Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Fakulteta za matematiko in fiziko

Sandi Režonja

Vlaganje verig peptidov v ravninsko mrežo

diplomsko delo

interdisciplinarni univerzitetni
študijski program prve stopnje
računalništvo in matematika

Mentor: dr. Andrej Brodnik

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V farmaciji in biokemiji je uporabnost nano-struktur vse večja. Uporabimo jih lahko na primer za dostavo zdravila na točno določeno mesto. V literaturi je znana gradnja nano-struktur na osnovi različnih gradnikov kot so DNK in verige peptidov. Pri slednjih pride do povezovanja posameznih verig v obliki obvitih vijačnic (*coiled coil*).

Ravninsko mrežo gradimo z uporabo verig peptidov tako, da jih vlagamo v mrežo. Po zaključenem vlaganju morajo peptidi tvoriti stabilno mrežo. Poleg tega nabor peptidov ne sme tvoriti mreže na dva različna načina – nedvoumnost.

V diplomski nalogi preučite možnost izgradnje periodične planarne kvadratne mreže iz dveh štiričlenskih peptidnih verig. Preglejte obstoječe delo na tem področju, abstraktno modelirajte proces gradnje mreže in predlagajte algoritem, ki bo predlagal nabor peptidnih verig za gradnjo mreže.

Zahvaljujem se mentorju dr. Andreju Brodniku za ure diskusij, zamisli in nasvete. Zahvaljujem se tudi staršem Slavku in Valeriji Režonja za podporo skozi celotni čas študija.

Kazalo

Povzetek

Naslov: Vlaganje verig peptidov v ravninsko mrežo

Avtor: Sandi Režonja

Peptidi so zaporedja aminokislin. Če imajo strukturni motiv obvite vijačnice, lahko predvidimo njihovo povezovanje in ustvarimo večje strukture. Za sestavljanje stabilne periodične mreže uporabljamo verige peptidov, ki jih v matematičnem modelu predstavimo kot graf verižne strukture. Tega vlagamo v ravninsko mrežo, da zagotovimo željeno obliko. Relacije med členi odločajo o tem, ali se bodo ti sami sestavili v mrežo. Matrika relacij je zaradi uporabe ortogonalne množice členov permutacijska. Če upoštevamo pogoje iz narave, stabilnosti in oblike mreže, lahko možnosti še dodatno omejimo. Predlagamo omejitev območja iskanja periodične mreže ter preverjanja enoličnosti. Predstavimo algoritem z lenim vlaganjem, kjer mrežo gradimo s postopnim povezovanjem členov, ki jih vlagamo, ko jim za to preostane le ena možnost.

Ključne besede: peptidi, molekularne strukture, samosestavljanje, mreža, vlaganje.

Abstract

Title: Embedding peptide chains in a planar net

Author: Sandi Režonja

Peptides are sequences of amino acids. In case of their coiled coil structural motif we can predict what bonds will form and create larger structures. We use chains made of peptides for assembly of stable periodic nets. We represent peptides as a chain structure graph in our mathematical model, which we embed into a planar net to guarantee the proper form. Relations between links given in a relation matrix decide whether or not links will self-assemble into a net. Because of the use of orthogonal link sets, relation matrices are permutation matrices. We can further limit relation matrices by applying several conditions derived from nature, stability and net shapes. For the search of periodic nets and uniqueness check we propose area restrictions. The presented lazy embedding algorithm connects one pair of links in a step and embeds them only if there is only one embedding option left.

Keywords: peptides, molecular structures, self-assembly, mesh, embedding.

