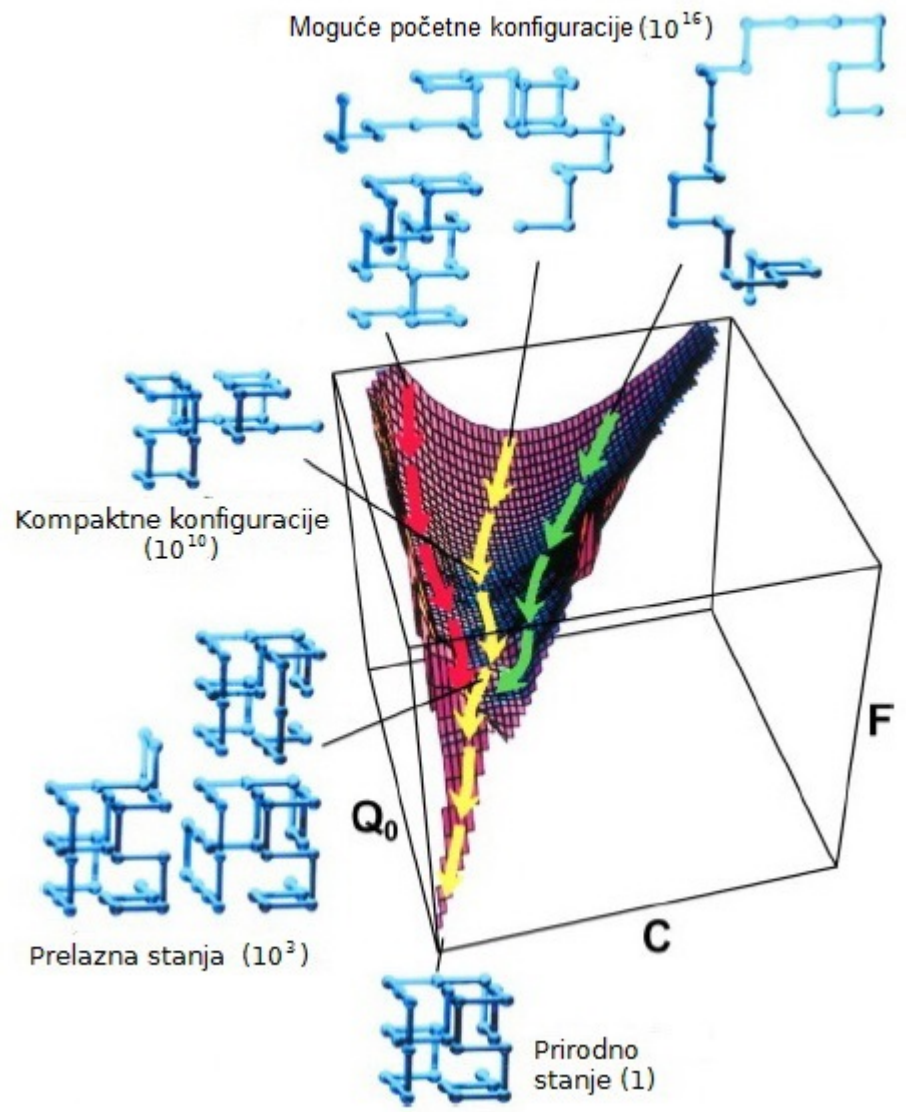
Posmatrajmo nasumičnu sekvencu koja se dobija iz ribozoma. Ako se lanac sastoji samo od polarnih i naelektrisanih aminokiselina, skoro svaka hemijska grupa u njemu može praviti vodoničnu vezu sa vodom, bilo da je lanac savijen ili ne, tako da ne bi bilo pokretačke snage da se formira kompaktna i regularna struktura. Mnoge ovakve sekvence postoje u prirodi i kao što se može pretpostaviti, nemaju stabilnu strukturu u rastvoru. Sekvence aminokiselina u rastvorivim proteinima su često mešavine polarnih i nepolarnih aminokiselina, često distribuiranim u lancu bez ikakvog primetnog šablona. Kada se ovakve sekvence sintetišu u vodi, one ne mogu ostati u tom obliku. Iako polarni bočni lanci, bočni lanci sa nabojem, kao i polarne peptidne grupe, mogu formirati vodonične veze sa vodom, nepolarni bočni lanci ne mogu. Njihovo prisustvo će poremetiti vodonočno vezanu strukturu

vode, bez da to nadomesti svojim vodoničnim vezama sa rastvorom. Da bi minimizovali efekat vode na strukturu, ovi bočni lanci imaju tendenciju da se grupišu, kao što to rade kapljice ulja u vodi. Hidrofobni efekat – grupisanje hidrofobnih bočnih lanaca iz različitih delova sekvence – prouzrokuje da polipeptid postane kompaktan. Sa energetske strane gledišta kompaktnost dovodi do dva povoljna rezultata: minimizuje ukupnu hidrofobnu površinu koja je u dodiru sa vodom i dovodi hidrofobne grupe blizu jednu drugoj, što omogućava stvaranje van der Valsovih veza između njih. Polarni delovi lanca ne moraju biti zaštićeni od rastvora pošto mogu praviti vodonične veze sa njim, tako da su oni uglavnom distribuirani po spoljnim delovima proteina.

Ipak, hidrofobni kolaps ima i negativne posledice. Kada se nepolarni bočni lanci grupišu da bi formirali hidrofobno jezgro, oni istovremeno vuku njihove polarne amidne grupe ka unutrašnjosti proteina. Ove polarne grupe pravile su vodonične veze sa vodom dok je lanac bio u razvijenom stanju, ali sada one to ne mogu da urade. Manjak vodoničnih veza kod ovih grupa dovodi do znatnog energetskog penala. Takođe, one ne mogu da prave vodonične veze sa polarnim bočnim lancima zbog toga što je većina takvih lanaca na površini gde intereaguje sa vodom. Rezultat je da se većina amino grupa i karbonilhih grupa vezuju između sebe. Zbog ove tendencije amino grupa da zadovolje svoj potencijal vodoničnih veza kroz samointerakciju, nastaje sekundarna struktura.

U novije vreme bilo je dosta pokušaja da se sekvence aminokiselina saviju u ispravnu trodimenzionalnu strukturu striktno računski. Postoji više ovakvih metoda, ali se svi zasnivaju na nekoliko pretpostavki. Prvo, pretpostavlja se da je ekvilibrijum konformacije globalni minimum slobodne energije. Ova pretpostavka je verovatno tačna, ali još uvek nije dokazana. Drugo, pretpostavlja se da su trenutni empirijski parametri potencijalne energije, koji se koriste da određivanje doprinosa vodoničnih veza i van der Valsovih interakcija ukupnoj stabilizacionoj energiji dovoljno precizni. Ovo nije sigurno. Treće, mnogi globularni proteini su oligomeri, a ovi metodi se koriste samo za monomere. Problem tretiranja oligomera u računskim pristupima savijanja još uvek nije adresiran.

Slika 2.2: Shematski prikaz savijanja proteina. Slobodna energija sistema (F) prikazana je kao funkcija koja zavisi od ukupnog broja kontakata između aminokiselina (C) i broja kontakata koji odgovaraju najstabilnijoj strukturi (Q_0)



2.6 Fizičke sile i principi u osnovi savijanja proteina i njegove strukture

Linearan polipeptidni lanac je autonomno organizovan u kompaktnu i dobro definisanu trodimenzionalnu strukturu. U globularnim proteinima, unutrašnje jezgro je najčešće sastavljeno od hidrofobnih aminokiselina, koje se drže zajedno van der Valsovim silama, a površina proteina sastavljena je uglavnom od polarnih bočnih lanaca i bočnih lanaca sa nabojem.

Sekvenca aminokiselina je primarna struktura proteina. C, O, N i H atomi leže na istoj ravni, a uzastopne ravni određuju prostorne uglove Φ i Ψ. Konformacija lanca od N aminokiselina se može na taj način, pomoću ovih uglova, definisati preko $2N$ parametara. Ograničena fleksibilnost polipeptidnog lanca je glavni faktor među onima koji određuju strukturu proteina i njegovo savijanje.

Prirodna (eng. *native*) konformacija mora biti energetski stabilna. Sa termodinamičke tačke gledišta, na slobodnu energiju proteina utiču sledeći faktori:

1. Hidrofobni efekat
2. Energija vodoničnih veza
3. Energija elektrostatičkih interakcija
4. Konformaciona entropija, nastala kao posledica ograničenog kretanja glavnog lanca i bočnih lanaca

Hidrofobni efekat bio je objašnjavan prvenstveno kao entropijski efekat koji nastaje kao posledica preuređivanja vodoničnih veza između molekula rastvora oko nepolarnih molekula. Ovaj proces je energetski nepovoljan i zbog toga tera nepolarne elemente da se grupišu, time smanjujući površinu koja je u dodiru sa rastvorom. Danas se hidrofobni efekat uglavnom posmatra kao kombinacija entropijskog efekta i van der Valsovih interakcija među molekulima (entalpijski efekat). On je entropijski na niskim temperaturama i entalpijski na visokim. Ipak, hidrofobni efekat je dugo smatran za glavnu pokretačku silu savijanja proteina, pošto dovodi do brzog kolapsa polipeptidnog lanca. Bez sumnje, hidrofobna interakcija je takođe jedna od glavnih stabilizacionih sila, doprinoseći termodinamičkoj stabilnosti savijenog proteina.

Uloga vodoničnih veza u savijanju i stabilnosti bila je dugo potcenjena, zbog mišljenja da se intermolekularne vodonične veze mogu zameniti vodoničnim vezama između molekula i rastvora. Sada, nakon mnogih istraživanja, vodonične veze su prihvaćene kao važan doprinos stabilnosti proteina, vazne koliko i hidrofobni efekat. Ovaj doprinos je procenjen na 1.5 ± 1.0 kcal/mol po intermolekularnoj vodoničnoj vezi.

Elektrostatičke interakcije, kao što su parovi jona i soni mostovi (eng. *salt bridges*) u proteinima su i dalje predmet izučavanja. Dok su vodonične veze i hidrofobna sila u osnovi neusmerene, elektrostatičke interakcije su uglavnom usmerene i zbog toga igraju važnu ulogu u određivanju načina savijanja proteina, kao i u njegovoj stabilnosti i funkciji. Računski i eksperimentalni dokazi pokazuju da soni mostovi mogu biti stabilišući i destabilišući. Sa druge strane, strukturno upoređivanje između termofilnih i mezofilnih proteina pokazuje da soni mostovi mogu dosta doprineti termalnoj stabilnosti proteina.

Glavni uticaj destabilizacije je konformaciona entropija polipeptidnog lanca. Savijanje dugog lanca u određenu, kompaktnu strukturu rezultuje značajnim smanjenjem entropije. Ovo je nadomešteno različitim interakcijama koje su već opisane. Rezultujuća ukupna stabilnost proteina (razlika slobodne energije između savijenog i nesavijenog stanja) je marginalna, $5 - 10 kcal/mol$. Mi znamo da hidrofobni efekat i vodonične veze imaju najveći uticaj na stabilizaciju, a da je konformaciona entropija najveći destabilišući faktor. Ipak, zbog kompenzatornog efekta u ukupnom balansu energije, kvantitativna predikcija koja uzima u obzir svaki tip interakcije ne može se pouzdano napraviti.

Unifikovani pogled na savijanje proteina bi trebao da bude dovoljno sveobuhvatan da interpretira različita eksperimentalna otkrića na ovom polju. Termodinamika pruža ovakav univerzalan pristup. Termodinamički sistem u ravnoteži zauzima stanje sa najmanjom Gibsovom (Gibbs) slobodnom energijom, pod konstantnim pritiskom i temperaturom.
Gibsova slobodna energija (G):

$$G(q) = H(q) - T(q)S(q),$$

gde je H entalpija, T apsolutna temperatura, S entropija proteina, a q predstavlja koordinate reakcije koje se koriste da opišu progres proteina na putu od početnog do prirodnog stanja. Pod fizioloskim uslovima, proteini zadržavaju

njihovu prirodnu strukturu zbog toga što povoljni termin entalpije, koji proizilazi iz interakcije proteina i rastvora, prevazilazi nepovoljni termin entropije, pa zato prirodno stanje ima manju Gibsovu slobodnu energiju od početnog stanja. Stabilnost proteina zavisi od rastvor-rastvor, rastvor-protein i protein-protein interakcija. Ove interakcije zavise od parametara koji opisuju termodinamičko stanje sistema.

Entalpijski i entropijski efekat je veliki, ali različitog znaka i skoro da se poništavaju. Razlika Gibsove slobodne energije između biološki aktivnih i razvijenih stanja proteina je prilično mala. Proteini se mogu razviti zagrevanjem, pod velikim pritiskom, ekstremnom pH vrednošću, dodavanjem soli i na druge načine.

2.7 Problem savijanja proteina

"Centralna dogma" molekularne biologije kaže da je tok sekvencijalnih informacija od nukleinske kiseline do proteina jednosmeran: sekvence nukleinskih kiselina kodiraju sekvencu proteina, ali jednom kad dođe do translacije, ne možemo više dobiti informacije o nukleinskim kiselinama na osnovu proteina. Moguća nadogradnja centralne dogme bila bi da protein enkodira svoju trodimenzionalnu strukturu. Ovo kodiranje se nekada zove i "drugi deo genetskog koda". Rešavanje ovog koda bilo bi ekvivalentno rešavanju problema savijanja proteina.

Pod pretpostavkom da polipeptid nasumično proverava sve konfiguracije, možemo proceniti vreme potrebno da se protein savije. Ako svaka veza dve uzastopne aminokiseline može imati, na primer, tri moguća stanja, onda protein sa 101 aminokiselinom može postojati u $3^{100} = 5 \times 10^{47}$ konfiguracija. Samo jedna od ovih konfiguracija odgovara prirodnom stanju. Čak i ako protein može da proverava 10^{13} konfiguracija u sekundi, trebaće mu 10^{27} godina da ih sve isproba. Ipak, proteini se savijaju za vrlo kratko vreme. Ovu kontradikciju prvi je primetio Cyrus Levinthal 1969. godine, pa je ona postala poznata kao Levintalov paradoks. Da bi prevazišao ovaj paradoks, Levintal je tvrdio da se protein ne može savijati koristeći nasumičnu pretragu, već da mora postojati određeni put savijanja.

Rešavanje problema savijanja proteina bilo bi od izuzetnog značaja za mnoge naučne oblasti, pre svega za industriju i medicinu. U industriji bi to dovelo do stvaranja novih materijala, čije bismo osobine mogli unapred da odredimo. Kada je reč o medicini, poznato je da su mnoge bolesti posledica

Tabela 2.2: Bolesti vezane za savijanje proteina [27]

Bolest	**Protein**	**Mesto savijanja**
Hiperholesterolemija	LDL-R	Endoplazmatični retikulum
Cistična fibroza	CFTR	Endoplazmatični retikulum
Fenilketonurija	PAH	Citosol
Hantingtonova bolest	HTT	Citosol
Marfanov sindrom	FBN1	Endoplazmatični retikulum
Osteogenesis imperfecta	Prokolagen	Endoplazmatični retikulum
Srpasta anemija	Hemoglobin	Citosol
Nedostatak $\alpha 1$ antitripsina	$\alpha 1$ antitripsin	Endoplazmatični retikulum
Skorbut	Kolagen	Endoplazmatični retikulum
Alchajmerova bolest	beta-APP	Endoplazmatični retikulum
Parkinsonova bolest	SNCA	Citosol
Krojcfeld—Jakobova bolest	Prioni	Endoplazmatični retikulum
Retinitis pigmentosa	RHO	Endoplazmatični retikulum
Katarakta	Kristalin	Citosil
Rak	p23	Citosil

pogrešnog savijanja proteina. Neke od ovih bolesti prikazane su u tabeli 2.2. Rešavanje problema savijanja proteina omogućilo bi nam da bolje razumemo prirodu ovih bolesti i da eventualno nađemo odgovarajuće tretmane za njih.

2.8 Kompleksnost problema savijanja proteina

Posmatraćemo dvodimenzionalni model savijanja proteina - konformaciju sa minimalnom slobodnom energijom proteina (MEP). Neka je dat graf $G = (V, A)$, gde je $V \subset \mathbb{Z} \times \mathbb{Z}$ (drugim rečima, V se sastoji od tačaka na rešetki, dok A predstavlja skup grana), $V = V_1 \cup V_2, \quad V_1 \cap V_2 = \varnothing, \quad A = A_1 \cup A_2, \quad A_1 \cap A_2 = \varnothing$, funkcija naelektrisanja $C : V \to \{-1, 0, 1\}$, energetska granica $K \in \mathbb{Z}^-$ i maksimalna udaljenost $L \in \mathbb{Z}^+$. Podgrafove možemo da rotiramo oko čvorova iz V_2, koji predstavljaju bočne lance koji se mogu rotirati za 180° tako da zauzimaju jedno od dva stabilna stanja, ali ne i oko čvorova iz V_1, čiji indukovani podgraf predstavlja kičmu proteina. Podskup grana A_1 predstavlja peptidne veze, dok A_2 predstavlja sve druge veze. Problem je pronaći uređenje V' čvorova iz V, gde je $V' \subset \mathbb{Z} \times \mathbb{Z}$ koji čuva $d(\overline{u}, \overline{v}), \quad \forall(\overline{u}, \overline{v}) \in A_1$, tako da je $E \leq K$, gde je $E = \sum \frac{C(\overline{u})C(\overline{v})}{d(\overline{u},\overline{v})}$, sumirano po svim čvorovima $\overline{u} = (x_1, y_1), \overline{v} = (x_2, y_2)$, gde $(\overline{u}, \overline{v}) \notin A_1$, i $d \leq L$, gde je d diskretizirana Euklidska razdaljina $\left\lceil ((x_2 - x_1)^2 + (y_2 - y_1)^2)^{\frac{1}{2}} \right\rceil$.

Slika 2.3: Graf G_1

Cilj nam je da pokažemo da je problem *MEP* NP-kompletan. Odgovarajući optimizacioni problem - nalaženje $min(E)$ umesto $E \leq K$ nije ništa jednostavniji od *MEP*. Generalno, ako je problem odlučivanja π_1 NP-kompletan, onda je njegov odgovarajući optimizacioni problem π_2 NP-težak. NP-teški problemi nisu lakši za rešavanje od NP-kompletnih problema.

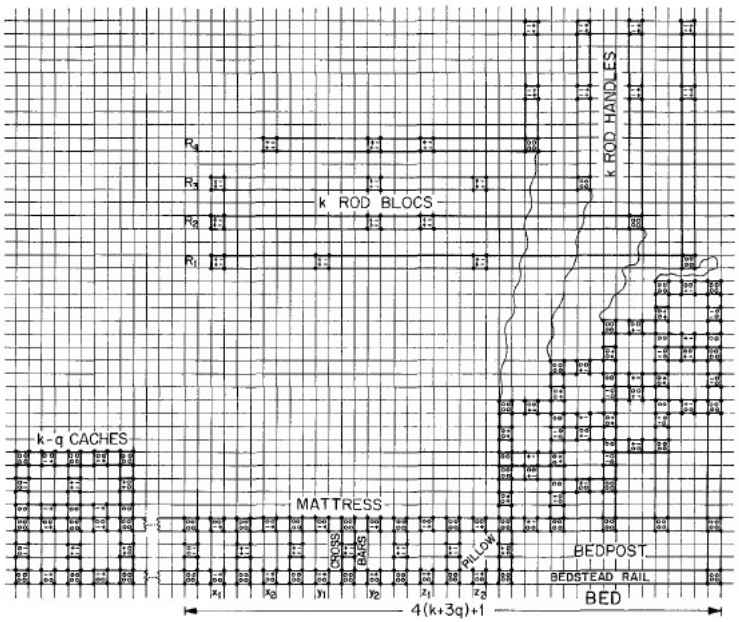
Da bismo uvideli da *MEP* pripada klasi NP problema, dovoljno je da primetimo da ukoliko nam je dato rešenje za *MEP*, potrebno je samo da proverimo da je $d(\overline{u}, \overline{v})$ očuvano za sve $(\bar{x}, \bar{y}) \in A_1$ i da je $E \leq K$, što se može izvršiti u polinomijalnom vremenu u odnosu na veličinu grafa G.