Poglavje 1

Uvod

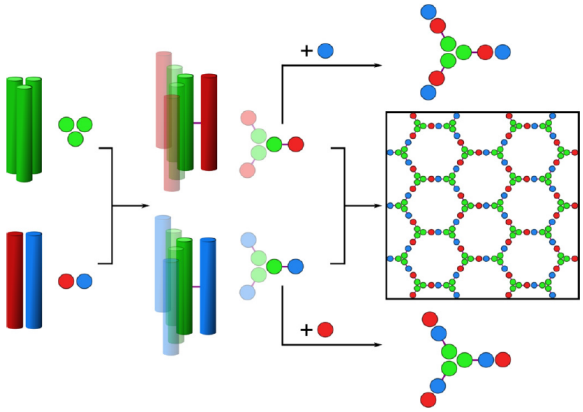
Molekule DNK se lahko sestavljajo v različne geometrijske strukture v dveh ali treh dimenzijah. Po vzoru DNK origamija se razvija tudi področje sestavljanja peptidov v geometrijske strukture. Peptidi imajo v naravi kompleksnejšo strukturo, hkrati pa jih je težje obvladovati. Njihove strukture imajo napredno uporabo v biomedicini, kemiji in znanosti o materialih [1, 2].

Leta 2013 je bila izdelana polipeptidna veriga, ki se samostojno sestavi v tetraeder. Taka veriga temelji na lastnosti peptidov, ki omogoča povezovanje z drugimi izbranimi peptidi v določeni usmerjenosti in tvorjenje togih parov. Vsak rob tetraedra sestavlja en par peptidov [1]. Raziskave na samosestavljanju poliedrov so se nadaljevale in do leta 2017 sta bili oblikovani še obliki štiristrane piramide in tristrane prizme [3].

Sestavljanje geometrijskih nanostruktur je lep primer interdisciplinarnega povezovanja kemije in biologije z računalništvom in matematiko. V [3] topologije grafov zagotavljajo obstoj konstrukcij in celo alternativne možnosti sestave. Iskanje ortogonalne množice za sestavljanje verige, v kateri vsaka molekula privlačno reagira le z eno molekulo, se lahko prevede na problem iskanja maksimalnega pokritja [4].

Zanimive oblike za sestavljanje peptidov so mreže. Z uporabo peptidov, ki se lahko vežejo na več straneh, so bile narejene samostojno sestavljive šestkotne mreže, prikazane na sliki 1.1. Pri tem so vsi peptidi postavljeni

pravokotno na ravnino mreže in na koncih niso povezani v verige, kakor pri prej omenjeni sestavi poliedrov [5].



Slika 1.1: Mreže iz peptidov z večimi vzdolžnimi povezavami. [5]

Naslednja ideja peptidnega origamija je najti samosestavljivo periodično dvodimenzionalno mrežo, ki se lahko neomejeno širi in dograjuje iz več ponavljajočih se verig, ležečih v ravnini mreže. Verige so pri tem podobne, kot so bile v [3] uporabljene za sestavo poliedrov. Zanima nas, ali taka mreža obstaja in kakšne lastnosti morajo imeti peptidi, da so v stanju najmanjše proste energije, ko so vsi vezani, v enolični in stabilni postavitvi. Splošni problem je kombinatorično zahteven, saj moramo upoštevati parametre, kot so oblike mreže (trikotne, kvadratne, šestkotne...), število različnih verig, dolžine verig, željene oblike verig v prostoru in relacije med posameznimi peptidi.