Ovde ćemo pokazati da je *3DM* $\propto$ *MEP* (*3DM* - trodimenzionalno uparivnje, eng. *Three-dimensional matching*). Neka su $X = \{x_1, x_2, ..., x_q\}, Y = \{y_1, y_2, ..., y_q\}, Z = \{z_1, z_2, ..., z_q\}$ konačni skupovi bez zajedničkih elemenata, a $R = \{r_1, r_2, ..., r_k\} \subseteq X \times Y \times Z$ instanca problema 3DM. Trebamo u polinomijalnom vremenu konstruisati graf $G = (V, E), \quad V \subset \mathbb{Z}^2, \quad K, L \in \mathbb{Z}^+$ i funkciju $C : V \to \{-1, 0, 1\}$ tako da R sadrži uparivanje $M \subseteq$ ako i samo ako preuređenje V' od V, gde $V' \subset \mathbb{Z}^2$ čuva vrednosti $d(\bar{x}, \bar{y})$ za sve $(\bar{x}, \bar{y}) \in A_1$, tako da je $E \leq K$.

Osnovni gradivni element za konstruisanje grafa G je kvadratni podgraf G_1 (Slika 2.3), koji se sastoji od četiri čvora na uglovima jediničnog kvadrata rešetke, povezanih sa četiri ivice iz A_1. Naelektrisanje u čvorovima je prikazano u uglovima kvadrata, gde $-$ označava -1, a $+$ označava 1.

Koristeći G_1 konstruišemo veće podgrafove (Slika 2.4): krevet (eng. *bed*), k šipki (eng. *rods*) i $k-q$ skladišta (eng. *caches*). Krevet se sastoji od okvira (eng. *bedstead*) koje sačinjavaju dve paralelne šine (eng. *bedstead rails*), svaka dužine $4(k+3q)+1$, levog dela koji obavija dušek (eng. *mattress*) dužine $12q$, koji se sastoji od $3q+1$ vertikalne prečke, deleći dušek na $3q$ jastuka (eng. *pillows*). Desni deo kreveta sastoji se od nogara (eng. *bedpost*) dužine $4k+1$, koji sadrži k vertikalnih kovčega (eng. *chests*) visine $10+3i, \quad 1 \leq i \leq k$, mereno od gornje šine. k vertikalnih ručki (eng. *rod handles*) imaju jednake dužine. Svaki od k horizontalnih blokova (eng. *rod blocs*) sadrži tri kopije

Slika 2.4: Primeri pomenutih podgrafova



G_1, čije lokacije odražavaju $r_1, r_2, ..., r_q$.

Ako je $r_i = (x_h, y_j, z_l)$, $(1 \leq h, j, l \leq q)$, onda su kopije G_1 na bloku šipke R_i na razdaljinama
$(4(k + 3q - i + 1 - h) + 1, 4(k + 2q - i + 1 - j) + 1, 4(k + q - i + 1 - l) + 1)$
od ručki $R_i, (1 \leq i \leq k)$. Svaka ručka sadrži dve kopije G_1, sa dodatkom jedne kopije G_1 (sa četiri 0 naelektrisanja) na preseku svake ručke sa svojim blokom.

Postavlja se $k - q$ kovčega sa leve strane levog kraja šina, na udaljenosti koja je za 1 veća od sume dužina svih k blokova i ručki od levog kraja. Naelektrisanja se postavljaju kao što je prikazano na slici 2.4.

Primećujemo da svaka od tri G_1 kopije u bloku ima dva $+1$ i dva -1 naelektrisanja. Jastuci imaju komplementarnu raspodelu naelektrisanja, tako da svaka kopija G_1 u bloku, kada se ubaci u centar jastuka, doprinosi minimalno -8 energiji. Analogno, distribucija naelektrisanja na svakoj ručki je takva da ukoliko je ubacimo u kovčeg ili u skladište, doprineće -8 energiji.

Na slici 2.4, horizontalne i vertikalne ivice pripadaju A_1, tako da one uka-

zuju na to da su razdaljine između krajnjih tačaka očuvane, dok sve druge ivice pripadaju A_2. Čvorovi povezani ivicama iz A_2 i bilo koji par nesusednih čvorova (tj., ne postoji ivica između njih), mogu se pomerati da bi se formirao V'. Iako se V i V' nalaze na planarnoj rešetki, njima odgovarajući grafovi ne moraju biti planarni.

Pretpostavimo da data instanca 3DM problema ima uparivanje $M = \{r_{i_1}, ..., r_{i_q}\} \subseteq R$. Lokacije kopija podgrafa G_1 na bloku šipke R_i ispunjavaju tri koordinate iz r_i. Zbog ovoga, šipke $R_{i_1}, ..., R_{i_q}$ se mogu ubaciti u krevet na taj način da svaki jastuk sadrži tačno jednu kopiju od G_1, a njihove ručke se ubacuju u unutrasnjošt odgovarajućih kovčega (na slici 2.4, R_1 i R_4 se mogu spustiti u jastuke na ovaj način).

Svaka kopija G_1 u jastuku ili u unutrašnjosti kovčega doprinosi -8 energiji, pa je doprinos ubacivanja $R_{i_1}, ..., R_{i_q}$ jednak $-40q$. Ručke preostalik $k - q$ šipki mogu se ubaciti u $k - q$ skladišta, tako da svaka doprinosi -16, to jest, sve zajedno doprinose $-16(k - q)$ ukupnoj energiji E. Dakle, ukupni doprinos energiji je:
$-40q - 16(k - q) = -8(2k + 3q) = K$

Sada pretpostavimo da imamo preuređenje V' od V, $V' \subseteq \mathbb{Z}^2$, tako da je $E \leq K$. One šipke čiji su blokovi ubačeni u jastuke i čija je ručka ubačena u odgovarajuće kovčege doprinose -40 energiji i za njih se kaže da su ispravno ugrađene (eng. *properly embedded*) Svaka od $k - q$ ručki ubačenih u $k - q$ skladišta doprinosi -16 energiji. Neke (ili sve) od ovih $k - q$ ručki se mogu smestiti u neke od $k - q$ kovčega, bez da se njihovi blokovi nalaze u jastuku, što takođe doprinosi -16 energiji. Za šipku smeštenu u skladište ili u kovčeg, tako da doprinosi -16 energiji, kaže se da je ispravno smeštena (eng. *properly placed*).

Ako je q šipki ispravno ugrađeno i $k - q$ šipki ispravno smešteno, onda je ukupna slobodna energija tačno K. Šta više, ispravno ugrađivanje šipki $R_{i_1}, ..., R_{i_q}$ ukazuje na to da R ima uparivanje $M = \{r_{i_1}, ..., r_{i_q}\}$. To pokazuje da svako drugo preuređenje grafa od V ima veću energiju od K.

Dokaz se zasniva na zapažanju da svaki čvor u iz G_1, koji se nalazi u šipci, a sa naelektrisanjem različitim od nule, može doprineti energiji najmanje -2, i da se ova donja granica može postići ako i samo ako je distanca čvora u od dva čvora pod pravim uglom i sa suprotnim naelektrisanjem od u, jednaka 1. Takođe, ova dva čvora moraju biti susedni čvoru u preko grane iz A_2. Pošto šipka ima 20 naelektrisanja različitih od 0, minimalna slobodna energija koju

može doprineti je -40, što se dobija kada je šipka ispravno ugrađena. Pošto ručka ima osam naelektrisanja različitih od 0, minimalna slobodna energija koju ona može da doprinese je -16, u slučaju da je ispravno smeštena.

Jedini naelektrisani čvorovi pod pravim uglom koji mogu da intereaguju sa naelektrisanim čvorovima na razdaljini 1 od njih su u jastucima, kovčezima i skladištima. Ovo isključuje interakcije između bilo koje dve šipke (ili više njih) izvan ovih struktura.

Geometrija konstrukcije implicira da je jedini način na koji šipka može da doprinese -40 taj da bude ispravno ugrađena. Zbog ovoga, dovoljno je pokazati da ako je q šipki ispravno ugrađeno, onda se $k-q$ preostalih šipki, ukoliko su ispravno smeštene, ne mogu preurediti tako da doprinose manje od -16.

Možemo primetiti da blok šipke koji je ispravno smešten u kovčeg ne može da intereaguje sa ostatkom kovčega da bi dalje smanjio energiju. Ovo se zaključuje iz činjenice da je raspodela naelektrisanja na ručkama i na blokovima komplementarna. Zbog toga, samo ispravno ugrađivanje q šipki i ispravno smeštanje $k-q$ ručki može doprineti K slobodnoj energiji. Takođe, ispravno ugrađivanje q šipki u $3q$ jastuka implicira da R ima uparivanje M. Ovim je tvrdnja dokazana [1].

[1] Dokaz preuzet iz [16]

3 ACO metaheuristika

3.1 Inteligencija rojeva

Posmatranje prirode već duže vreme pruža inspiraciju mnogim naučnicima, pogotovo u oblasti računarskih nauka. Jedan od izvora ovakve inspiracije je način na koji se organizmi u prirodi ponašaju kada su u grupama, na primer, kolonija mrava, pčela, bakterija, jato ptica i drugi. U ovim i mnogim drugim slučajevima, grupa individua pokazuje različito ponašanje u odnosu na pojedinačne organizme. Drugim rečima, posmatramo grupu jedinki kao jednu jedinku, odnosno roj. Na neki način, roj se pokazuje inteligentnijim od svih pojedinačnih jedinki u njemu.

Ovo zapažanje je osnova za mnoge koncepte i algoritme. Možemo posmatrati mozak i nervni sistem kao jedan primer ovog koncepta, gde se pojedinačni neuroni posmatraju kao agenti. Ipak, pojam inteligencije rojeva se najuže odnosi na mali i specifičan skup zapažanja i algoritama, koji zadovoljavaju sledeće osobine:

1. Korisno ponašanje proizlazi iz pojedinačnog truda grupe agenata
2. Pojedinačni agenti su u velikoj meri homogeni
3. Pojedinačni agenti deluju paralelno i asinhrono
4. Ima vrlo malo (ili uopšte nema) centralne kontrole
5. Komunikacija među agentima vrši se u nekoj formi stigmergije (eng. *stigmergy*), što je mehanizam indirektne kontrole agenata
6. "Korisno ponašanje"je relativno jednostavno

Kolekcija ćelija u mozgu ne zadovoljava ove uslove (2, 5 i 6), pa se ne može smatrati inteligencijom rojeva. Inteligencija rojeva odnosi se na jedinke koje sarađuju (znajući to ili ne), da bi dostigle određen konačan cilj. Primer su mravi koji pronalaze najkraći put od mravinjaka do izvora hrane, ili pčele koje pronalaze najbolji izvor nektara. Ovi i slični procesi su neposredno doveli do razvoja familija algoritama koji su se pokazali vrlo dobro u oblasti optimizacije i mašinskog učenja.

Inteligencija rojeva odnosi se najviše na dve familije algoritama: optimizacija zasnovana na koloniji mrava (eng. *Ant Colony Optimization*) i optimizacija zasnovana na roju čestica (eng. *Particle Swarm Optimization*).

3.2 *ACO* metaheuristika

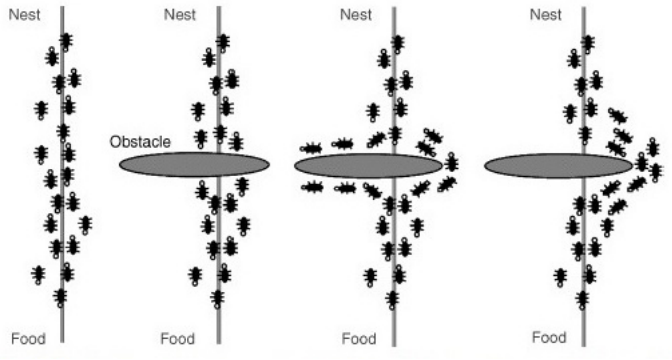
Osnovna ideja *ACO* algoritama je da imitiraju kooperativno ponašanje pravih mrava da bi rešili optimizacione probleme. *ACO* metaheuristiku je prvi predložio Marko Dorigo (*Marco Dorigo*) [36]. Ova metaheuristika može se posmatrati kao sistem sa više agenata, u kome je ponašanje svakog agenta inspirisano ponašanjem pravog mrava. Tradicionalno, *ACO* se koristi za kombinatorne optimizacione probleme i pokazali su se vrlo uspešni u resavanju mnogih problema (problem planiranja, problem rutiranja, dodele, itd.).

Glavna ideja preuzeta iz ponašanja pravih mrava je da mrav koristi kolektivno ponašanje da bi izvršio kompleksne zadatke, kao što su transport hrane i nalaženje najkraćeg puta do izvora hrane. *ACO* algoritmi oponašaju princip da korišćenjem vrlo jednostavnih mehanizama komunikacije, kolonija mrava može pronaći najkraći put između dve tačke.

Mravi ne mogu dobro da vide. Kolonija ima pristup izvoru hrane, sa kojim je povezana putem dve staze od izvora ka mravinjaku. U toku puta, mravi ostavljaju hemijski trag (feromone) na zemlji. Feromoni su isparljive supstance koje uticu na čulo mirisa. Uloga ovog traga je da navodi druge mrave ka cilju. Što je veća količina feromona na određenom putu, veća je verovatnoća da će mrav izabrati taj put. Na slici 3.1 može se videti kako grupa mrava pronalazi najkraći put od kolonije do izvora hrane.

Feromoni isparavaju u toku vremena, a količina koju mrav ostavlja za sobom zavisi od količine hrane (reinforcement process). Osnovni algoritam se sastoji iz dva iterativna koraka: konstrukcija rešenja i ažuriranje feromona. Osnovni algoritam opisan je pomoću pseudokoda 1.

Slika 3.1: Pronalaženje najkraćeg puta



Može se pokazati da algoritam ACO konvergira ka optimalnom rešenju. Detaljan dokaz može se pronaći u postojećem radu [22].

3.3 Faza konstrukcije

Konstrukcija rešenja vrši se prema probabilističkom pravilu prelaza stanja (eng. *state transition rule*). Mravi se mogu posmatrati kao stohastičke pohlepne procedure koje konstruišu rešenje na probabilistički način, dodavanjem kokomponenata rešenja parcijalnom rešenju, sve dok se ne dobije kompletno rešenje. Ciljni optimizacioni problem se može videti kao graf odlučivanja, gde će mrav konstruisati putanju. Ovaj iterativni proces uzima u obzir:

1. ***Trag feromona*** – trag feromona pamti karakteristike dobrog generisanog rešenja, što će voditi konstrukciju novih rešenja. Trag feromona se dinamički menja u toku pretrage da bi odrazio prikupljeno znanje. Predstavlja memoriju celog pretraživačkog procesa.

2. ***Heurističke informacije*** - informacije specifične za problem dodatno navode mrave pri konstrukciji rešenja.

Svaki mrav će konstruisati rešenje na stohastički[1] način. Ako je data nasumična početna tačka i, mrav će odabrati sledeću tačku j sa verovatnoćom:

$$p_{i,j} = \frac{\tau_{i,j}}{\sum_{k \in S} \tau_{i,k}}, \quad \forall j \in S$$

[1] zasnovano na slučajnosti

Algoritam 1: Pseudokod algoritma *ACO*

```
input  : Sekvenca S, brojMrava, α, β, ρ
output: Konformacija sa minimalnom pronađenom energijom
begin
    matricaFeromona ← inicijalizacijaMatriceFeromona();
    najboljeRešenje ← null;
    optimalnaEnergija ← aproksimacijaOptimalneEnergije(S);
    while not kriterijumZaustavljanjaIspunjen() do
        skupRešenja ← null;
        for i = 1 to brojMrava do
            rešenje ← konstruišiRešenje(S, matricaFeromona, α,
            β);
            dodajUSkupRešenja(skupRešenja, rešenje);
        lokalnoNajboljeRešenje ←
        nađiRešenjeSaMinEnergijom(skupRešenja);
        lokalnoNajboljeRešenje ←
        lokalnaPretraga(lokalnoNajboljeRešenje);
        if (energija(lokalnoNajboljeRešenje) <
        energija(najboljeRešenje)) then
            najboljeRešenje ← lokalnoNajboljeRešenje;
        ažurirajTragFeromona(lokalnoNajboljeRešenje,
        optimalnaEnergija, ρ);
    return najboljeRešenje;
```

gde S predstavlja još neposećene tačke u pretrazi, a $\tau_{i,j}$ vrednost feromona kada se ide od aminokiseline i ka aminokiselini j. Sve vrednosti feromona inicijalizovane su kao $\tau_{i,j} = 1$, kako bi svaka putanja imala određenu verovatnoću da bude izabrana. Ima smisla da se definiše minimalnu vrednost feromona, kako se zbog procesa isparavanja ne bi desilo da neke putanje postanu zauvek nedostupne, ukoliko im vrednost padne na 0. Mravi koriste nasumično uzetu početnu tačku u fazi konstrukcije.

Dodatna heuristika definiše se u zavisnosti od problema, posmatrajući vrednosti $\eta_{i,j}$, gde je $\eta_{i,j} = 1/d_{i,j}$, a $d_{i,j}$ predstavlja udaljenost između tačaka i i j. Što je veća vrednost heuristike $\eta_{i,j}$, manja je razdaljina između tačaka i i j. Računanje verovatnoća prelaska iz tačke i u tačku j se, dakle, računa kao:

$$p_{i,j} = \frac{\tau_{i,j}^{\alpha}\ \eta_{i,j}^{\beta}}{\sum_{k \in S} \tau_{i,k}^{\alpha}\ \eta_{i,k}^{\beta}}, \quad \forall j \in S$$

gde su α i β parametri koji određuju relativni doprinos vrednosti feromona i heurističkih vrednosti. Ako je $\alpha = 0$, algoritam ACO biće sličan stohastičkom pohlepnom algoritmu [2], u kome će bliže tačke imati veću verovatnoću da budu odabrane. Ako je $\beta = 0$, samo će vrednosti feromona usmeravati pretragu. U ovom slučaju, može doći do brže konvergencije ka lokalnom najboljem rešenju.

3.4 Ažuriranje feromona

Ažuriranje feromona se vrši na osnovu generisanih rešenja. Odvija se u dve faze:

1. ***Faza isparavanja*** - trag feromona automatski isparava. Svaka vrednost feromona se smanjuje po formuli:

   $$\tau_{i,j} = (1 - \rho)\tau_{i,j}, \quad \forall i, j \in [1, n]$$

   gde $\rho \in [0, 1]$ označava stopu isparavanja feromona. Cilj isparavanja je izbegavanje preuranjene konvergencije.

2. ***Faza pojačavanja*** - trag feromona se ažurira na osnovu generisanih rešenja. Mogu se primeniti različite strategije.