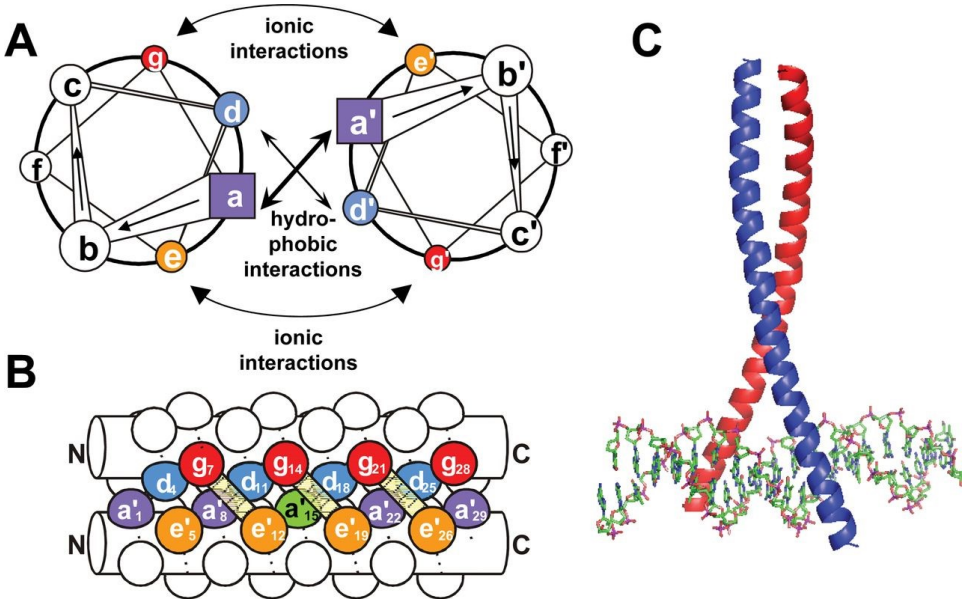
1.1 Kemijsko ozadje

Razumevanje kemije in vseh reakcij, ki potekajo med peptidi, ni bistvenega pomena za razumevanje naloge, saj je to že bilo prevedeno na preprostejši model [4, 6].

Peptidi so sestavljeni iz aminokislin, ki se vežejo zaporedno v daljše verige. Imajo več aplikacij v naravi kot DNK in so veliko bolj raznovrstni, saj obstaja

20 različnih naravnih aminokislin. Zato so posebej zanimivi kot gradniki za večmolekulske nanoarhitekture z uporabo v kemiji, biologiji, medicini in znanosti o materialih [4, 7]. Ustvarijo pa se še druge vezi (z drugimi peptidi), ki jim dajo 3D oblike.

1.1.1 Obvite vijačnice

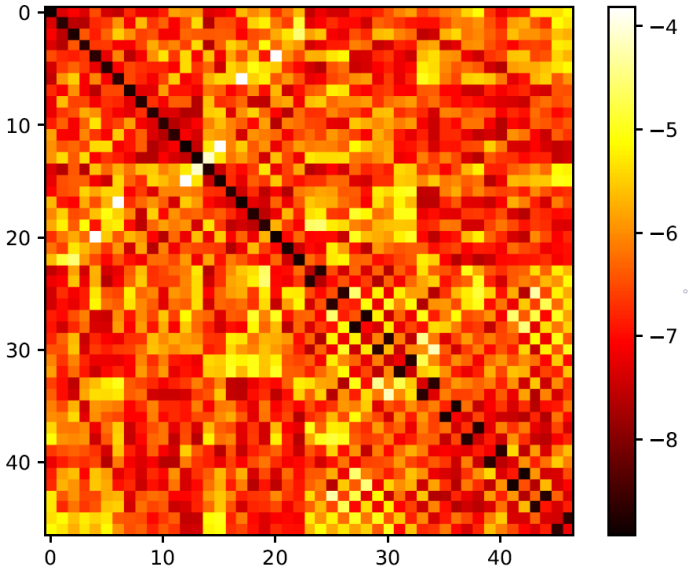


Slika 1.2: Struktura obvite vijačnice. [8]

Oblika obvitih vijačnic (*coiled coils*) je strukturni motiv, ki je raziskan bolje kot drugi motivi, in iz primarne strukture lahko sklepamo, ali se bo tvoril. Na sliki 1.2 je iz različnih perspektiv prikazana vezava peptidnih obvitih vijačnic v togi par. Aminokisline so na položajih kongruentnih po modulu 7, zato so aminokisline označene z a-g (sliki 1.2A, 1.2B). Enemu skupku sedmih aminokislin pravimo heptada, posamezen peptid pa je lahko sestavljen iz različnega števila heptad. Privlačne sile med peptidi so upoštevaje usmerjenost odvisne od sil med vsemi njihovi deli po celotni dolžini.