[2] *Stohastički pohlepni algoritam* je verzija pohlepnog algoritma. Dok pohlepni algoritam bira naredni korak deterministički, bazirano na lokalno najboljoj opciji, stohastički pohlepni algoritam bira naredni korak stohastički. Drugim rečima, verovatnoća izbora lokalno najbolje opcije nije 1, kao kod običnog pohlepnog algoritma, već bolje opcije imaju veću verovatnoću da budu izabrane [12].

Strategije ažuriranja feromona:

1. ***Onlajn korak po korak ažuriranje:*** Trag feromona $\tau_{i,j}$ ažurira mrav pri svakom koraku konstrukcije rešenja.

2. ***Onlajn odloženo ažuriranje:*** ažuriranje feromona vrši se onda kada mrav završi sa generisanjem kompletnog rešenja. Svaki mrav ažurira informacije o feromonima sa vrednošću koja je proporcionalna kvalitetu nađenog rešenja. Što je bolje rešenje, mrav će ostaviti više feromona.

3. ***Oflajn ažuriranje feromona:*** Trag feromona se ažurira u trenutku kada svi mravi završe sa konstrukcijom rešenja. Ovo je najpopularniji pristup kod koga se mogu koristiti različite strategije:

   (a) *Ažuriranje zasnovano na kvalitetu:* Ova strategija ažurira vrednosti feromona na osnovu najboljeg pronađenog rešenja među svim mravima (ili k najboljih rešenja). Vrednost koja se dodaje zavisi od kvaliteta izabranih rešenja. Za svaku komponentu koja pripada najboljem rešenju $\widehat{\pi}$, dodaje se pozitivna vrednost Δ:
   $\tau_{i,\widehat{\pi}(i)} = \tau_{i,\widehat{\pi}(i)} + \Delta, \quad \forall i \in [1, n]$

   (b) *Ažuriranje zasnovano na rangu:* Najboljih k rešenja mogu ažurirati feromone vrednošću koja zavisi od ranga rešenja.

   (c) *Ažuriranje zasnovano na najgorem rešenju:* Mrav koji proizvede najgore rešenje će smanjiti vrednost feromona koji odgovaraju komponentama rešenja.

   (d) *Elitističko ažuriranje:* Najbolje rešenje koje je do tada pronađeno će ažurirati vrednosti feromona. Minimalna vrednost feromona je definisana, tako da svaka opcija ima minimalnu verovatnoću da bude odabrana.

4 Savijanje proteina zasnovano na *ACO* metaheuristici

4.1 *2D HP* model

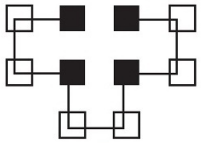
Jedan od najkorišćenijih modela proteina je hidrofobno-polarni model (eng. *HP – hidrophopic-polar*). *HP* model je apstrakcija hidrofobnih interakcija u procesu savijanja proteina, predstavljajući šablon hidrofobnosti u proteinu – nepolarne aminokiseline klasifikuju se kao hidrofobne, a polarne aminokiseline kao hidrofilne. Sekvenca se sastoji od aminokiselina $s \in H, P$, gde H predstavlja hidrofobnu, a P hidrofilnu aminokiselinu.

HP model ograničava moguće konformacije na samozaobilazeće putanje na rešetki, gde su temena označena aminokiselinama. Postoji veliki broj mogućih konformacija, na primer, sekvenca dužine 25 ima 5768299665 ovakvih samozaobilazećih putanja. Rešetka omogućava diskretizaciju prostora konformacija proteina. *HP* model je dosta proučavan za konformacije projektovane na *2D* i *3D* rešetku. Uprkos jednostavnosti ovog modela, on je dovoljno dobar da se na njemu zapaze mnoge osobine proteina.

Potencijalna energija *HP* modela reflektuje osobinu hidrofobnih aminokiselina da teže formiranju hidrofobnog jezgra. Da bi se modelirala ova osobina proteina, *HP* model dodaje vrednost ϵ za svaki par hidrofobnih aminokiselina koje formiraju topološki kontakt: topološki kontakt formira par hidrofobnih aminokiselina koje su susedne na rešetki, ali nisu uzastopne u sekvenci. Za vrednost konstante ϵ uglavnom se uzima -1.

Sekvenca S projektuje se na *2D* rešetku tako da svaka dva susedna ele-

Slika 4.1: a) Konfiguracija sekvence *HPPHPPHPPH* na kvadratnoj rešetki. b) Projekcija korišćenjem Dekartovog koordinatnog sistema. c) Projekcija korišćenjem internih koordinata. d) Projekcija korišćenjem matrice udaljenosti



a)

P	H	H	P
P	H	H	P
-	P	P	-

b)

RRLRRLRR

c)

	1	2	3	4	5	6	7	8	9	10
1	0	1	2	1	2	3	2	3	2	1
2	1	0	1	2	3	4	3	4	3	2
3	2	1	0	1	2	3	2	3	4	3
4	1	2	1	0	1	2	1	2	3	2
5	2	3	2	1	0	1	2	3	4	3
6	3	4	3	2	1	0	1	2	3	2
7	2	3	2	1	2	1	0	1	2	1
8	3	4	3	2	3	2	1	0	1	2
9	2	3	4	3	4	3	2	1	0	1
10	1	2	3	2	3	2	1	2	1	0

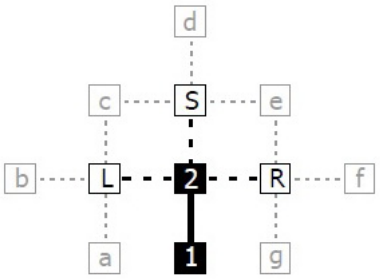
d)

menta iz S zauzimaju susedno mesto na rešetki i nijedno polje na rešetki ne sadrži više od jednog elementa. Sekvenca se na rešetku može projektovati na više načina. Tri česta metoda su: *Dekartov koodrinatni sistem* (lokacija svake aminokiseline na rešetki se predstavlja odvojeno), *interne koordinate* (protein se predstavlja kao niz koraka od jedne aminokiseline do druge), *matrica udaljenosti* (lokacije aminokiselina se dobijaju iz njihovoh uzajamnih distanci). Ovi metodi prikazani su na slici 4.1. U ovom radu, korišćena je kombinacija Dekartovog koordinatnog sistema i internih koordinata, radi što bržeg izvršavanja programa.

Savijanje *HP* sekvence enkodirano je *apsolutnim* (**E**(ast), **W**(est), **N**(orth), **S**(outh)) ili *relativnim* (**S**(traight), **L**(eft), **R**(ight)) koordinatama (Slika 4.2). Pokazalo se da postoji veća šansa pronalaska dobrog rešenja ako se koriste relativne koordinate.

Naravno, ovaj model je veoma jednostavan. Interakcije u pravim proteinima su dosta složenije i trodimenzionalne. Ipak, ovaj model omogućava nam da proučavamo tehnike za globalnu optimizaciju (u ovom slučaju *ACO*) za pojednostavljenu verziju problema savijanja proteina. Model ima svoje biomolekularno opravdanje.

Slika 4.2: Pozicije koje treba proveriti pri dodavanju novog elementa, korišćenjem relativnih koordinata



4.1.1 Poboljšanje *HP* modela

Posmatramo konformaciju na slici 4.3, koja prikazuje dve konformacije iste sekvence. Klasičan energetski potencijal *HP* modela uzima u obzir samo direktne H-H kontakte, pa bi potencijalna energija obe konformacije bila ista. Ipak, figura (a) je bliže formiranju optimalne konformacije od figure (b).

Ovo se može popraviti proširenjem funkcije energije, tako da omogući uticaj razdaljine H-H interakcija na energiju. Moguće je konstruisati takvu funkciju koja čuva rang konformacije u *HP* modelu, ujedno omogućavajući prepoznavanje boljih konformacija među konformacijama sa istim brojem H-H interakcija.

Na primer, ako je $d_{i,j}$ udaljenost između hidrofobnih aminokiselina H_i i H_j, možemo koristiti:

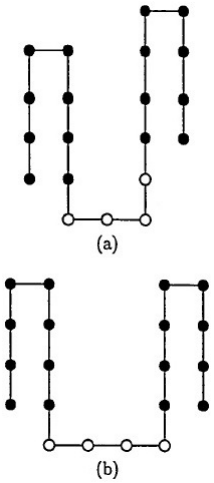
$$\widehat{E}_{H_i,H_j}(d_{i,j}) = \begin{cases} -1 & , d_{i,j} = 1 \\ \frac{-1}{d_{i,j}^k N_h} & , d_{i,j} > 1 \end{cases}$$

gde je N_H broj hidrofobnih aminokiselina u sekvenci, a parametar k se obično bira tako da bude $k = 4$ za 2D rešetke, a $k = 5$ za 3D rešetke.

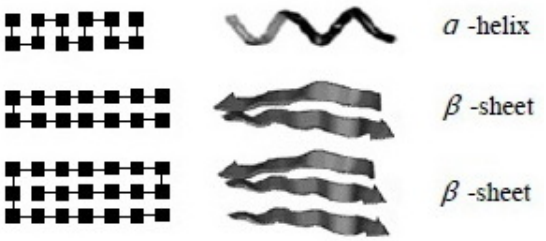
4.1.2 Prepoznavanje elemenata sekundarne strukture

Iako je model kvadratne rešetke samo apstrakcija modela savijanja proteina, neke konformacije mogu ukazivati na elemente sekundarne strukture proteina. Slika 4.4 pokazuje *2D-HP* konformacije i odgovarajuće sekundarne strukture, kao što su $\alpha - heliksi$ i $\beta - ravni$. Ipak, ovaj model nije zadovoljavajući za mnoge biologe. Trodimenzionalna struktura specifičnog proteina

Slika 4.3: Dve konformacije sa istom energijom



Slika 4.4: *HP* konformacije i odgovarajuće sekundarne strukture

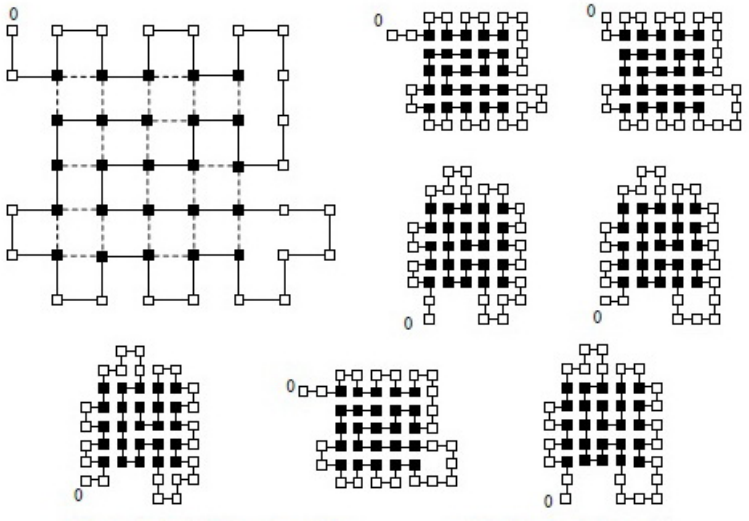


je jedinstvena, ali kao što se može videti sa slike 4.5, postoji više ekvivalentnih konformacija za istu sekvencu aminokiselina. Zbog ovoga je *HP*-model previše jednostavan da bi mogao u potpunosti da predstavi strukturu proteina.

4.1.3 Problem parnosti

Veliki propust kvadratne *HP* rešetke je da bilo koja dva elementa na parnoj udaljenosti u sekvenci ne mogu formirati topološki kontakt jedan sa drugim kada se protein projektuje na kvadratnu rešetku. Ovaj problem se naziva *problem parnosti* (eng. *Parity problem*).

Slika 4.5: Neke od konformacija za sekvencu 14. Crne tačke označavaju hidrofobne aminokiseline, dok bele tačke označavaju polarne aminokiseline

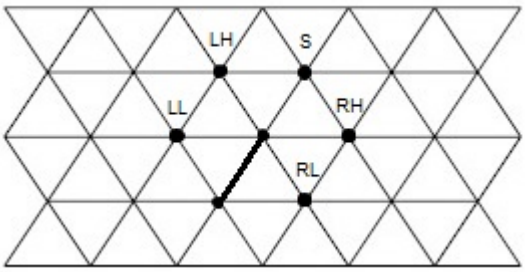


Problem parnosti ne postoji u prirodi. Iz ovog razloga, razmatramo mogućnost upotrebe trougaonih rešetki. Lako se uočava da ovakva rešetka nema problem parnosti. Drugim rečima, za svake dve aminokiseline na udaljenosti makar dva jedna od druge, može se konstruisati savijanje na rešetki u kome ove aminokiseline nekovalentno intereaguju jedna sa drugom.

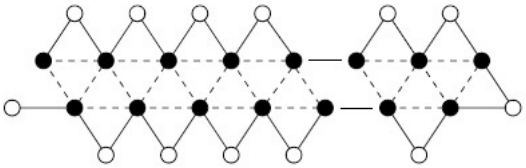
Korišćenje trougaone rešetke ne menja previše algoritam. Očigledno, postoji više smerova u koje se može saviti naredna aminokiselina: umesto tri moguća smera na kvadratnoj rešetki (**L**eft, **S**traight, **R**ight), na trougaonoj rešetki imamo pet mogućih smerova (**LL** – *Left Low*, **LH** – *Left High*, **S** – *Straight*, **RH** – *Right High*, **RL** – *Right Low*), kao što se može videti na slici 4.6. Takođe, prilikom izbora najboljeg smera na trougaonoj rešetki, potrebno je proveriti 17 tačaka, za razliku od 7 tačaka na kvadratnoj rešetki. Maksimalan broj novih H-H interakcija se takođe razlikuje na trougaonoj rešetki: možemo da imamo najviše 5 novih interakcija za krajnje tačke sekvence ili najviše 4 nove interakcije za unutrašnje tačke sekvence. Matrica feromona se mora izmeniti tako da čuva informacije za pet smerova, umesto za tri smera. Osim ovoga što je navedeno, algoritam sa trougaonom rešetkom se ne razlikuje u odnosu na algoritam sa kvadratnom rešetkom.

Treba primetiti da se minimalna pronađena energija na ovim modelima

Slika 4.6: Trougaona HP rešetka



Slika 4.7: Sekvenca $HPHPHP...HP$ savijena na trougaonoj rešetki



može drastično razlikovati. Na primer, sekvenca $HPHPHP...HP$ ne može formirati ni jednu interakciju između dva H elementa na kvadratnoj rešetki, pošto se svi nalaze na neparnim pozicijama. Sa druge strane, na trougaonoj rešetki ova sekvenca se može saviti na način koji podseća na β-ravan, sa značajnim brojem H-H interakcija, kao što se može videti na slici 4.7.

4.1.4 Uopštavanje hidrofobnosti

HP model može da se dodatno uopšti. Činjenica je da određene hidrofobne aminokiseline pokazuju veći stepen hidrofobnosti od drugih (Tabela 4.1). Dok u standardnom *HP* modelu sve hidrofobne aminokiseline imaju istu hidrofobnu vrednost, drugi modeli omogućavaju različitim hidrofobnim aminokiselinama da imaju različite hidrofobne vrednosti i kontakt između dve hidrofobne aminokiseline ima doprinos energiji koji je proporcionalan njihovoj ukupnoj vrednosti.

U novom modelu, dozvoljavamo svakoj od 20 aminokiselina da ima vrednost iz skupa realnih brojeva. Aminokiseline sa vrednošću većom od 0 pred-

Tabela 4.1: Različite skale hidrofobnosti

Aminokiselina	**kdHidrofobnost**[18]	**wwHidrofobnost**[19]	**hhHidrofobnost**[20]	**mfHidrofobnost**[21]
Ile	4.5	0.31	-0.60	-1.56
Val	4.2	-0.07	-0.31	-0.87
Leu	3.8	0.56	-0.55	-1.81
Phe	2.8	1.13	-032	-2.20
Cys	2.5	0.24	-0.13	0.49
Met	1.9	0.23	-0.10	-0.76
Ala	1.8	-0.17	0.11	0.0
Gly	-0.4	-0.01	0.74	1.72
Thr	-0.7	-0.14	0.52	1.78
Ser	-0.8	-0.13	0.84	1.83
Trp	-0.9	1.85	0.30	-0.38
Tyr	-1.3	0.94	0.68	-1.09
Pro	-1.6	-0.45	2.23	-1.52
His	-3.2	-0.96	2.06	4.74
Glu	-3.5	-2.02	2.68	1.64
Gln	-3.5	-058	2.36	3.01
Asp	-3.5	-1.23	3.49	2.95
Asn	-3.5	-0.42	2.05	3.74
Lys	-3.9	-0.99	2.71	5.39
Arg	-4.5	-0.81	2.58	3.71

stavljaju hidrofobne aminokiseline, dok sve ostale vrednosti predstavljaju polarne aminokiseline.

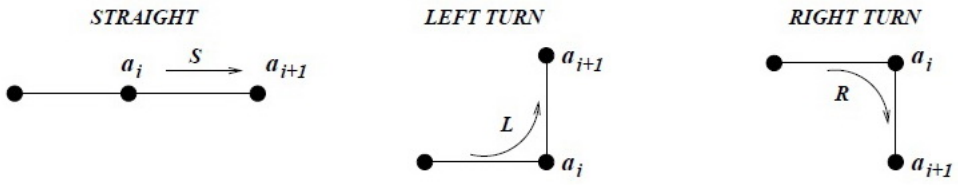
Ako je odnos između najmanje i najveće vrednosti hidrofobnosti ρ, onda korišćenje klasičnog algoritma za binarni *HP* model može u najgorem slučaju rezultirati aproksimacijom za faktor ρ manjom od optimalne. Ovakvi gubici se dešavaju pošto neke pozicije u savijanju prave manje interakcija od drugih, pa ako se na ovo mesto postavi aminokiselina sa visokom vrednošću hidrofobnosti, aproksimacija je na gubitku. Zbog toga treba proširiti algoritam tako da prilikom konstrukcije uzima u obzir i vrednosti hidrofobnosti. Jedan ovakav algoritam opisan je u [17].

4.2 Faza konstrukcije

Konformacije se generišu nasumično sa različitim početnim tačkama. Zbog ovoga, neki mravi će savijati protein sa jednog kraja na drugi, dok će drugi početi od neke tačke unutar proteina, a zatim savijati prvi deo, a onda drugi deo proteina. Ideja je da ova vrsta dodeljivanja različitih početnih pozicija mravima proširuje prostor pretrage.

Nemoguće konformacije generisane tokom faze konstrukcije popravljaju se korišćenjem pretrage unazad (eng. *Backtracking*). Svaki put kada algoritam doda novu aminokiselinu, on proverava da li je trenutna konformacija

Slika 4.8: Komponente rešenja



moguća. Ukoliko ne postoji pravac u kome bi se niz mogao nastaviti, brišemo polovinu već dodatih aminokiselina (ova strategija pretrage unazad pokazala se kao najbolja[4]).

Proces konstrukcije izvršava svaki mrav počevši od prazne konformacije i iterativno proširujući parcijalnu konformaciju novim aminokiselinama. Određivanje položaja naredne aminokiseline vrši se probabilistički, u skladu sa tragom feromona τ i funkcije prilagođenosti η.

Verovatnoća da se izabere određeni pravac d u koji će se saviti trenutna aminokiselina a_j kada idemo iz aminokiseline a_i računa se kao:

$$p_{i,j,d} = \frac{[\tau_{i,j,d}]^{\alpha}[\eta_{i,j,d}]^{\beta}}{\sum_{k\in\{L,R,S\}}[\tau_{i,j,d}]^{\alpha}[\eta_{i,j,d}]^{\beta}},$$

gde je $\eta_{i,j,d}$ funkcija prilagođenosti, $\tau_{i,j,d}$ intenzitet traga između i i j u pravcu d, α parametar koji reguliše uticaj $\tau_{i,j,d}$, a β parametar koji reguliše uticaj $\eta_{i,j,d}$. Pošto vrednosti $\tau_{i,j}$ imaju definisanu minimalnu vrednost, ne može se desiti da verovatnoća izbora nekog pravca bude 0.