Iz velike množice vijačnih reakcij je bil izdelan model (funkcija vredno-

tenja) predvidevanja povezovanja v odvisnosti od interakcij med aminokislinami. Omogoča nam, da relacije med peptidi poenostavimo v „se povezuje“ ali „se ne povezuje“. S pomočjo tega lahko najdemo *ortogonalno množico* peptidov, v kateri se vsak peptid povezuje z natanko enim peptidom iz te množice v določeni usmerjenosti. Moč povezovanja med posameznimi peptidi lahko predstavimo s toplotnim diagramom (gl. sliko 1.3). Za potrebe te naloge lahko predpostavljamo, da imamo ortogonalno množico [4, 7] že pripravljeno. Mreže sestavljamo s peptidi, zasidranimi v membrani, in tako zagotovimo 2D površino za povezovanje verig.



Slika 1.3: Ortogonalna množica peptidov. Vrstice in stolpci predstavljajo vrsto peptida, notranjost pa raven privlačnosti. [4]

1.1.2 Relacijska matrika

Ker se lahko reakcije med peptidi poenostavijo v „se povezuje“ ali „se ne povezuje“, lahko na množici peptidov, ki jih bomo uporabljali, definiramo binarno relacijo. Peptida sta v relaciji R, če se člena povezujeta vzporedno. Vse relacije predstavimo z *relacijsko matriko*, ki je diskretizacija toplotnega

diagrama na sliki 1.3 na vrednosti 0 in 1. Ker smo predpostavili, da tvorijo peptidi ortogonalno množico, so vse relacijske matrike permutacijske. V nadaljevanju bomo tako množico peptidov opisali z relacisjko matriko A.

1.2 Struktura naloge

V prvem delu te naloge se ukvarjamo z matematičnim modeliranjem problema. Definicije bodo neodvisne od izbire parametrov. V preostanku naloge predstavimo dva algoritma za vlaganje. Posebno pozornost posvetimo lenemu, ki ga podrobneje opišemo in analiziramo ter predstavimo rezultate implementacije za dve verigi s štirimi členi. Delo zaključimo s povzetkom in navedemo nekatere možne posplošitve obravnavanega problema.

Poglavje 2

Matematični model

Da bomo lahko opredelili lastnosti verig, ki bodo zagotavljale stabilno in nedvoumno vložitev v ravninsko mrežo, bomo v tem poglavju definirali matematični model.

2.1 Verige

Naši gradniki so zaporedno vezani peptidi, ki jih imenujemo *verige*, posamezen pedtid je *člen* verige. Vsako verigo sestavlja k členov, čemur pravimo *dolžina* verige. Stičišče med členoma je *vozlišče*. Imamo n strukturno enakih verig $V = \{v_0, ..., v_{n-1}\}$, kjer se pri gradnji mreže vsaka lahko pojavi večkrat. Vsaka veriga je zaporedje členov, kjer j-ti člen verige v_i zapišemo kot $v_i(j)$, $v_i = [v_i(0), v_i(1), ..., v_i(k-1)]$. Konkretno pojavitev verige v ravnini označujemo z v_i^*.

Ker so členi verige usmerjeni, se glede na usmerjenost povezujejo vzporedno ali obratno vzporedno. Obratno usmerjenost člena $v_i(j)$ zapišemo $-v_i(j)$. Velja $-(-v_i(j)) = v_i(j)$. Tako sta verigi $v_i = (v_i(0), v_i(1), ..., v_i(k-1))$ in $-w_i = (-v_i(k-1), ..., -v_i(1), v_i(0))$ enaki. Člena $v_i(a)$ in $v_j(b)$ se povezujeta vzporedno, če velja $A[ik + a][jk + b] = 1$, ali obratno vzporedno, če velja $A[ik + a + nk][jk + b] = 1$, kjer je A relacijska matrika.