4.3 Matrica feromona

Matrica feromona treba da sadrži informacije iz prethodnih iteracija algoritma koje će pomoći mravima da izgrade rešenje. Kao krajnji rezultat želimo da pojačamo najjednostavniju, ali i dalje korisnu komponentu rešenja. Komponente koje bi trebalo pojačati su konfiguracije prikazane na slici 4.8.

Dakle, pojačavamo izabrani relativni pravac trenutne aminokiseline. Zbog toga što ćemo savijati protein ne nužno od prve aminokiseline, moramo uzeti u razmatranje slučajeve kada idemo od aminokiseline 1 ka 2, ali i od 2 ka 1. Postoji određena simetrija koja se može iskoristiti, zato što ako idemo od 1

Tabela 4.2: Matrica feromona

$a_i \to a_j$	S	R	L	Napomena
$1 \to 2$	$\tau_{1,2,S}$	$\tau_{1,2,R}$	$\tau_{1,2,L}$	$\tau_{1,2,S} = \tau_{2,1,S}$; $\tau_{1,2,L} = \tau_{2,1,R}$; $\tau_{1,2,R} = \tau_{2,1,L}$
$2 \to 3$	$\tau_{2,3,S}$	$\tau_{2,3,R}$	$\tau_{2,3,L}$	$\tau_{2,3,S} = \tau_{3,2,S}$; $\tau_{2,3,L} = \tau_{3,2,R}$; $\tau_{2,3,R} = \tau_{3,2,L}$
...	...	...	...	...
$N \to N-1$	$\tau_{N-1,N,S}$	$\tau_{N-1,N,R}$	$\tau_{N-1,N,L}$	$\tau_{N-1,N,S} = \tau_{N,N-1,S}$; $\tau_{N-1,N,L} = \tau_{N,N-1,R}$; $\tau_{N-1,N,R} = \tau_{N,N-1,L}$

ka 2 desno, to je isto kao da idemo od 2 ka 1 levo, a ići od 1 ka 2 pravo je isto kao i ići od 2 ka 1 pravo. Zbog ovoga, matricu feromona možemo predstaviti kao tabelu 4.2, gde su a_i i a_j uzastopne aminokiseline, S,R,L pravci pravo, desno i levo, a $\tau_{i,j,d}$ vrednosti feromona kada idemo iz a_i ka a_j u pravcu d.

Feromoni se ažuriraju u skladu sa jednačinom:

$$\tau_{i,j,d} = (1 - \rho)\tau_{i,j,d} + \delta(i, j, d, S),$$

gde je $0 < \rho \leq 1$ brzina isparavanja (eng. *evaporation rate*) traga, tako da $(1 - \rho)$ modelira trajnost (eng. *persistence*) traga (što je ρ veće, brže će se zaboravljati informacije stečene u prethodnim iteracijama), a $\delta(i, j, d, S)$ je odnos totalne energije konfiguracije S (jedne ili više njih, u zavisnosti od tipa ažuriranja feromona) koja sadrži granu $i \to j$ sa pravcem d i optimalne energije (koja je ili poznata ili se procenjuje na osnovu broja H aminokiselina u sekvenci). Ažuriranje se vrši koristeći relativan kvalitet rešenja da bi se osigurali da sekvence sa velikim vrednostima energije ne dovedu do *ACO* stagnacije.

4.4 Heuristika

Funkcija prilagođenosti $\eta_{i,j,d}$ treba da navodi konstrukciju prepoznajući dobitke i gubitke u ciljnoj funkciji. U ovom slučaju, funkcija prilagođenosti treba da pomogne pri odlučivanju o tome koja heuristička vrednost treba biti dodeljena za svaki mogući pravac za trenutnu aminokiselinu. Najlogičniji način na koji možemo definisati funkciju prilagođenosti je broj H-H interakcija koje će se dobiti ako trenutnu aminokiselinu savijemo u određenom pravcu. Postoje dva osnovna slučaja na koje treba obratiti pažnju:

- Ako je naredna aminokiselina P, neće biti novih H-H interakcija. Onda je $\eta_{i,j,S} = \eta_{i,j,L} = \eta_{i,j,R} = 0$
- Ako je naredna aminokiselina H, $\eta_{i,j,d}$ će biti jednak broju H-H interakcija nastalih pri savijanju. Mora se proveriti sedam susednih lokacija da bi se izračunao broj H-H interakcija, kao što se može videti na slici 4.2. Broj novih H-H interakcija može biti maksimalno 3 za krajnje tačke sekvence, ili 2 za unutrašnje.

Definišući funkciju prilagođenosti na ovaj način, ona će takođe vršiti i proveru mogućnosti daljeg savijanja, tako što će vratiti 0 ako su sve susedne lokacije popunjene, pa će verovatnoća izbora tog pravca biti 0 (ako nema interakcija vraćamo 1, ako ima, vraćamo $broj\_interakcija + 1$). Dakle, prihvatljive heurističke vrednosti biće:

- 1 – ako nema H-H interakcija
- 2 – ako postoji jedna H-H interakcija
- 3 – ako postoje dve H-H interakcije
- 4 – ako postoje tri H-H interakcije

4.5 Lokalna pretraga

Da bismo poboljšali performanse algoritma *ACO*, uvodimo fazu lokalne pretrage. Faza lokalne pretrage počinje od kompletnog rešenja dobijenog u fazi konstrukcije i pokušava da nađe rešenje sa manjom energijom u nekoj okolini početnog rešenja. Možemo posmatrati dve vrste okoline:

- ***Okolina zasnovana na mutaciji tačke*** – gde se dva rešenja razlikuju samo za pravac u jednoj tački
- ***Okolina zasnovana na makromutacijama*** – gde se dva rešenja razlikuju za jedan ili vise pravaca.

Ovde je korišćena lokalna pretraga zasnovana na prvom poboljšanju (eng. *First Improvement Local Search*) koji pretražuje okolinu, tražeći bolje rešenje. Ako je nađeno rešenje sa manjom energijom, ono zamejuje trenutno rešenje i lokalna pretraga se nastavlja. Ovi koraci se ponavljaju sve dok ne prođe određen proj iteracija, a da bolje rešenje nije nađeno.

Počinjemo tako što odredimo početnu poziciju i menjamo relativni pravac u toj tački. Nakon toga, pokušavamo da dodamo ostale tačke u istom relativnom pravcu koji su imale i pre mutacije. Ukoliko tačku ne možemo da savijemo u istom pravcu kao ranije, probabilistički biramo novi pravac. U ovoj implementaciji, ukoliko se dođe u situaciju da ne možemo da nastavimo savijanje, ne radi se pretraga unazad već se prijavljuje neuspeh. Ovo je zbog činjenice da je pretraga unazad veoma skupa operacija, a zbog smanjenog broja odluka u ovom delu algoritma, možemo očekivati da većina mogućih konformacija već bude prethodno ispitana.

5 Paralelizacija

5.1 Opšte o arhitekturi

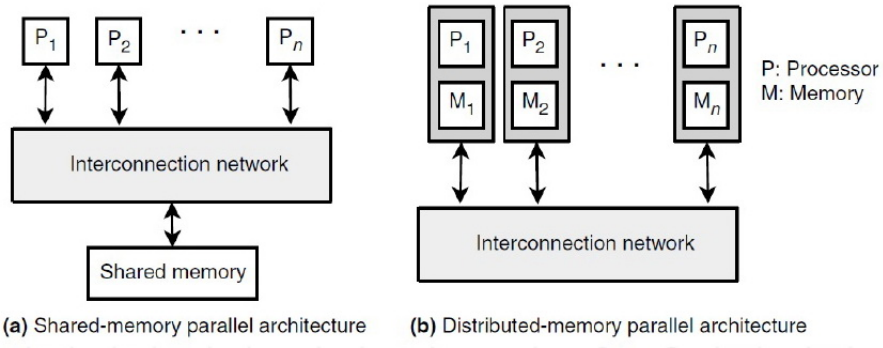
Efikasna implementacija paralelne metaheuristike u opštem slučaju zahteva da se u obzir uzme arhitektura. Razlikujemo sledeće glavne paralelne arhitekture: klasteri/mreže radnih stanica, simetrični multiprocesori i višejezgarni procesori, mreže i GPU. Neke od paralelnih arhitektura mogu se videti na slici 5.1.

Klasteri i mreže radnih stanica (*COW/NOW*) su arhitekture sa distribuiranom memorijom gde svaki procesor ima svoju memoriju. Da bi se informacije razmenjivale među procesorima, potrebno je da se eksplicitno prosleđuju poruke, što otežava programiranje i usporava izvršavanje. U nekim slučajevima, klasteri rade sa nekim od modela deljene memorije, kao što je na primer *NUMA* (*Non-uniform memory access*), gde vreme pristupa zavisi od lokacije memorije u odnosu na procesor. U tom slučaju, klasteri zapravo funkcionišu kao *SMP*.

Simetrični multiprocesori (*SMP*) i višejezgarni procesori su arhitekture sa deljenom memorijom, gde su procesori povezani sa zajedničkom memorijom. Informacije koje se razmenjuju među procesorima smeštene su u određenom adresnom prostoru, ali se i dalje mora vršiti sinhronizacija.

Mreže možemo da posmatramo kao bazene (eng. *pools*) heterogenih i dinamičkih resursa, geografski distribuiranih na više administrativnih domena, u vlasništvu različitih organizacija. Ovi resursi su uglavnom platforme sa visokim performansama, povezanih mrežom velike brzine, ili radne stanice

Slika 5.1: Različiti tipovi paralelnih arhitektura



povezane preko Interneta.

GPU su uređaji koji se koriste u računarima za upravljanje grafikom. Pošto se tehnologija grafičkih kartica značajno razvila zadnjih godina, one se sve više koriste da ubrzaju aplikacije opšte namene, kao što su aplikacije za obradu slika, masovnu obradu podataka, računarske simulacije i mnoge druge. Konvencionalna *NVIDIA* grafička kartica sadrži dosta multiprocesora i procesora, koji izvršavaju više koordinisanih procesa. Na ovom hardveru postoji nekoliko tipova memorije, koji se razlikuju u veličini i brzini pristupa.

5.2 *ACO* nivoi granularnosti

Dekompozicija algoritma *ACO* na manje poslove koji se izvršavaju na različitim procesorima može biti različite granularnosti. Jedan od glavnih ciljeva paralelizacije je da pronađe kompromis između broja poslova i cene upravljanja tim poslovima. Bazirano na strukturi algoritma *ACO*, prepoznaju se četiri nivoa granularnosti: kolonija, iteracija, mrav i rešenje.

Paralelizacija na nivou kolonije se sastoji od izvršavanja celog algoritma *ACO* na jednom procesoru. Postoje dva pristupa: *izvršavanje više nezavisnih kolonija* ili *izvršavanje više kolonija koje sarađuju*. Uglavnom se jedna kolonija dodeljuje jednom procesoru, ali moguće je dodeliti više kolonija jednom procesoru. Na ovom nivou, glavno pitanje na koje treba obratiti pažnju prilikom paralelizacije je homogenost kolonija i interakcija između njih. Homogenost kolonija odnosi se na to da različite kolonije mogu različito da se

ponašaju. Nehomogeni pristup koristi se kod optimizacionih problema sa većim brojem kriterijuma, gde svaka kolonija radi sa drugačijim optimizacionim kriterijumom. Interakcija između kolonija se ostvaruje na jedan od četiri načina[29]:

- ***Razmena globalno najboljeg rešenja*** - pri svakom koraku razmene informacija, računa se globalno najbolje rešenje i šalje se svim kolonijama, u kojima ono takođe postaje lokalno najbolje rešenje.
- ***Kružna razmena lokalnih najboljih rešenja*** - uspostavlja se virtuelna okolina među kolonijama, tako da one formiraju usmeren prsten. U svakom koraku razmene informacija, svaka kolonija šalje svoje lokalno najbolje rešenje samo jednom susedu - narednoj koloniji u prstenu.
- ***Kružna razmena migranata*** - kao i u prethodnom slučaju, kolonije formiraju usmeren prsten. U svakom koraku razmene informacija, svaka kolonija poredi svojih m_b najboljih rešenja sa m_b najboljih rešenja naredne kolonije u prstenu. Najboljih m_b od ovih $2m_b$ rešenja koriste se za ažuriranje matrice feromona.
- ***Kružna razmena lokalnih najboljih rešenja sa dodatkom migranata*** - kombinacija prethodne dve strategije.

Iako se ovakav pristup paralelizaciji pokazao dobar u nekim slučajevima[10], on nije implementiran u ovom radu, iz razloga što autor ovog rada smatra da je ovakav pristup više prilagođen za rad na *CPU* nego na *GPU*.

U zavisnosti od dizajna, paralelizacija na nivou iteracije možemo da posmatramo kao paralelizaciju na nivou kolonije ili paralelizaciju na nivou mrava. Drugim rečima, nivo iteracije možemo da posmatramo kao hibrid između ova dva nivoa. Ideja je da se iteracije algoritma podele među dostupnim procesorima. Jedan način da se implementira ova strategija je da se mravi jedne kolonije podele u grupe i da se omogući svakoj grupi da nezavisno evoluira u toku izvršavanja. Drugi način je da se omogući ovim grupama da dele svoje informacije o feromonima nakon određenog broja iteracija. Na ovom nivou, performanse algoritma zavise najviše od načina na koji se upravlja interakcijama među grupama.

Paralelizacija na nivou mrava podrazumeva distribuciju zadataka (mrava) jedne iteracije među dostupnim procesorima. Uglavnom se odnosi na fazu konstrukcije, ali takođe može biti primenjena na fazu ažuriranja feromona. Ovaj nivo se odnosi na tipičnu strategiju paralelizacije algoritma *ACO*, gde

se jedan ili više mrava dodeljuju svakom procesoru. U tom slučaju, moramo da obratimo pažnju na to da ažuriranje feromona i opšte upravljanje operacijama ne umanje previše performanse algoritma. Glavni potencijalni problem kod ovog pristupa je eventualna potreba za čestom komunikacijom među mravima, ukoliko se rešenja svih mrava konstruktora (ili većine) koriste da ažuriraju matricu feromona. Ovaj pristup korišćen je u ovom radu.

Sve do pre par godina, paralelizacija na nivou mrava bila je strategija paralelizacije sa najmanjom granularnošću. Ipak, pojavljivanje paralelnih arhitektura kao što je *GPU* doveli su i do finijeg pristupa, tj. do paralelizacije na nivou elementa rešenja. U tom slučaju, glavne operacije koje se uzimaju u obzir prilikom paralelizacije su pravilo tranzicije stanja (eng. *state transition rule*) i evaluaacija rešenja. U prvom slučaju, jedna moguća strategija je paralelna evaluacija većeg broja kandidata kako bi se ubrzao proces određivanja sledećeg koraka za svakog mrava. U drugom slučaju, evaluacija ciljne funkcije pojedinačnog mrava je podeljena većem broju procesora.

5.3 Računski entiteti

Danas, tipični paralelni računar visokih performansi je hijerarhija nekoliko različitih arhitektura. Na primer, nije neuobičajeno naići na klastere sa više distribuiranih SMP čvorova, od kojih se svaki sastoji od višejezgarnih procesora i grafičkih kartica. Kako bismo postigli najbolje rezultate na ovim platformama, algoritam mora biti implementiran u skladu sa ovom hijerarhijom. Ovde se mogu uočiti pet računskih entiteta: sistem, čvor, proces, blok i nit.

Sistem definiše paralelni objedinjen računski resurs, koji može biti standardna radna stanica ili klaster. Postoji razlika između ovih sistema i mreža, koje se posmatraju kao više sistema.

Čvor je deo sistema kome se mogu dodeliti zadaci. Sistem se može sastojati od jednog čvora (kao što je slučaj sa radnim stanicama), ili od više čvorova (kao što je slučaj sa klasterima).

Proces je entitet koji upravlja i izvršava sekvencijalni ili paralelni program. Može da sadrži jednu ili više niti, koje mogu (a ne moraju) biti grupisane. Kada proces izvršava samo sekvencijalni kod, onda se smatra za najmanji nedeljivi entitet implementacije.

Blok je entitet između procesa i niti. Uglavnom ga srećemo kod grafičkih kartica, gde predstavlja skup više niti koje izvršavaju isti kod nad različitim podacima

Nit je sekvencijalni tok instrukcija. Predstavlja nedeljivi entitet. Iako u praksi može biti više niti od procesora (u tom slučaju, neke niti će se izvršavati dok druge miruju), u ovom modelu mi pretpostavljamo da ove niti mogu biti spojene u manju grupu niti, tako da broj grupa odgovara broju procesora.

5.4 Memorija

Memorija je važan aspekt algoritama *ACO*. Služi za čuvanje informacija o feromonima, podatke o problemu i različite parametre. Pošto će mogućnost i brzina pristupa imati značajan uticaj na performanse paralelne implementacije, memoriju delimo na tri vrste: lokalna, globalna i udaljena (eng. *remote*).

Lokalna memorija predstavlja memorijski prostor kome entiteti određenog nivoa mogu neposredno pristupati na brz način (relativno u odnosu na nivo entiteta). Na primer, deljena memorija jedne grafičke kartice smatra se lokalnom memorijom za sve niti u jednom bloku multiprocesora. Procesorski registri bi se takođe mogli smatrati lokalnom memorijom, da se njima neposredno upravlja, što uglavnom nije slučaj.

Globalna memorija predstavlja memorijski prostor kome se može neposredno pristupiti, ali ima relativno veliko vreme pristupa. Na primer, device memorija na grafičkoj kartici smatra se globalnom memorijom za niti istog bloka.

Udaljena memorija je memorijski prostor kome se ne može neposredno pristupati, ali iz koje informacije mogu postati dostupne uz pomoć eksplicitnih operacija među entitetima. Očigledno, udaljena memorija je sporija od globalne memorije. Na primer, memorija dostupna procesoru jednog čvora smatraće se udaljenom memorijom za procesore svih drugih čvorova.

5.5 *CUDA*

CUDA (eng. *Compute Unified Device Architecture*) je specifikacija arhitekture uređaja koja omogućava drastična poboljšanja performansi korišćenjem

grafičke jedinice za izračunavanja. Razvijena je od strane korporacije NVIDIA 2006. godine. Koristi se u mnogim oblastima koje zahtevaju intenzivna izračunavanja, kao što su procesiranje slika, biologija i hemija, simulacija dinamike fluida, seizmička analiza i mnoge druge.

Platforma *CUDA* se sastoji iz kompajlera *NVCC* (*Nvidia CUDA Compiler*) i skupa alata za paralelno programiranje na grafičkim karticama. *CUDA* programi izvršavaju se na *CPU* i na *GPU*. *NVCC* prepoznaje ove delove i host kod (kod koji se izvršava na *CPU*) šalje C/C++ kompajleru, dok device kod (kod koji se izvršava na *GPU*) kompajlira sam *NVCC*.

CUDA API predstavlja proširenje programskog jezika C. Ovo proširenje uvodi nove ključne reči, tipove podataka, biblioteke, kao i nešto izmenjenu sintaksu.

Ovde nećemu dublje da ulazimo u način *CUDA* programiranja, već će samo ukratko biti prikazani osnovni arhitekturalni principi i ograničenja koja mogu da utiču na performanse programa.

5.5.1 Razlike izmedju CPU i GPU

Grafičke kartice i centralni procesori su arhitekturalno veoma različiti uređaji. *CPU* je dizajniran da izvršava mali broj relativno kompleksnih zadataka. Grafičke kartice su dizajnirane da izvršavaju veliki broj relativno jednostavnih zadataka. Glavni cilj grafičkih kartica je da rešavaju probleme koji se mogu razbiti na veliki broj sitnih fragmenata, koji se mogu individualno rešavati.

Centralni procesori i grafičke kartice podržavaju niti na različite načine. *CPU* ima mali broj registara po jezgru, koji se koriste da bi se izvršio bilo koji zadatak. Da bi ovo postigli, oni često menjaju kontekst. Menjanje konteksta je vremenski skupo, jer ceo skup registara mora biti prepisan u *RAM*, dok se skup registara drugog zadatka mora prepisati u procesor. Grafičke kartice takođe koriste koncept menjanja konteksta, ali umesto jednog skupa registara, one imaju više nezavisnih skupova registara. Zbog ovoga se promena konteksta izvršava daleko brže, jer je potrebno samo promeniti selektor skupa registara, umesto da se podaci prepisuju u *RAM*.

Jedna od glavnih razlika između *CPU* i *GPU* je ukupan broj procesora na svakom uređaju, što se može videti na slici 5.3. *CPU* uglavnom ima mali broj procesora, dok sa druge strane *GPU* imaju veliki broj jednostavnih pro-

cesorskih jezgara *SP* (eng. *Streaming Processors*), koji su grupisani u multiprocesore *SM* (eng. *Streaming Multiprocessors*), što omogućava grafičkim karticama da postignu visok stepen paralelizacije. Shematski prikaz arhitekture *GPU* prikazan je na slici 5.2. Štaviše, grupisanje *SP* u *SM* nije unapred predefinisano već može da se izvodi dinamički, prema potrebi.

Iako na grafičkim karticama imamo veliki broj procesora, one ne bi bile u stanju da postignu veliko ubrzanje da nisu opremljene odgovarajućom memorijom. Činjenica da se računske operacije mogu obavljati izuzetno brzo na *GPU* ne bi nam bila od velike koristi ukoliko memorija ne bi bila u stanju to da isprati. Zbog toga se na GPU uglavnom koriste brzi tipovi memorije, kao što su *DDR3* ili *GDDR5*.

5.5.2 Grananje

Grananje dovodi do divergencije u toku izvršavanja. Na *CPU* postoji kompleksan hardver koji može da predvidi grananje, na osnovu prethodnih izvršavanja. Na taj način *CPU* unapred kopira instrukcije iz memorije. Pod pretpostavkom da je predikcija ispravna, *CPU* izbegava kašnjenje.

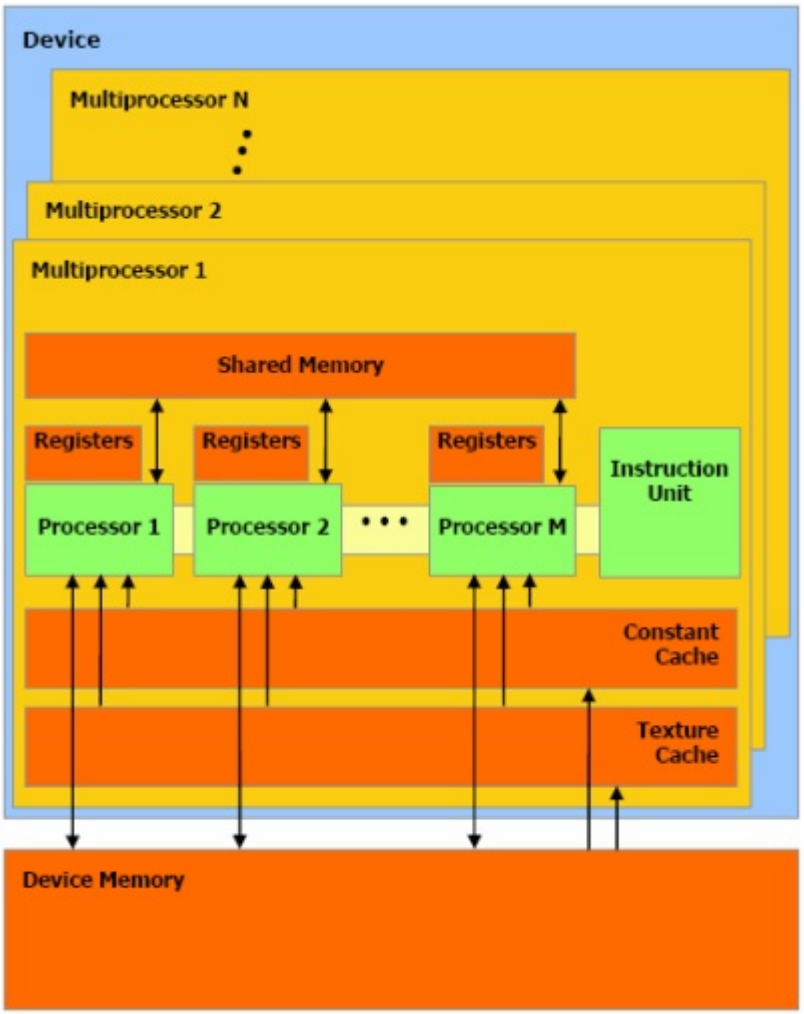
GPU je mnogo jednostavniji uređaj. On prvo izvršava jednu granu, a zatim drugu. U prvom prolazu se izvršavaju one niti koje zadovoljavaju uslov, dok se druge niti označavaju kao neaktivne. Nakon što se one izvrše, izvršava se i druga strana grananja, nakon čega niti ponovo konvergiraju.

Posmatrajmo sliku 5.4. Nakon što se odredi vrednost uslova za svaku od niti, može doći do divergencije, osim ako sve niti imaju istu vrednost uslova. Pošto hardver *GPU* može da obezbedi samo jedan tok instrukcija po grupi niti (koja se naziva vorp, engl. *warp*), jedan deo procesa će da čeka dok drugi deo ne završi svoj posao. Ovo je očigledno veoma loše, pošto je u tom slučaju prosečna iskorišćenost uređaja samo 50%. Ovaj problem se prevazilazi tako što se programi pišu tako da imaju što manje uslovnih naredbi i sa što jednostavnijim uslovnim granama. Posebno je važno da grane koja se ređe izvršavaju budu što jednostavnije, tj. što kraće.

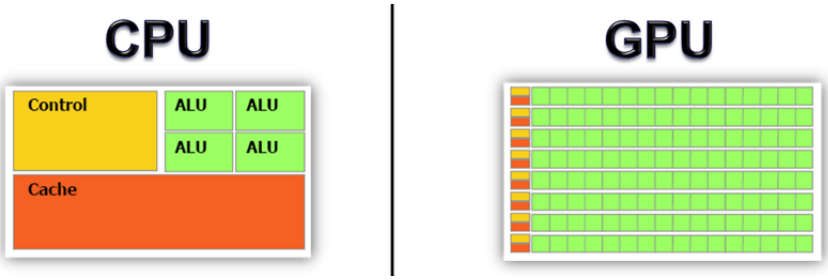
5.5.3 Tipovi memorije

Na grafičkim karticama postoji više tipova memorije, koji se razlikuju po veličini i vremeni pristupa. Najbrži tip memorije su registri unutar uređaja. Nešto sporija od njih je deljena memorija (eng. *shared memory*), a još sporija je globalna memorija. Na slici 5.5 mogu se videti različiti tipovi memorije i

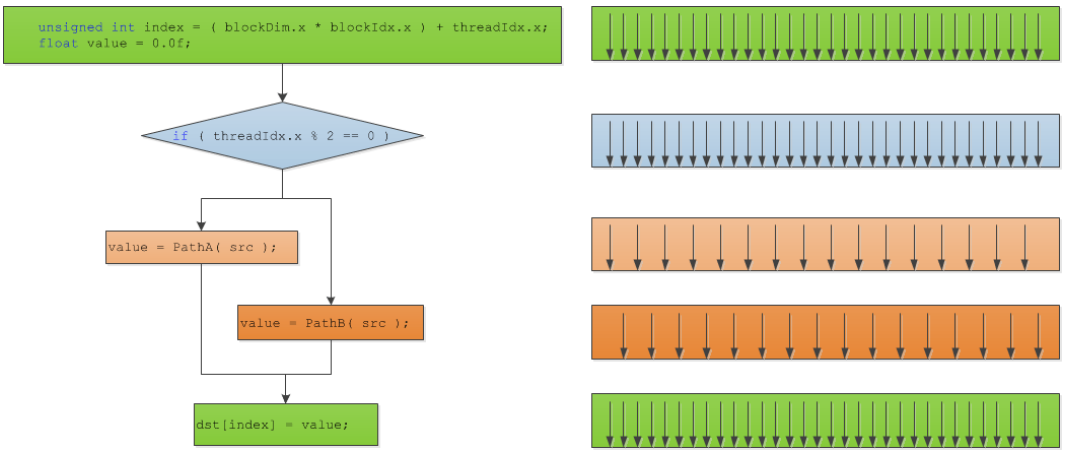
Slika 5.2: Shematski prikaz GPU



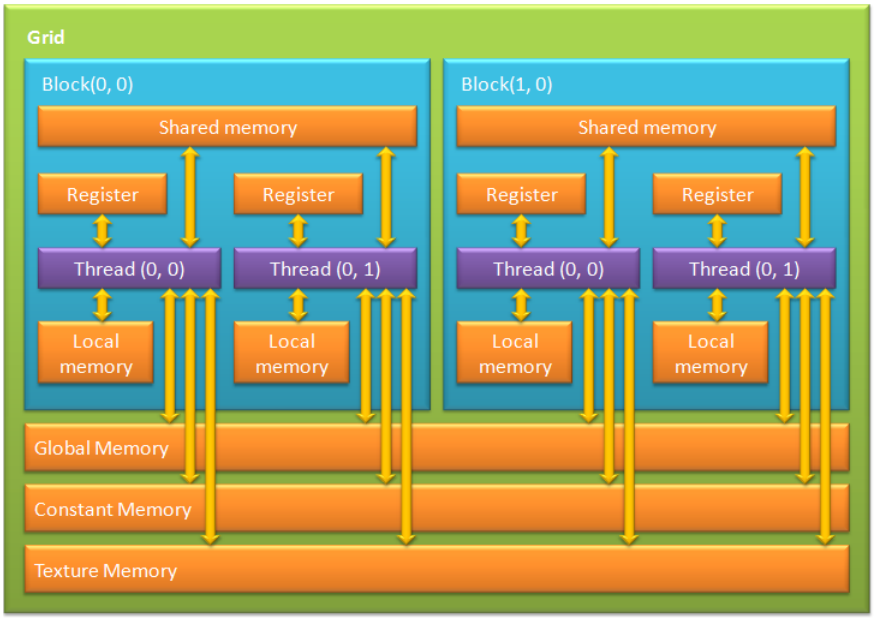
Slika 5.3: Razlika između CPU i GPU



Slika 5.4: Divergencija pri grananju



Slika 5.5: Različiti tipovi memorije na grafičkim karticama



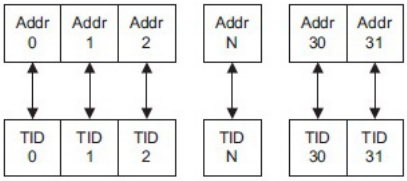
odakle im se može pristupati.

Većina *CUDA* programa se piše progresivno, tako što se u početku koristi samo globalna memorija. Nakon što isprogramiramo početnu verziju programa, uvodimo ostale tipove memorije, kao što su zero-copy i deljena memorija, konstantna memorija, a na kraju razmatramo i upotrebu registara. To se radi tako da se, sa jedne strane, dobije što manje prebacivanja podataka, a da sa druge strane, podaci koji se obrađuju budu u što bržoj memoriji.

GPU ima na hiljade registara po svakom *SM* (eng. *streaming multiprocessor*). *SM* se može posmatrati kao višenitno *CPU* jezgro. Za razliku od *CPU*, gde imamo 2, 4, 6 ili 8 jezgara, *GPU* ima veliki broj *SM* jezgara.

Deljena memorija je korisnički kontrolisan *L1* keš. L1 keš i deljena memorija dele 64k memorijskog prostora po *SM*, na svim grafičkim karticama sa *CUDA* računskom sposobnošću (eng. *CUDA compute capability*) makar verzije 2.0. *GPU* koristi *load-store* model memorije, tako da se svaki operand mora da se učita u registar pre svake operacije. Zbog ovoga se učitavanje podataka u deljenu memoriju mora da se opravda čestim korišćenjem podataka

Slika 5.6: Poželjan način pristupanja globalnoj memoriji



ili deljenjem podataka među nitima, inače bi bilo brže neposredno učitati podatke iz globalne memorije u registar.

U globalnu memoriju se možmo da pišemo i iz jezgra[1] i sa hosta. Može biti adresirana od strane bilo kog uređaja na *PCI-E* magistrali. Na ovaj način, grafičke kartice mogu da šalju podatke između sebe, bez pomoći *CPU*. Iako je globalna memorija dosta sporija od drugih tipova memorije, njeno kašnjenje se možemo da maskiramo ukoliko joj pristupamo po objedinjujućem šablonu (eng. *coalesced pattern*). Ovaj šablon podrazumeva da sve niti redom pristupaju uzastopnim lokacijama u bloku memorije, kao što je prikazano na slici 5.6. Ukoliko imamo ovakav sekvencijalan i poravnat pristup memoriji, svi zahtevi biće objedinjeni u samo jedan zahtev memoriji. Poravnanje memorije se postiže korišćenjem specijalne malloc instrukcije:

```
extern __host__ cudaError_t CUDARTAPI cudaMallocPitch(
void **devPtr, size_t *pitch, size_t width, size_t height);
```

Ova instrukcija alocira onoliko memorije preko tražene, koliko je potrebno da bi memorija bila poravnata. Iako nam ovakav pristup omogućava dosta manje kašnjenje, on nije bio primenljiv na naš slučaj, jer zbog probabilističke prirode algoritma *ACO*, ne možemo da garantujemo da će sve niti da pristupaju upravo onim podacima kojima bi trebale da pristupaju kako bi ovakav pristup funkcionisao.

5.6 Alternativna ideja za paralelizaciju

Ovde se postavlja pitanje da li možemo izvrsiti paralelizacija na nivou faze konstrukcije. Pokušavamo da nađemo način da rešenja konstruišemo

[1]Jezgro (eng. *kernel*) je C funkcija koja se izvršava na grafičkoj kartici. Definisanje ovakvih funkcija vrši se pomoću ključne reči *__global__*, ne podržava rekurziju i povratna vrednost mora biti tipa *void*.

Tabela 5.1: Vreme pristupa različitih tipova memorije

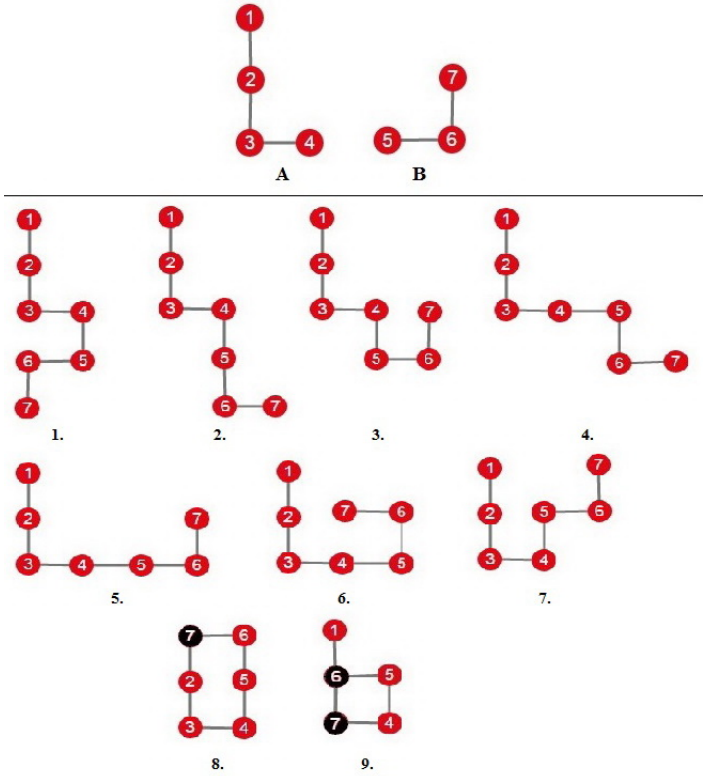
Tip memorije	Registri	Deljena memorija	Globalna memorija
Propusni opseg	∼8 TB/s	∼1.5 TB/s	∼200 MB/s
Kašnjenje	1 ciklus	1 do 32 ciklusa	∼400 do 600

sekvencijalno, ali tako da se akcije unutar faze konstrukcije izvršavaju paralelno. Problem kod ovakvog pristupa je to što je faza konstrukcije po prirodi sekvencijalna. Da bismo odredili na koju će se stranu saviti sledeća aminokiselina, moramo da uzmemo u obzir dve informacije: informaciju o feromonima i informaciju o heuristici. Za razliku od nekih drugih problema na koje se primenjuje *ACO*, vrednost funkcije prilagođenosti u potpunosti zavisi od prethodnih koraka. Ipak, moguće bi bilo konstruisati algoritam sa nešto izmenjenom funkcijom prilagođenosti, koja bi uzimala u obrir broj svih novih H-H interakcija koje nastaju nadovezivanjem dve podsekvence na određeni način.

Za konstrukciju ovog algoritma najbolje je da se koristi metoda "podeli pa vladaj". Neka je sekvenca dužine n. Dodeljujemo po dve uzastopne aminokiseline svakom od $\frac{n}{2}$ procesora (ukoliko je broj procesora manji od $\frac{n}{2}$, možemo dodeliti veći broj aminokiselina svakom procesoru i sekvencijalno saviti tu podsekvencu). Nakon što svaki procesor odradi savijanje svoje podsekvence, sledi faza objedinjavanja. Algoritam pokušava da sastavi dve savijene parcijalne konformacije tako što se probabilistički bira relativni pravac u kome se one nadovezuju.

Dve parcijalne konformacije se mogu nadovezati na najviše devet načina, kao što je prikazano na slici 5.7. Od svih mogucih načina koji ne dovode do kolizije, bira se onaj koji za rezultat ima najmanju energiju. Ukoliko ne postoji nijedan pravac koji ne dovodi do kolizije, potrebno je izvršiti bektreking.

Slika 5.7: Načini nadovezivanja dve podsekvence



Složenost ovakvog algoritma je

$$T(n) = 2T\left(\frac{n}{2}\right) + 9\left(\left(\frac{n}{2}\right)^2\right)$$

pa je po Master teoremi kompleksnost algoritma $O(n^2)$. Sa druge strane, kompleksnost klasične faze konstrukcije je $\mathcal{O}(7n) = \mathcal{O}(n)$.

Takođe, dosta je komplikovanije odraditi bektreking ukoliko se koristi ovaj pristup. Da bi se on mogao efikasno izvršiti, potrebno je $n\ log_2(n)$ dodatnih memorijskih lokacija, za pamćenje međurezultata.

Iz ovog razloga, ovakav algoritam nije implementiran.

6 Implementacija

6.1 Osnovno o programu

U okviru ovog rada implementirane su dve verzije algoritma: sekvencijalna i paralelna. Sekvencijalna verzija zasnovana je većim delom na postojećem radu [4]. Program se sastoji iz parsera (koji klasifikuje aminokiseline i ulaznu sekvencu pretvara u *HP* sekvencu) i algoritma *ACO*. Svaki mrav obavlja svoj zadatak sekvencijalno.