Ko konkretne verige opazujemo v ravnini, moramo vedeti, kdaj sta člena

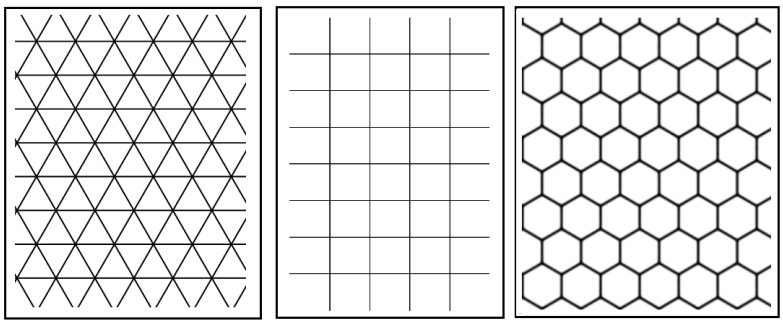
(a) (b)

Slika 2.1: Grafa verižne strukture. Členi so predstavljeni kot modre in rdeče povezave. V grafu na (b) je prikazana verižna struktura s povezanima členoma in z združenimi vozlišči.

povezana. Člen $v_i^*(a)$ je *povezan* s členom $v_j^*(b)$, če se člena $v_i(a)$ in $v_j(b)$ povezujeta in sta v ravnini neposredno skupaj v pravi usmerjenosti. Vsak konkreten člen je lahko naenkrat povezan le z enim drugim členom.

Množico verig z (delno) povezanimi členi imenujemo *verižna struktura*, ki jo lahko predstavimo kot multigraf $S(V, E)$. V sestoji iz vozlišč, v katerih združimo vozlišča verig, povezave v E pa predstavljajo člene. Primer je prikazan na sliki 2.1.

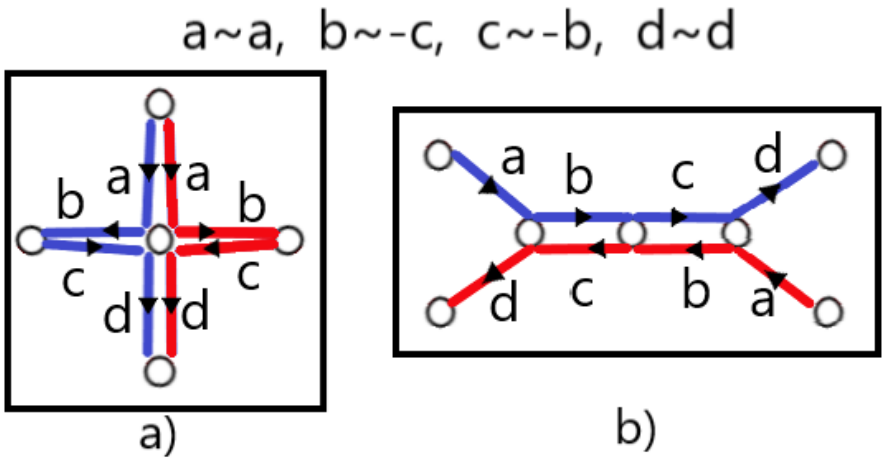
2.2 Mreža



Slika 2.2: Različne mreže.

Graf $S(V, E)$ želimo vložiti v ravninsko mrežo (prim. slika 2.2). Graf

vložimo v mrežo tako, da V vložimo v vozlišča mreže in E v njene povezave.



Slika 2.3: Primer dveh enakih verig v stanju najmanjše proste energije a). S povezavami b) pa tako stanje v dveh dimenzijah ni mogoče. Relacije med členi so prikazane na vrhu slike.

Verige v naravi težijo proti *stanju najmanjše proste energije*, kar pomeni, da se poveže čim več členov. Primer z dvema ponovitvama strukturno enake verige je prikazan na sliki 2.3. V tem stanju mora biti verižna struktura ravninska in v obliki mreže, zato mora biti vsak člen verige, ki je povezana v verižno strukturo, vložen, sicer je verižna struktura neveljavna.

2.3 Stabilnost

Ko graf S vložimo v mrežo, moramo preveriti, da je struktura trdna v vozliščih in ne more razpasti. Te zahteve definirajmo kot pogoja *stabilnosti*:

- Povezave mreže morajo biti po vlaganju v celoti dvojno pokrite. Vsako povezavo morata pokrivati natanko dva člena verižne strukture G.
- Za vsako vozlišče $t \in V$ in poljubno razbitje množice njegovih sosedov na disjunktni neprazni množici M_1 in M_2 morata v neki verigi v_i

obstajati zaporedna člena $v_i(j)$, $v_i(j+1)$, za katera velja, da je $v_i(j)$ povezava med (x,t) in $v_i(j+1)$ med (t,y), kjer $x \in M_1$ in $y \in M_2$.

Prvi pogoj stabilnosti zagotavlja, da so vsi členi na povezavah v parih, na isti povezavi pa so lahko le povezani členi. Ker morajo biti vsi členi vloženi, morajo biti vsi členi tudi povezani. Stabilna verižna struktura je tedaj zagotovo v stanju najmanjše proste energije.

Če je p število sosedov (v kvadratni mreži je $p = 4$), preverimo drugi pogoj stabilnosti v času $\Theta(p)$. V ta namen definirajmo graf $G(t)$, v katerem so vozlišča $v \in V(G(t))$ vsa sosednja vozlišča vozlišča t v mreži. Vozlišči $v_1, v_2 \in G(t)$ sta povezani, če med njima v mreži obstajata člena verige. Potem lahko pokažemo:

Izrek 2.1 *Drugi pogoj stabilnosti je izpolnjen za vozlišče t natanko tedaj, ko je $G(t)$ povezan.*

Dokaz. Na primer, da $G(t)$ ni povezan. Potem obstaja razbitje vozlišč v $G(t)$ na množici M_1 in M_2, med katerima ne obstaja povezava. To pa je v nasprotju z drugim pogojem stabilnosti.

Obratno, če ne velja drugi pogoj stabilnosti, obstaja razbitje sosedov t na množici M_1 in M_2, med katerima ne obstaja povezava. To pa pomeni, da $G(t)$ ni povezan. □

Definirajmo graf G, v katerem so vozlišča V konkretne verige v mreži, E pa so povezave med verigama v_i^* in v_j^*, če sta njuna člena povezana. Če je G povezan, pravimo, da so vse verige *vpete* v verižno strukturo.

Izrek 2.2 *Če velja drugi pogoj stabilnosti, so vse verige vpete v verižno strukturo.*

Dokaz. Iz izreka 2.1 velja za vsak t, da je $G(t)$ povezan. Iz vozlišča t pa se mora skozi sosednje vozlišče z nadaljevati vsaj ena veriga, drugače imamo dva konca na isti povezavi vozlišča z in $G(z)$ ni povezan. Iz tega očitno vidimo, da mora biti G povezan. □

Logična razlika med neveljavno in nestabilno verižno strukturo je, da neveljavna sploh ne more nastati, za nestabilno pa nočemo, da nastane.

2.4 Unikatnost in periodičnost

Unikatnost pomeni, da verige, definirane z relacijsko matriko, tvorijo le eno verižno strukturo. Pri tem moramo definirati, kdaj sta verižni strukturi S_1 in S_2 enaki ali različni. Hkrati želimo, da so verižne strukture periodične, zato lahko enakost preverimo s paralelogramom, ki je omejen s točkami 0, t_1 in t_2, kjer sta t_1 in t_2 vektorja invariantne translacije v S_1. Strukturi sta enaki, če v S_2 najdemo enako območje z upoštevanjem premika in simetrij, ki se ponavlja z enakima periodama t_1 in t_2, drugače sta različni. Po tej definiciji so aperiodične strukture vedno različne, kar ustreza našim zahtevam.