Kod paralelne implementacije, odrađena je implementacija na nivou mrava. Ovakav pristup je najintuitivniji za ovu metaheuristiku, a pokazao se i kao dovoljno efikasan. Svaki mrav dodeljuje se jednoj niti na grafičkoj kartici, na kojoj čuva sopstvenu verziju rešetke, trenutnu energiju, putanju i informacije potrebne da bi se izvršilo vraćanje unazad. Kako bi se izbeglo preterano prenošenje podataka sa *GPU* na *CPU*, korišćen je pristup ažuriranju matrice feromona zasnovan na kvalitetu, u kome se bira samo jedan mrav sa najkvalitetnijim rešenjem, čija se staza se kopira na host.

U okviru rada testirane su dve paralelne implementacije, koje se razlikuju po načinu predstavljanja rešetke. U prvoj verziji, rešetka je predstavljena kao matrica karaktera. Ovakav pristup zauzima više memorije, ali se proveravanje susednih pozicija u funkciji prilagođenosti obavlja u konstantnom vremenu. Druga verzija koristi niz struktura *grid_struct*, koja sadrži koordinate tačke i njenu vrednost. Svaki put kada se dodaje nova aminokiselina trenutnom savijanju, u niz se ubacuje novi element koji odgovara toj aminokiselini. Proveravanje susednih pozicija u funkciji prilagođenosti vrši se pretraživanjem niza. Ukoliko tražena tačka nije pronađena u nizu, onda su njene koordinate slobodne. Sa druge strane, ukoliko pronađemo tačku sa zadatim koordi-

natama u nizu, čitamo vrednost u toj tački. Ovaj pristup zauzima manje memorije, ali je provera susednih pozicija ipak vremenski skuplja zbog pretrage. Zbog smanjene potrošnje memorije, ovakav niz je moguće smestiti u deljenu memoriju, kako bi se donekle smanjilo vreme pretrage.

6.2 Tehničke informacije

Sve verzije programa implementirane su korišćenjem programskih jezika *C++* i *CUDA C*. Za korisnički interfejs korišćen je *Qt*, kako bi program bio kompatibilan sa svim operativnim sistemima.

Za pokretanje programa neophodno je imati *NVidia* grafičku karticu, sa *CUDA* računskom sposobnošću makar verzije 2.0, što zadovoljavaju sve novije *NVidia* grafičke kartice, počevši od grafičkih kartica sa mikroarhitekturom *Fermi*.

Pri testiranju programa za dugačke sekvence pod operativnim sistemom *Windows*, potrebno je isključiti *TDR* (*Timeout Detection & Recovery*). Ovo je svojstvo Windows operativnog sistema koje detektuje zastoj na grafičkim karticama. Ukoliko se grafička kartica neko vreme ne odaziva operativnom sistemu (predefinisano vreme čekanja je 2 sekunde), operativni sistem resetuje grafičku karticu. Ovo vreme je sasvim dovoljno za uobičajene poslove sa kojima se *GPU* susreće, ali problem nastaje ukoliko na *GPU* izvršavamo kompleksne algoritme. U tom slučaju se može desiti da jezgru ponestane vremena za izvršavanje i da bude prekinut od strane operativnog sistema.

6.3 Algoritam

Osnovni paralelni aloritam koji je korišćen može se videti iz sledećih pseudokodova: 2, 3 i 4.

6.4 Prenos podataka

Razmena podataka između centralnog procesora i grafičke kartice može biti usko grlo algoritma paralelizovanog na nivou mrava, pošto je prenos po-

[1]Paralelna redukcija koristi se za pronalaženje minimalne vrednosti u nizu koji sadrži vrednosti energija koje odgovaraju svakom od pronađenih rešenja. Algoritam će biti objašnjen dalje u tekstu.

Algoritam 2: Parallel ACO

input : Sekvenca S, brojMrava, α, β, ρ
output: Konformacija sa minimalnom pronađenom energijom

```
begin
    matricaFeromona ← inicijalizacijaMatriceFeromona();
    najboljeRešenje ← null;
    optimalnaEnergija ← aproksimacijaOptimalneEnergije(S);
    while not kriterijumZaustavljanjaIspunjen() do
        startnePozicije ← odrediti StartnePozicije(brojMrava);
        graditelji ←
        paralelnoPokrenutiGraditelje(S, startnePozicije, α, β);
        lokalnoNajboljeRešenje ← graditelji.vratiNajboljeRešenje();
        while not dostignutBrojIterativnihPoboljsanja() do
            popravljači ← paralelnoPokreniPopravljače(S,
            lokalnoNajboljeRešenje, BrojMrava);
            lokalnoNajboljeRešenje ←
            popravljači.vratiNajboljeRešenje();
        if (energija(lokalnoNajboljeRešenje) <
        energija(najboljeRešenje)) then
            najboljeRešenje ← lokalnoNajboljeRešenje;
        ažurirajTragFeromona(lokalnoNajboljeRešenje,
        optimalnaEnergija, ρ);
    return najboljeRešenje;
```

Algoritam 3: ANT BUILDER

input : Sekvenca S, startnaPozicija, α, β
output: Pronađena konformacija

```
begin
    konformacija ← array();
    inicijalizujHPRešetku();
    inicijalizujPrveDveTačke(startnaPozicija, konformacija);
    while (not sviElementiSekvenceDodati()) do
        pravac ← probabilističkiIzabratiPravacSavijanja(
        funkcijaPrilagođenosti, tragFeromona);
        if pravac == null then
            izvršitiPretraguUnazad();
        else
            dodatiNoviElement(pravac, konformacija);
    izvršitiParalelnuRedukciju()[1]; ;
    if trenutniMravImaNajmanjuEnergiju then
        return konformacija;
```

Algoritam 4: ANT IMPROVER

input : Sekvenca S, startnaPozicija, staraKonformacija
output: Pronađena konformacija

```
begin
    novaKonformacija ← array();
    for i = 0 to startnaPozicija do
        novaKonformacija[i] ← staraKonformacija[i];
    IzvršitiMutaciju();
    dodatiOstaleTačkeUIstomRelativnomPravcuKojiSuImale();
    if tačkaSeNeMožeDodatiUIstomPravcu() then
        IzvršitiMutaciju();
        if tačkaSeNeMožeDodatiNiUJednomPravcu() then
            return null;
    izvršitiParalelnuRedukciju() ;
    if trenutniMravImaNajmanjuEnergiju then
        return novaKonformacija;
```

dataka relativno spor u odnosu na druge operacije. Prenos podataka obavlja se u dva smera: prenos podataka sa *CPU* na *GPU* i prenos podataka sa *GPU* na *CPU*. Što se tiče prenosa podataka sa *CPU* na *GPU*, on nije problematičan. Tada se grafičkoj kartici prosleđuje matrica feromona, sekvenca koja se savija, kao i α i β parametri. Ovi podaci prosleđuju se samo jednom za sve mrave i čuvaju se u globalnoj memoriji.

Sa druge strane, prenos podataka sa *GPU* na *CPU* može biti problematičan, pošto u najgorem slučaju treba vratiti sva rešenja koja su mravi izgradili, zajedno sa pronađenim energijama. Kako bi se ovo izbeglo, odlučeno je da se koristi pristup ažuriranju feromona zasnovan na kvalitetu rešenja, gde samo jedan mrav ažurira matricu feromona. Drugim rečima, potrebno je vratiti rezultate za samo jednog mrava. Ipak, da bi se ovaj pristup implementirao, potrebno je efikasno pronaći koji mrav je pronašao najbolje rešenje, pre vraćanja rezultata na *CPU*. Ovo se postiže tako što svi mravi, nakon završenog procesa konstrukcije, smeštaju nađenu energiju i svoj indeks u pomoćni niz, koji se nalazi u globalnoj memoriji. Zatim se traži najveća vrednost u nizu, korišćenjem algoritma paralelne redukcije. Samo mrav sa najboljom energijom prepisaće svoje rešenje na *CPU*.

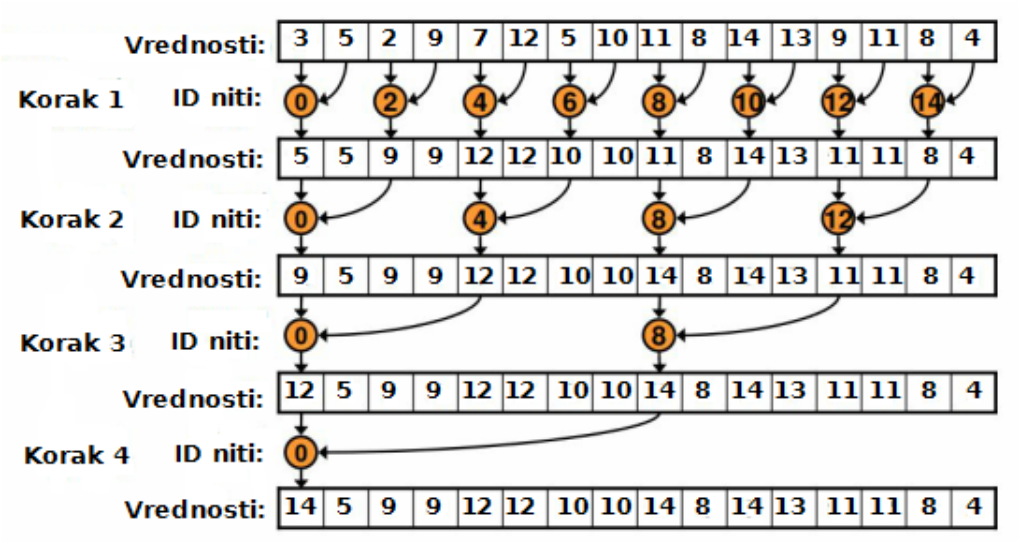
Algoritam paralelne redukcije za traženje maksimalne vrednosti u nizu prikazan je na slici 6.1. Ovaj algoritam može raditi za sve asocijativne operacije, kao što su sabiranje, množenje, konjukcija, disjunkcija, maksimum, minimum i mnoge druge.

6.5 Reprezentacija rešetke

Kao što je već pomenuto, u okviru ovog rada implementirane su dve paralelne verzije programa, koje se razlikuju po načinu na koji predstavljaju *HP* rešetku. Prvi i najintuitivniji način predstavljanja rešetke svakako je predstavljanje uz pomoć matrice. Ovakav pristup zauzima $\mathcal{O}(n^2 + 2(n-2))$ (dodatnih $2(n-2)$ memorijskih lokacija su za čuvanje podataka o stazi i bektreking informacijama) memorijskih lokacija po niti, što u nekim slučajevima može biti neprihvatljivo, ukoliko radimo sa dugačkim sekvencama i velikim brojem niti. Ipak, u većem broju slučajeva čak i ova potrošnja je prihvatljiva, pošto za grafičke kartice sa $2GB$ memorije, možemo imati i do 2000 niti za sekvencu dužine 1000. Pri računanju heurističkih vrednosti, dovoljno je neposredno proveriti okolne tačke. Ovakav pristup prikazan je na slici 6.2.

Drugi pristup bio bi čuvanje informacija u vidu niza. Krećemo od pra-

Slika 6.1: Algoritam paralelne redukcije za pronalaženje elementa sa maksimalnom vrednošću u nizu



Slika 6.2: Savijanje *HP* sekvence korišćenjem matrice

	0	1	2	3	4
0	-	-	H	H	-
1	-	-	P	-	-
2	-	H	P	-	-
3	-	-	-	-	-
4	-	-	-	-	-

znog niza i nakon svakog dodavanja nove tačke, u niz ubacujemo strukturu *grid_ struct* sa odgovarajućim informacijama. Proveravanje vrednosti okolnih tačaka vrši se pretragom niza. Ukoliko u nizu naiđemo na strukturu sa koordinatama koje se traži, možemo pročitati vrednost u toj tački, u suprotnom, ta tačka je slobodna. Ideja ovog pristupa bila je da se omogući izvršavanje za dugačke sekvence, time što će da smanji potrošnju memorije. Za ovakav pristup, potrebno je $\mathcal{O}(9n + 2(n-2)) = \mathcal{O}(n)$ memorijskih lokacija. Iako je početna ideja bila da se ovim pristupom omogući izvršavanje programa za duže sekvence, autor je odlučio da smesti niz sa informacijama o rešetki u deljenu memoriju, kako bi se izvršavanje programa ubrzalo. Ovo očigledno značajno smanjuje mogućnost da se program izvršava za duge sekvence, iz razloga što je deljena memorija dosta manjeg kapaciteta. Dužina sekvence određena je parametrom *MAX_ SEQUENCE_ LENGT*, koji je podrazumevano podešen na 256. Verzija programa koji niz dinamički alocira u globalnoj memoriji nije testirana.

Savijanje sekvence korišćenjem niza može se predstaviti kao:

$$\{((2,1),H),((2,2),P),((1,2),P),((0,2),H),((0,3),H)\}$$

```
struct grid_struct{
        Point coordinates;
        char value;
};
```

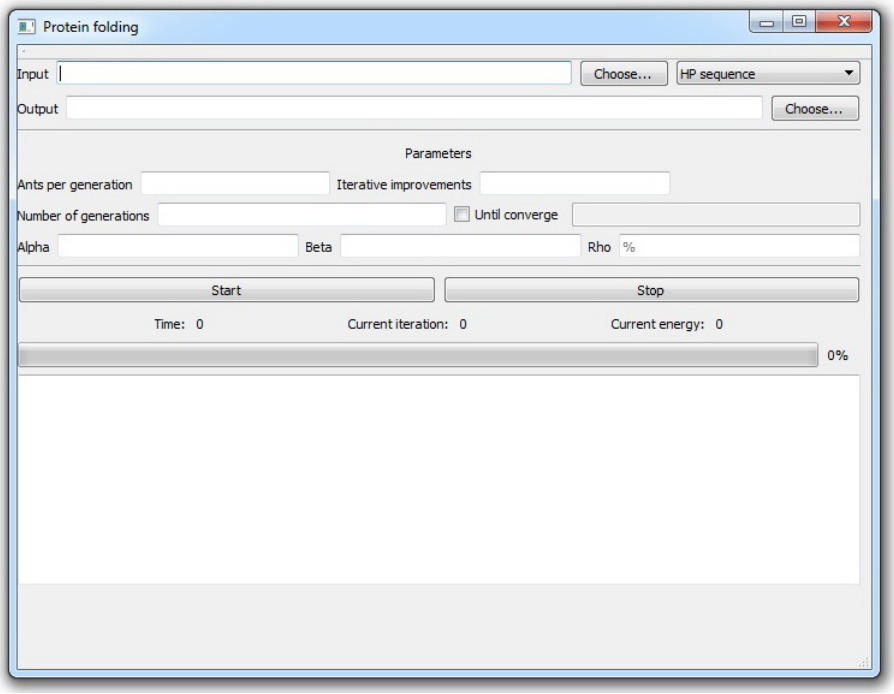
Razlog ovog prebacivanja predmeta interesovanja sa omogućavanja rada za duge sekvence na brzinu izvršavanja bila je hipoteza da se program sa ovakvom implementacijom može izvršavati još brže od implementacije koja koristi matricu. Naime, pri svakom dodavanju nove tačke potrebno je proveriti makar 7 okolnih lokacija, što se postiže pretragom niza. Ipak, pošto je dužina niza uvek manja od n, imalo bi smisla uzeti da se u proseku pretražuje niz dužine $\frac{n}{2}$. Zato bismo mogli reći da algoritam izvršava $7(n-2)(\frac{n}{2}) = (\frac{7n^2}{2}) - 7n$ instrukcija (ovo je ne računajući sve dodatne instrukcije, koje su iste za obe verzije algoritma). Sa druge strane, implementacija koja koristi matricu proverava okolne pozicije u konstantnom vremenu, tako da nam za proveru 7 lokacija treba 7 instrukcija. Problem kod ovakvog pristupa je što se matrica mora inicijalizovati pre upotrebe, što zahteva n^2 instrukcija. Dakle, kod ove verzije imamo da nam je potrebno $7(n-2) + n^2$ instrukcija. Uzmemo li još da je niz smešten u deljenoj memoriji, razumno je očekivati da će se ova implementacija brže izvršavati.

Ipak, rezultati testiranja su drugačiji nego što je očekivano. Možemo videti da se verzija koja implementira matricu zapravo brže izvršava. Ovo je posledica negativnog efekta grananja, koje više utiče na verziju koja radi sa nizom, nego na verziju koja radi sa matricom. Ovo je zbog vrste koda koji se izvršava, jer ukoliko moramo da pretražujemo lokalni niz, broj grananja je veliki, pa je izvršavanje višestruko sporije. Na primer, na dužini 7 imamo i do 2^7 grananja, tj. 128 različitih putanja koje moraju da se izvrše, što je, prirodno, 128 puta sporije nego kod bez grananja. Takođe, ukoliko vraćanjem izbacimo k već ubačenih elemenata, verzija sa matricom moraće da izvrši dodatnih $7k$ instrukcija (po 7 provera za svaki element), dok će verzija sa nizom morati da izvrši u proseku $7k(\frac{n}{2})$ instrukcija (po 7 pretraga niza za svaki element, pod pretpostavkom da je prosečna duina niza $\frac{n}{2}$).

6.6 Interfejs

Na slici 6.3 može se videti grafički korisnički interfejs.

Slika 6.3: Interfejs



U polje *Input* unosi se adresa tekstualne datoteke koja sadrži sekvencu. Postoje tri izbora pri učitavanju sekvence:

- *HP sequence* - u datoteci se nalazi sekvenca u obliku *HP*.
- *Amino acids (Hydrophobic)* - u datoteci se nalazi sekvenca aminokiselina, koja se prevodi u *HP* sekvencu. Aminokiseline R, K, D i E klasifikuju se kao hidrofobne prilikom prevođenja.
  Primer: $RKDNLHTVYGF \rightarrow HHHPHPPHPHH$
- *Amino acids (Polar)* - u datoteci se nalazi sekvenca aminokiselina, koja se prevodi u *HP* sekvencu. Aminokiseline R, K, D i E klasifikuju se kao polarne prilikom prevođenja.
  Primer: $RKDNLHTVYGF \rightarrow PPPPHPPHPHH$

U polje *Output* unosi se adresa tekstualne datoteke u koju će se upisati rezultat savijanja proteina. Ovo polje je opciono.

U polje *Ants per generation* unosi se broj mrava po generaciji. Broj mrava trebao bi biti deljiv sa 32.

U polje *Iterative improvements* unosi se broj iterativnih poboljšanja po generaciji. Drugim rečima, tek nakon ovog broja pozivanja mrava popravljača bez ikakvog poboljšanja energije će algoritam preći na sledeću generaciju.

U polje *Number of generations* unosi se broj iteracija algoritma. Umesto da se ograniči broj iteracija, moguće je štiklirati polje *Until converge*, a zatim uneti željeni procenat konvergencije. U tom slučaju, algoritam će nastaviti sa izvršavanjem sve dok energija rešenja ne dostigne uneti procenat procenjene optimalne energije. Ovde treba biti obazriv, pošto procenjena optimalna energija ne mora odgovarati pravoj optimalnoj energiji, zbog čega nije preporučljivo unositi vrednost 100.

U polja *Alpha*, *Beta* i *Rho* unose se parametri α, β i ρ, čija je funkcija već opisana.

Dugme *Start* pokreće izvršavanje programa, dok dugme *Stop* prekida program i ispisuje trenutno najbolje rešenje.