2.5 Vrste matrike relacij

Relacijska matrika omogoča izgradnjo verižne strukture, ki jo nato vlagamo v mrežo. Glede na vlaganje lahko za relacijsko matriko rečemo, da je:

- *nemogoča*: ko ne obstaja verižna struktura, ki jo lahko vložimo v mrežo.
- *nestabilna*: ko ne tvori verižne strukture, katere vloženost bi bila stabilna.
- *dvoumna*: ko tvori vsaj dve (z upoštevanjem simetrij mreže) različni verižni strukturi, ki obe lahko stabilno vložimo v mrežo.
- *ustrezna*: ko tvori natanko eno verižno strukturo, ki jo lahko stabilno vložimo v mrežo.

Poglavje 3

Tvorjenje relacijskih matrik

Model predpostavlja obstoj relacijske matrike A, ki definira vezave med členi verig. Ti so v vrsticah in stolpcih matrike relacij v zaporedju:
$v_0(0)$, ..., $v_0(k-1)$, $v_1(0)$, ..., $v_{n-1}(k-1)$, $-v_0(0)$, ..., $-v_0(k-1)$, ..., $-v_{n-1}(k-1)$.

Velikost matrike je torej $2nk \times 2nk$. Z našim algoritmom želimo najti vse ustrezne matrike. Možen pristop je tvoriti vse matrike in preveriti njihovo ustreznost. Seveda želimo v procesu tvorjenja že v naprej izločiti neustrezne, s čimer zmanjšamo iskalni prostor.

Že v poglavju 1.1.2 smo zapisali, da imamo opravka s permutacijskimi matrikami. Opazimo lahko še več zakonitosti, ki zmanjšajo število različnih matrik. Za relacijo R velja:

1. Simetričnost: $a \; R \; b \iff b \; R \; a$ in $a \not R \; b \iff b \not R \; a$
2. Nasprotnost: $a \; R \; b \iff -a \; R \; -b$ in $a \; R \; -b \iff -a \; R \; b$

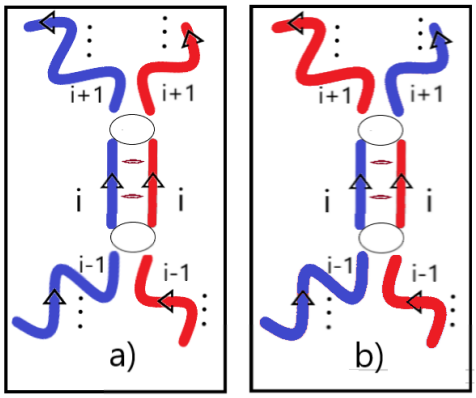
3.1 Inherentno dvoumne matrike

Za nekatere matrike lahko hitro ugotovimo, da so dvoumne:

Izrek 3.1 *Vsaka matrika, ki omogoča vzporedno povezovanje člena s sebi enakim, je nemogoča ali dvoumna.*

Dokaz.

Izrek najlažje dokažemo s primerom na sliki 3.1, kjer sta prikazani dve po lastnostih enaki verigi, katerih i-ta člena sta pararelno povezana. Velja $0 < i < k - 1$, ker če bi bila robna člena povezana ($i = 0$ ali $i = k - 1$), bi postala verižna struktura nestabilna po drugem pogoju stabilnosti. Na sliki je za prejšnje in nadaljnje člene od i nakazano poljubno nadaljevanje. V vsaki mogoči verižni strukturi matrike A imamo strukturo a), a na isto



Slika 3.1: Vzporedno povezana i-ta člena enakih verig in zamenjava nadaljavanj, ki vodi v dvoumnost.

mesto lahko pride struktura b), ki je drugačna, zato je A dvoumna. □

3.2 Vpetost

Po izreku 2.2 mora biti vsaka konkretna veriga vpeta v verižno strukturo. Lastnost lahko preverjamo že v relacijski matriki. Verige v relacijski matriki ne smejo tvoriti podmnožice B, ki se ne povezuje z njenim komplementom B^c, v katerem so vse verige iz verižne strukture, ki niso v B. Drugače B in B^c tvorita ločeni verižni strukturi, ki se ne povezujeta. Ustreznost B in B^c lahko preverimo pri manjših n, zato tako matriko izločimo. Ta pogoj lahko preverjamo s povezanostjo grafa, podobno kot v izreku 2.1.