7 Rezultati

7.1 Izbor parametara

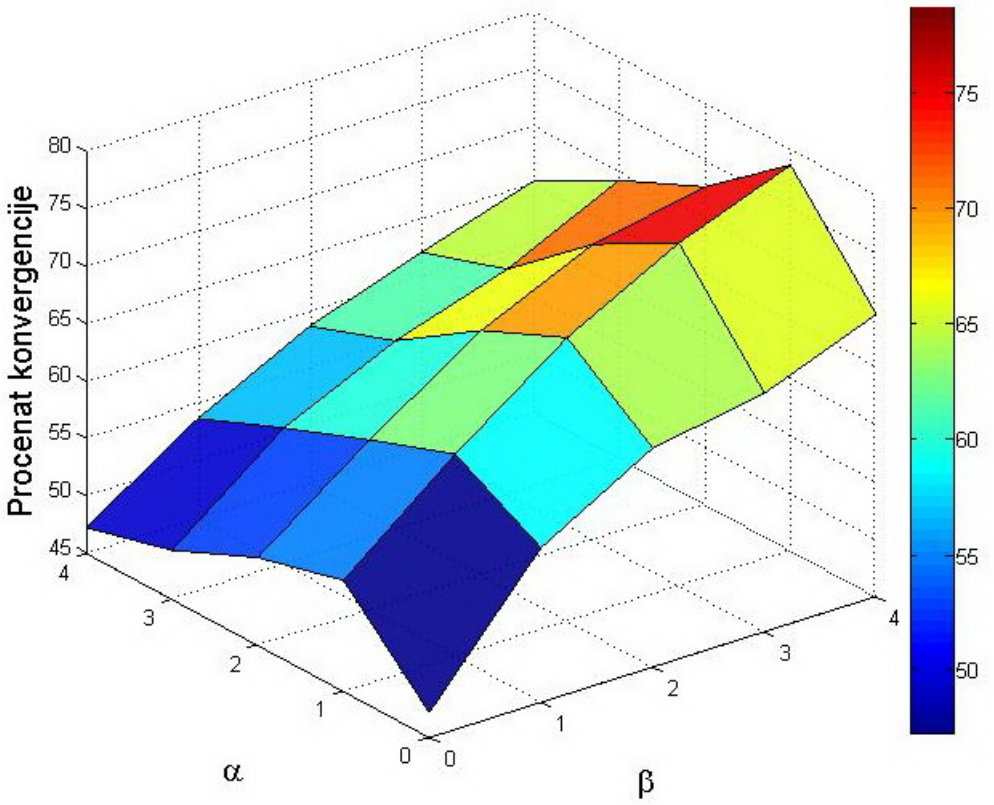
Kako bi se dobili što bolji rezultati, potrebno je da dobro izaberemo parametre α, β i ρ. Izbor ovih parametara dosta utiče na brzinu kojom program konvergira ka optimalnom rešenju.

Za nalaženje najboljih vrednosti parametara α i β, posmatran je prosečan stepen konvergencije deset nezavisnih pokretanja programa, primenjen na svaku sekvencu iz tabele A.1. Iz praktičnih razloga, testiranje je bilo vremenski ograničeno. Ideja je bila da čak i vremenski ograničena izvršavanja mogu da daju korisne informacije o brzini konvergencije. Rezultati se mogu videti na slici 7.1 ili u tabeli 7.1. Može se primetiti trend poboljšanja rezultata kada povećavamo parametar β, za svaku fiksiranu vrednost α. Rezultati za vrednosti parametra β veće od 4 nisu prikazane na slici, jer iako daju dobre rezultate na ovom testu (u proseku iste kao i za $\beta = 4$), u praki se pokazalo da program ipak donekle sporije konvergira za te vrednosti. Autor ovog teksta preporučuje korišćenje vrednosti parametara $\alpha = 1$ i $\beta = 4$. Ovakvi rezultati su u skladu sa drugim istraživanjima [4][15].

Tabela 7.1: Prosečan stepen konvergencije za različite vrednosti parametara α i β. Prikazani procenti su prosek rezultata svih sekvenci iz tabele A.1

		β				
		0	1	2	3	4
α	0	47.18%	58.46%	64.06%	65.77%	69.56%
	1	54.69%	62.6%	69.67%	74.83%	78.54%
	2	52.67%	59.83%	66.28%	70.71%	72.63%
	3	49.29%	56.86%	61.38%	64.54%	69.2%
	4	47.26%	53.68%	58.64%	61.99%	65.11%

Slika 7.1: Prosečan stepen konvergencije za različite kombinacije α i β parametara



Testiranje parametra ρ izvršeno je na dva načina. Za testiranje sekvenci od 1 do 13 iz tabele A.1 uzeto je prosečno vreme konvergencije algoritma izraženo u sekundama. Svi rezultati se mogu videti u tabeli 7.2, dok je

na slici 7.2 prikazano prosečno vreme izvršavanja za različite vrednosti parametra ρ. Kako bi sve sekvence imale isti uticaj na rezultate prikazane na slici 7.2, rezultati za svaku sekvencu su skalirani na taj način da najkraće vreme izvršavanja za svaku sekvencu ima vrednost 1, dok sve ostale vrednosti predstavljaju odnos vremena izvršavanja za određenu vrednost parametra ρ i minimalnog vremena izvršavanja. Prema tome, vrednosti bliže jedinici predstavljaju poželjnije vrednosti parametra ρ. Sa druge strane, iz praktičnih razloga, sekvence od 14 do 20 iz tabele A.1 testirane su sa vremenskim ograničenjem od pola sata, poređenjem stepena konvergencije za različite vrednosti parametra ρ (odnos pronađene energije u odnosu na optimalnu energiju). U oba slučaja, uzet je prosek rezultata pet nezavisnih izvršavanja programa za svaku sekvencu i svaku vrednost parametra ρ. U tabeli 7.3 prikazani su svi dobijeni podaci za ovaj pristup, dok se na slici 7.3 može videti prosečan stepen konvergencije za svaku vrednost parametra ρ.

Rezultate sa slike 7.2 ipak treba uzimati ozbiljnije nego rezultate sa slike 7.3, zbog samog načina testiranja. Kombinovanim posmatranjem ove dve slike možemo zaključiti da je najbolja vrednost za parametar $\rho = 0.175$, pošto je na oba testa dao dobre rezultate.

Tabela 7.2: Prosečno vreme konvergencije za različite vrednosti parametra ρ izraženo u sekundama

	0.05	0.075	0.1	0.125	0.15	0.175	0.2	0.225	0.25	0.275	0.3
1	728	667	674	537	546	***510***	624	598	678	721	690
2	689	654	613	***545***	573	568	526	607	593	648	658
3	638	612	595	498	***521***	534	545	539	596	612	675
4	892	756	698	574	598	***542***	570	785	645	697	743
5	846	852	692	758	656	676	***633***	656	785	732	841
6	1023	895	834	***621***	691	726	734	689	798	789	897
7	715	693	766	472	***423***	678	632	1073	967	864	1423
8	2351	2102	1955	1865	1768	***1721***	1856	1823	1984	2067	2132
9	1956	1754	1523	1548	***1321***	1398	1495	1468	1689	1745	1856
10	2295	2195	2103	2064	***1823***	1954	2045	2215	2102	2234	2345
11	737	689	598	***545***	648	624	765	759	895	964	998
12	754	701	598	534	495	***428***	512	493	567	548	645
13	3152	2865	2405	2954	***2350***	2428	2799	3012.2	2676	3121	3056

Slika 7.2: Prosečno vreme izvršavanja za različite vrednosti parametra ρ, testirano na sekvencama od 1 do 13 iz tabele A.1, skalirano u odnosu na minimalno vreme izvršavanja.

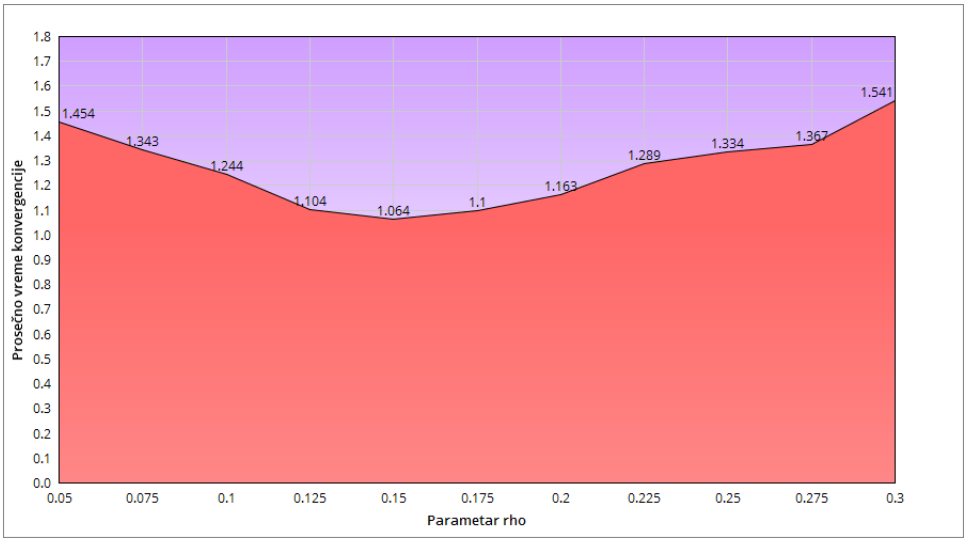
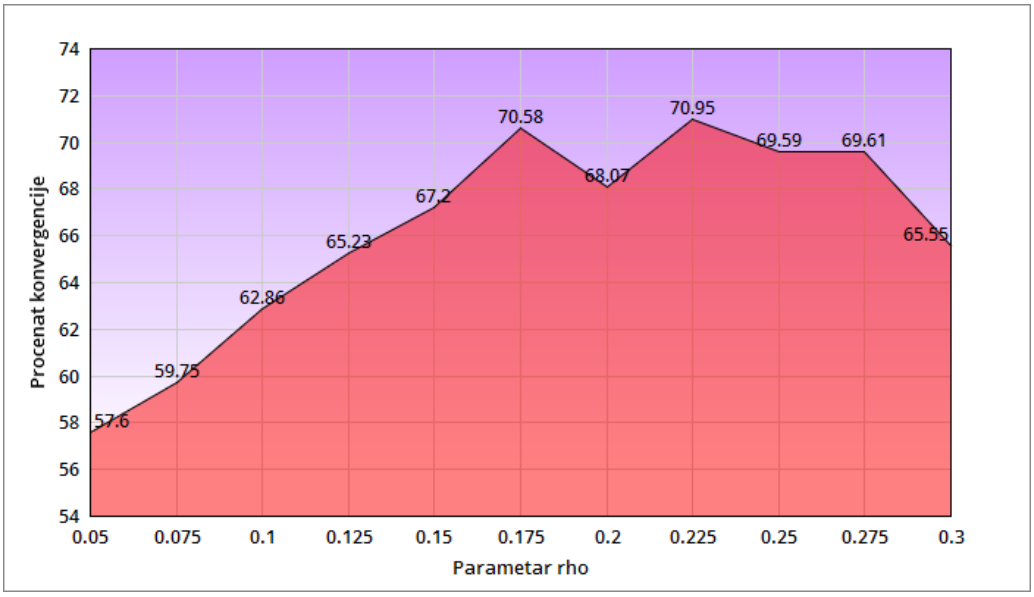


Tabela 7.3: Prosečan procenat konvergencije za različite vrednosti parametra ρ, testirano na sekvencama 14-20 iz tabele A.1

	0.05	0.075	0.1	0.125	0.15	0.175	0.2	0.225	0.25	0.275	0.3
14	52.10%	56.50%	65.20%	73.04%	73.90%	73.90%	69.60%	72.18%	69.23%	73.90%	65.20%
15	52.00%	61.90%	62.10%	66.60%	69.46%	80.90%	67.56%	71.40%	70.44%	70.25%	66.60%
16	72.00%	73.80%	75.00%	75.00%	76.12%	80.56%	83.30%	78.90%	78.35%	77.80%	75.60%
17	50.00%	52.40%	52.40%	59.50%	64.30%	61.90%	59.60%	69.10%	66.70%	64.30%	59.50%
18	56.60%	58.50%	66.00%	64.10%	66.00%	66.10%	67.90%	69.90%	71.70%	66.10%	64.10%
19	58.00%	56.00%	60.00%	58.00%	56.00%	64.00%	62.40%	68.00%	63.60%	64.00%	62.00%
20	62.50%	59.14%	58.30%	60.40%	64.60%	66.70%	68.80%	67.20%	67.12%	70.90%	65.86%

Slika 7.3: Prosečan procenat konvergencije za različite vrednosti parametra ρ, testirano na sekvencama 14-20 iz tabele A.1



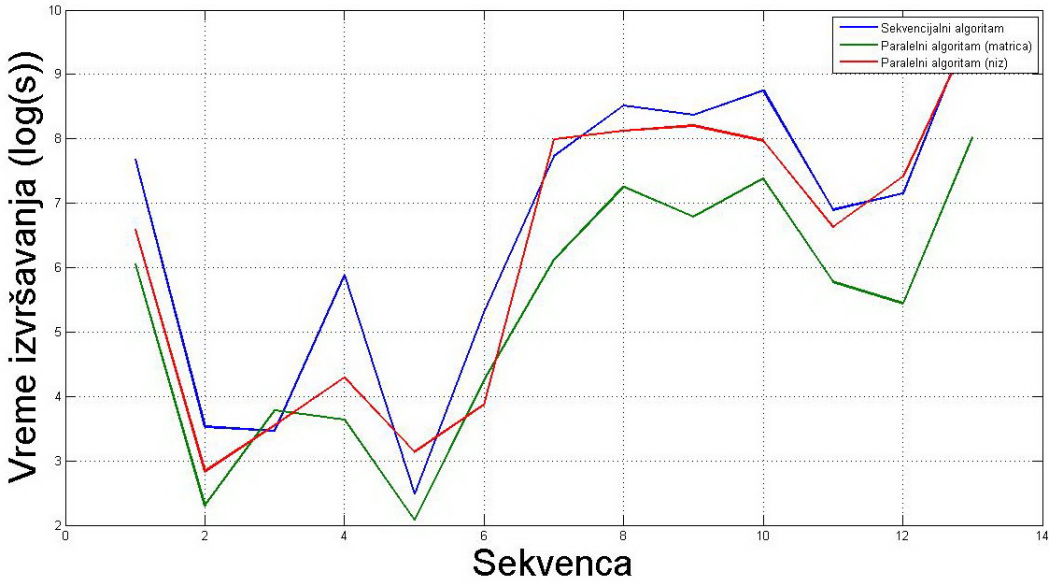
7.2 Vreme izvršavanja

U tabeli 7.4 prikazani su najbolji rezultati postignuti izvršavanjem sve tri verzije algoritma, primenjene na sve sekvence iz tabele A.1. Za sekvence za koje je algoritam konvergirao ka optimalnoj energiji prikazano je minimalno vreme izvršavanja, dok je za sekvence za koje nije postignuta konvergencija prikazana minimalna pronađena energija. Na slici 7.4 može se videti vreme potrebno za konvergenciju sekvenci od 1 do 13 iz tabele A.1, uzimajući u obzir sve tri implementacije. Zbog velike razlike u vrednostima, vreme konvergencije prikazano je na logaritatamskoj skali. Potrebno je uzeti u obzir da se zbog probabilističke prirode ovog algoritma mogu dobiti i drugačiji rezultati. Za izradu ovih rezultata uzeto je prosečno vreme pet nezavisnih izvršavanja algoritma u sekundama.

Tabela 7.4: Rezultati testiranja sekvenci iz tabele A.1. Prikazana je pronađena energija, zajedno sa procentom konvergencije, kao i vreme izvršavanja programa.

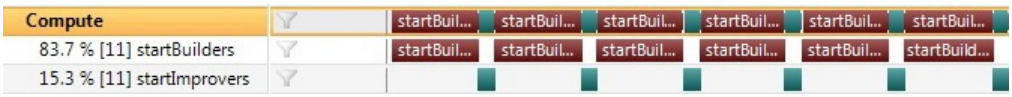
	sekvencijalna v.		paralelna v. (matrica)		paralelna v. (niz)	
	Energija	Vreme	Energija	Vreme	Energija	Vreme
1	-4 (100%)	36min 12s	-4 (100%)	7min 8s	-4 (100%)	12min 8s
2	-9 (100%)	34s	-9 (100%)	10s	-9 (100%)	17s
3	-8 (100%)	32s	-8 (100%)	44s	-8 (100%)	35s
4	-10 (100%)	6min	-10 (100%)	38s	-10 (100%)	1min 13 s
5	-9 (100%)	12s	-9 (100%)	8s	-9 (100%)	23s
6	-9 (100%)	3min 21s	-9 (100%)	1min 10s	-9 (100%)	48s
7	-8 (100%)	37min 48s	-8 (100%)	7min 36s	-8 (100%)	49min 13s
8	-10 (100%)	1h 23min	-10 (100%)	23min 33s	-10 (100%)	56min 22s
9	-11 (100%)	1h 12min	-11 (100%)	14min 46s	-11 (100%)	1h 1min
10	-9 (100%)	1h 45min	-9 (100%)	26min 51s	-9 (100%)	48min 23s
11	-10 (100%)	16min 26s	-10 (100%)	5min 24s	-10 (100%)	12min 34s
12	-7 (100%)	21min 14s	-7 (100%)	3min 52s	-7 (100%)	27min 41s
13	-14 (100%)	5h 19min	-14 (100%)	50min 41s	-14 (100%)	4h 33min
14	-20 (86.96%)	13h 26min	-23 (100%)	8h 36min	-19 (82.61%)	13h 32min
15	-18 (85.71%)	6h	-19 (90.48%)	6h	-18 (85.71%)	6h
16	-30 (83.33%)	6h	-30 (83.33%)	6h	-28 (77.78%)	6h
17	-34 (80.95%)	6h	-36 (85.71%)	6h	-33 (78.57%)	6h
18	-46 (86.79%)	6h	-48 (90.57%)	6h	-43 (81.13%)	6h
19	-42 (84%)	6h	-44 (88%)	6h	-39 (78%)	6h
20	-40 (83.33%)	6h	-43 (89.58%)	6h	-38 (79.16%)	6h

Slika 7.4: Minimalno vreme izvršavanja programa za prvih 13 sekvenci iz tabele A.1, predstavljeno na logaritamskoj skali



Kao što možemo videti sa slike 7.5, algoritam potroši najviše vremena za konstrukciju konformacije (čak 83.7%), dok se lokalna pretraga izvršava relativno brzo. Ovo je zbog toga što lokalna pretraga ne koristi vraćanje unazad, kao i zbog toga što se tačke dodaju sa dosta manje provera.

Slika 7.5: Relativno vreme izvršavanja faze konstrukcije i lokalne pretrage



U tabeli 7.5 može se videti prosečno vreme izvršavanja jedne generacije mrava, u odnosu na broj mrava. Iz tabele se može zaključiti da paralelna verzija algoritma, koja koristi matricu za predstavljanje rešetke, daje bolje rezultate od sekvencijalne verzije, što nije slučaj sa verzijom koja koristi niz. Takođe možemo primetiti da za manji broj mrava, sekvencijalna verzija daje bolje rezultate, tako da paralelnu verziju ima smisla koristiti samo za veći broj mrava. Grafički prikaz prosečnog ubrzanja paralelnih verzija u odnosu na sekvencijalnu verziju može se videti na slici 7.6.

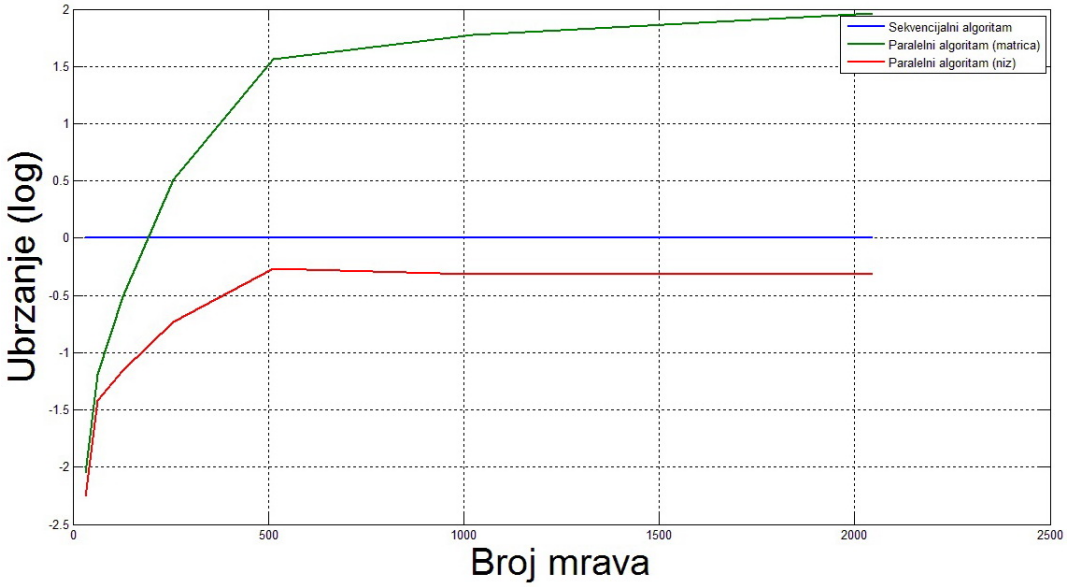
Tabela 7.5: Prosecno vreme izvrsavanja jedne generacije mrava za svaku sekvencu iz tabele A.1, u odnosu na broj mrava, izrazeno u *ms*

1.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***47***	***103***	***186***	467	1598	4865	13548
Paralelna v. (matrica)	337	342	406	***415***	***446***	***921***	***1723***
Paralelna v. (niz)	345	364	421	912	1794	3756	7698
2.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***47***	***101***	***195***	474	1565	4988	12250
Paralelna v. (matrica)	349	356	398	***408***	***459***	***956***	***1789***
Paralelna v. (niz)	321	386	458	948	1765	3854	7546
3.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***48***	***100***	***182***	490	1628	4803	15264
Paralelna v. (matrica)	346	355	404	***412***	***474***	***933***	***1766***
Paralelna v. (niz)	323	374	484	933	1837	3696	7708
4.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***54***	***101***	***234***	598	1843	6598	17165
Paralelna v. (matrica)	402	420	428	***445***	***507***	***976***	***2009***
Paralelna v. (niz)	312	415	561	945	1846	4401	8864
5.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***51***	***111***	***202***	553	1919	6229	17565
Paralelna v. (matrica)	375	430	431	***454***	***497***	***1012***	***1954***
Paralelna v. (niz)	306	409	589	996	1905	4262	8932
6.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***59***	***132***	***249***	774	3365	9952	21519
Paralelna v. (matrica)	469	477	551	***552***	***619***	***1251***	***2457***
Paralelna v. (niz)	427	544	638	1314	2523	4905	10777
7.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***64***	***137***	***258***	832	3058	10167	20265
Paralelna v. (matrica)	502	514	564	***602***	***661***	***1310***	***2607***
Paralelna v. (niz)	590	554	713	1384	2738	5408	11469
8.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***68***	***132***	***246***	848	3128	10191	20132
Paralelna v. (matrica)	493	524	566	***612***	***681***	***1298***	***2548***
Paralelna v. (niz)	599	562	743	1348	2756	5389	11534
9.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***63***	***141***	***267***	818	3009	10102	20315

Paralelna v. (matrica)	512	521	571	***598***	***643***	***1331***	***2657***
Paralelna v. (niz)	609	557	736	1421	2786	5373	11549
10.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***62***	***126***	***247***	847	3071	10193	20305
Paralelna v. (matrica)	501	523	571	***612***	***669***	***1336***	***2664***
Paralelna v. (niz)	603	564	725	1367	2755	5454	11529
11.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***64***	***142***	***263***	851	3076	10121	20312
Paralelna v. (matrica)	493	523	552	***623***	***673***	***1365***	***2626***
Paralelna v. (niz)	576	567	706	1364	2764	5448	11431
12.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***60***	***121***	***269***	812	3102	10157	20268
Paralelna v. (matrica)	482	536	572	***626***	***661***	***1321***	***2624***
Paralelna v. (niz)	601	561	738	1365	2748	5464	11497
13.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***89***	***203***	***428***	1511	6921	14127	34369
Paralelna v. (matrica)	752	756	780	***842***	***948***	***1999***	***3893***
Paralelna v. (niz)	1396	1401	1925	4103	8475	19688	58236
14.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***117***	***312***	***787***	2566	10418	16836	40697
Paralelna v. (matrica)	1001	1016	1098	***1176***	***1513***	***2051***	***5749***
Paralelna v. (niz)	1155	1421	3279	7931	17016	32660	77687
15.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***126***	***338***	***775***	2926	10390	19703	41604
Paralelna v. (matrica)	1124	1164	1275	***1444***	***1621***	***3277***	***6309***
Paralelna v. (niz)	1354	1622	3442	7545	15485	41829	86059
16.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***155***	***433***	***1132***	4549	11590	21368	48287
Paralelna v. (matrica)	1151	1351	1398	***1728***	***1853***	***3562***	***6987***
Paralelna v. (niz)	1400	1936	4026	8634	17981	37938	90164
17.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***163***	***512***	***1234***	4842	13041	29700	62065
Paralelna v. (matrica)	1489	1531	***1608***	***1829***	***2029***	***4096***	***8272***
Paralelna v. (niz)	1824	2178	4630	9741	20784	44180	98625
18.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***261***	***962***	***2415***	10069	24833	38453	77133
Paralelna v. (matrica)	2251	2390	2630	***3286***	***4359***	***8264***	***14180***

Paralelna v. (niz)	4187	3404	8409	15333	36702	78148	168291
19.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***384***	***1585***	3668	11458	31125	43839	81637
Paralelna v. (matrica)	2845	3154	***3254***	***4598***	***6785***	***11532***	***19481***
Paralelna v. (niz)	4487	4643	10239	25628	44134	95199	229014
20.							
Broj mrava	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Sekvencijalna v.	***435***	***1683***	3386	8051	19133	33870	82008
Paralelna v. (matrica)	2688	3258	***3377***	***4361***	***6718***	***11841***	***19566***
Paralelna v. (niz)	4595	4899	10375	22136	44095	101594	231459

Slika 7.6: Ubrzanje u odnosu na sekvencijalni algoritam, prikazano na logaritamskoj skali



8 Zaključak

U ovom radu posmatrana je efikasnost izvršavanja paralelne i sekvencijalne verzije algoritma *ACO*, primenjene na problem savijanja proteina. Implementirane su tri verzije algoritma: sekvencijalna verzija, paralelna verzija koja *HP* rešetku čuva u obliku matrice i paralelna verzija koja *HP* rešetku čuva u obliku niza. Sve tri verzije testirene su veći broj puta za različite parametre, kako bi se dobile što bolje informacije o prosečnom vremenu izvršavanja programa. Ove informacije ne treba uzimati kao garanciju da će se algoritam izvršiti u predviđenom roku, pošto je *ACO* metaheuristika po svojoj prirodi probabilistička aproksimativna metoda, pa se rezultati mogu značajno razlikovati od prikazanog proseka.

Hipoteza je bila da će paralelna verzija konvergirati daleko brže ka optimalnom rešenju, zbog velikog broja mrava u svakoj generaciji koji se izvršavaju paralelno. Posle opsežnog testiranja, došlo se do zaključka da iako paralelni pristup pruža određeno ubrzanje, ono je daleko manje od očekivanog. Konkretno, paralelna verzija koja *HP* rešetku čuva u obliku matrice može za isto vreme proveriti do 8 puta više konformacija u odnosu na sekvencijalnu verziju. Ovakvi rezultati su posledica arhitekture grafičkih kartica, koje nisu optimizovane za rad sa algoritmima u kojima je granajne relativno česta operacija. Šta više, algoritam *ACO* koristi vraćanje unazad u slučaju da se nađe u situaciji kada savijanje ne može biti nastavljeno. Ukoliko dođe do potrebe za vraćanjem, sve druge niti u vorpu čekaju dok se vraćanje ne izvrši, pre nego što nastave sa izvršavanjem. Ovo izuzetno smanjuje procenat iskorišćenosti grafičke kartice. Problem grananja je još primetniji kod verzije koja *HP* rešetku čuva u obliku niza, tako da se ova verzija izvršava još sporije od sekvencijalne verzije i nema je smisla koristiti.

Treba primetiti da se ubrzanje prikazano na slici 7.6 ne mora nužno preslikati na vreme konvergencije, koje se može videti na slici 7.4. Ovo se može objasniti na više načina. Prvo, algoritam *ACO* nije deterministički proces, pa ubrzanje nekih komponenti ovog algoritma neće u istoj srazmeri ubrzati ceo algoritam. Drugo, iako korišćenjem velikog broja niti možemo povećati broj proverenih konformacija po generaciji, ažuriranje traga feromona se i dalje obavlja samo jednom za svaku generaciju mrava, što može dosta uticati na brzinu kojom algoritam konvergira ka optimalnom rešenju.

Iako ovaj pristup nije postigao očekivano ubrzanje, on ipak može biti interesantan za dalja istraživanja. Mišljenje autora je da bi paralelizacija bila daleko efikasnija na višejezgarnim procesorima (eng. *Multicore processors*), umesto na mnogojezgarnim procesorima (eng. *Manycore processors*).

Rad ostavlja mogućnost za dodatno proučavanje problema, pošto varijacije sa trougaonom *HP* rešetkom i sa uopštenom hidrofobnošću nisu implementirane, kao ni verzija programa sa paralelizacijom na nivou kolonije.

Bibliografija

[1] A. Szilágyi , J. Kardos , S. Szilágyi , L. Barna , P. Závodszky *Protein Folding.*

[2] David Corne, Alan Reynolds and Eric Bonabeau *Swarm Intelligence.*

[3] El-Ghazali Talbi *Metaheuristics : from design to implementation.* ISBN 978-0-470-27858-1

[4] Alena Shmygelska, Rosalía Aguirre-Hernández, and Holger H. Hoos *An Ant Colony Optimization Algorithm for the 2D HP Protein Folding Problem.* Department of Computer Science, University of British Columbia

[5] Alena Shmygelska, Holger H. Hoos *An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem.* Department of Computer Science, University of British Columbia

[6] Pierre Delisle *Parallel Ant Colony Optimization: Algorithmic Models and Hardware Implementations.*

[7] José M. Cecilia, José M. García, Manuel Ujaldón, Andy Nisbet, Martyn Amos *Parallelization Strategies for Ant Colony Optimisation on GPUs.*

[8] Kai-Cheng Wei, Chao-Chin Wu, Chien-Ju Wu *Using CUDA GPU to Accelerate the Ant Colony Optimization Algorithm.*

[9] Natalio Krasnogor, William E. Hart, Jim Smith, David A. Pelta *Protein structure prediction with evolutionary algorithms.*

[10] Marco Dorigo, Thomas Stützle *Ant Colony Optimization.*

[11] Hu, X. and Zhang, J. and Li, Y. (2008) *Flexible protein folding by ant colony optimization.* In: Computational Intelligence in Biomedicine and Bioinformatics: Current Trends and Applications. Springer-Verlag , New York, pp. 317-336. ISBN 9783540707769

[12] Viswa Viswanathan, Anup K Sen, Soumyakanti Chakraborty *Stochastic Greedy Algorithms - A Learning-Based Approach to Combinatorial Optimization.* International Journal on Advances in Software, vol 4 no 1 & 2, year 2011, *http://www.iariajournals.org/software/*

[13] Shane Cook CUDA Programming - A Developer's Guide to Parallel Computing with GPUs. *ISBN: 978-0-12-415933-4*

[14] Cyril Zeller CUDA C/C++ Basics - Supercomputing 2011 Tutorial. *NVIDIA Corporation*

[15] Chris Wilton, Dr Martyn Amos Testing an Ant Colony Optimization Algorithm for the Two-Dimensional, Hidrophobic-Polar Protein Folding Problem. *Department of Biological Sciences, Department of Computer Science, University of Exeter*

[16] Aviezri S. Fraenkel COMPLEXITY OF PROTEIN FOLDING. *Department of Mathematics, University of Pennsylvania, Philadelphia, PA 19104-6395, U.S.A.*

[17] Scott E. Decatur Protein Folding in the Generalized Hydrophobic-Polar Model on the Triangular Lattice. *MIT Laboratory for Computer Science*

[18] Kyte J, Doolittle RF. A simple method for displaying the hydropathic character of a protein. *J Mol Biol. 1982 May 5;157(1):105-32*

[19] Wimley WC, White SH. Experimentally determined hydrophobicity scale for proteins at membrane interfaces. *Nat Struct Biol. 1996 Oct;3(10):842-8*

[20] Hessa T, Kim H, Bihlmaier K, Lundin C, Boekel J, Andersson H, Nilsson I, White SH, von Heijne G. Recognition of transmembrane helices by the endoplasmic reticulum translocon. *Nature. 2005 Jan 27;433(7024):377-81*

[21] Moon CP, Fleming KG. Side-chain hydrophobicity scale derived from transmembrane protein folding into lipid bilayers. *Proc Natl Acad Sci USA. 2011 Jun 21;108(25):10174-7*

[22] Lorenzo Carvelli, Giovanni Sebastiani Some Issues of ACO Algorithm Convergence. *Mathematics department, Sapienza Università di Roma, Istituto per le Applicazioni del Calcolo "Mauro Picone", CNR, Italy*

[23] Christian Blum Ant colony optimization: Introduction and recent trends. *ALBCOM, LSI, Universitat Politècnica de Catalunya, Jordi Girona 1-3, Campus Nord, 08034 Barcelona, Spain*

[24] NVidia CUDA C BEST PRACTICES GUIDE. *DG-05603-001_ v8.0 | January 2017*

[25] Aboul ella Hassanien, Mariofanna G. Milanova, Tomasz G. Smolinski, Ajith Abraham Computational Intelligence in Solving Bioinformatics Problems: Reviews, Perspectives, and Challenges.

[26] Trudy McKee, James R. McKee Biochemistry - THE MOLECULAR BASIS OF LIFE. *OXFORD UNIVERSITY PRESS, FIFTH EDITION*

[27] Christopher M. Dobson Principles of protein folding, misfolding and aggregation. *Department of Chemistry, University of Cambridge, Lensfield Road, Cambridge CB2 1EW, UK*

[28] Jonas Gamalielsson Models for protein structure prediction by evolutionary algorithms.

[29] M. Middendorf, F. Reischle, and H. Schmeck. Multi Colony Ant Algorithms. *Journal of Heuristics, 8(3):305–320, 2002.*

[30] Matijević Luka Nalaženje pravila pridruživanja u proteinskim sekvencama na osnovu α-heliksa. *Univerzitet u Beogradu, Matematički fakultet, 2016.*

[31] Mao Chen and Wen-Qi Huang A Branch and Bound Algorithm for the Protein Folding Problem in the HP Lattice Model. *School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China.*

[32] Mario Garza-Fabre, Eduardo Rodriguez-Tello, and Gregorio Toscano-Pulido Comparative Analysis of Different Evaluation Functions for Protein Structure Prediction Under the HP Model. *Information Technology Laboratory, CINVESTAV-Tamaulipas, Km. 5.5 Carretera Ciudad Victoria-Soto La Marina 87130, Ciudad Victoria, Tamaulipas, México*

[33] Jing Xiao, Liang-Ping Li, Xiao-Min Hu Solving Lattice Protein Folding Problems by Discrete Particle Swarm Optimization. *JOURNAL OF COMPUTERS, VOL. 9, NO. 8, AUGUST 2014*

[34] Graham A. Cox and Roy L. Johnston Analyzing energy landscapes for folding model proteins. *School of Chemistry, University of Birmingham, Edgbaston, Birmingham B15 2TT, United Kingdom*

[35] Eyal Halm Genetic Algorithm for Predicting Protein Folding in the 2D HP Mode. *Leiden Institute of Advanced Computer Science, University of Leiden, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands*

[36] Marco Dorigo Optimization, Learning and Natural Algorithms. *PhD thesis, Politecnico di Milano, Italy, 1992.*

A Dodatak

A.1 Test primeri

Ovaj skup test primera korišćen je u svim drugim naučnim radovima (delom ili u celosti) koji se bave ovom oblašću [31][32][33][15][5][11][34][35]. Većina pomenutih radova ograničava se samo na podskup kraćih sekvenci, dužine do 36 aminokiselina.

Tabela A.1: Sekvence proteina u *HP* modelu. l - dužina sekvence, E^* - minimalna energija

No.	l	E^*	Sekvenca
1	18	-4	HHPPPPPHHPPPHPPPHP
2	18	-9	PHPPHPHHHPHHPHHHHH
3	18	-8	HPHPHHHPPPHHHHPPHH
4	20	-10	HHHPPHPHPHPPHPHPHPPH
5	20	-9	HPHPPHHPHPPHPHHPPHPH
6	24	-9	HHPPHPPHPPHPPHPPHPPHPPHH
7	25	-8	PPHPPHHPPPPHHPPPPHHPPPPHH
8	25	-10	HHPHHPHPPHHPHPPHPPPHHPHPH
9	25	-11	PHHHHPHHHPPPPPHHHPHHHPHHH
10	25	-9	PHHHPPPHPPHHPHHPPPPPHPPHH
11	25	-10	HPPHPHPHHHPPPPPHPHPHPHHPH
12	25	-7	HPPPHPHPHPPHPPPHHHPPPHHPP
13	36	-14	PPPHHPPHHPPPPPHHHHHHHPPHHPPPPHHPPHPP
14	48	-23	PPHPPHHPPHHPPPPPHHHHHHHHHHPPPPPPHHPP HHPPHPPHHHHH

No.	l	E^*	Sekvenca
15	51	-21	HHPHPHPHPHHHHPHPPPHPPPHPPPPHPPPHPPPH PHPHHHHHPHPHPHH
16	60	-36	PPHHHPHHHHHHHHPPPHHHHHHHHHHHPHPPPHH HHHHHHHHHHPPPPHHHHHHPHHPHP
17	64	-42	HHHHHHHHHHHHPHPHPPHHPPHHPPHPPHHPPHH PPHPPHHPPHHPPHPHPHHHHHHHHHHHH
18	85	-53	HHHHPPPPHHHHHHHHHHHHPPPPPPHHHHHHHHH HHHPPPHHHHHHHHHHHHPPPHHHHHHHHHHHPP PHPPHHPPHHPPHPH
19	100	-50	PPPHHPPHHHHPPHHHPHHPHHPHHHHPPPPPPPP HHHHHHPPHHHHHHPPPPPPPPPHPHHPHHHHHH HHHHHPPHHHPHHPHPPHPHHHPPPPPPHHH
20	100	-48	PPPPPPHPHHPPPPPHHHPHHHHHPHHPPPPHHPP HHPHHHHHPHHHHHHHHHHPHHPHHHHHHHPPPP PPPPPPPHHHHHHHPPHPHHHPPPPPPHPHH