3.3 Ekvivalenca relacijskih matrik

Matriki A in A' sta ekvivalentni, če njune verige tvorijo enake verižne strukture. Ekvivalentne matrike dobimo s premešanjem stolpcev in vrstic tako, da ohranjamo zaporedja členov. To lahko storimo na naslednja načina:

1. prešteviličimo verige, npr. $v_1 \to v'_2$ in $v_2 \to v'_1$,
2. poljubno verigo obrnemo, npr. za $k = 4$ v obrnemo v v' tako:

$$\begin{aligned} v'(0) &= -v(3), \\ v'(1) &= -v(2), \\ v'(2) &= -v(1), \\ v'(3) &= -v(0) \end{aligned}$$

V vsaki verižni strukturi, tvorjeni z verigami z lastnostmi A, vse oznake za verige v^* zamenjamo z v'^* in dobimo verižno strukturo za A'. Tako vidimo, da sta matriki res ekvivalentni.

Na prvi način dobimo permutacije verig, ki jih je $n!$, in za vsako permutacijo vsaki verigi določimo usmerjenost, za kar imamo 2^n možnosti. Za vsako matriko dobimo $n!2^n$ ekvivalentnih matrik, med katerimi so lahko nekatere enake.

3.4 Algoritem za tvorjenje relacijskih matrik

V poglavju smo predstavili nekaj zakonitosti za relacije med členi, ki zmanjšajo število možnih matrik. Sedaj pa želimo generirati množico matrik za izbrana n in k, ki ustrezajo tem zakonitostim.

V kodi 3.2 predstavimo algoritem za generacijo matrik z upoštevanjem pogojev simetričnosti, nasprotnosti, vpetosti ter zaznavanjem dvoumnosti in ekvivalence. S funkcijo *NC(i)* pridobimo vrstico ali stolpec nasprotnega člena i-tega člena v matriki. Funkcija *najdi_matrike* sprejme kot argument delno zgrajeno relacijsko matriko *relMatrika RM* in najde vse veljavne matrike, ki

```
int NC(i)
   vrni (i+n*k)%(2*n*k)

najdi_matrike(relMatrika RM)
    če ima vsaka vrstica enico
       če so vse verige vpete
           če še ni shranjena ekvivaletna matrika RM
               shrani RM
       se vrni
    najdi prvo vrstico brez enke i
       za vsak stolpec brez enke j
          če i != j
             spremembe:
                RM[i][j] = 1
                RM[j][i] = 1
                RM[NC(i)][NC(j)] = 1
                RM[NC(j)][NC(i)] = 1
             najdi_matrike(RM)
             spremenjena mesta postavimo nazaj na 0
```

Koda 3.2: Psevdokoda rekurzivnega programa za generacijo relacijskih matrik. Začetni vhod je matrika ničel velikosti $2nks$.

jih lahko dobimo iz RM z dodajanjem relacij (vstavljanjem enic v matriko) ob upoštevanju pogojev.

Algoritem temelji na rekurzivnem generiranju permutacijskih matrik. Simetričnost, nasprotnost in dvoumnost matrik preverjamo med samo generacijo, vpetost in ekvivalenco pa pred shranjevanjem matrike. Za vpetost preverjamo povezanost grafa, v katerem vozlišča V predstavljajo verige v matriki, povezave pa relacijo med členi verig. Pri preverjanju ekvivalence ustvarimo vseh $n!2^n$ ekvivalentnih matrik in jih primerjamo s prej shranjenimi matrikami. Tako preverjanje ekvivalence podaljša čas izvajanja, ki je sedaj enak številu primerjanj v listih drevesa rekurzije, ko zapolnimo vse vrstice in stolpce v matriki. Če označimo število matrik brez upoštevanja ekvivalentnosti m_b in z upoštevanjem ekvivalentnosti m_e, je časovna zahtevnost programa enaka $\Theta(n!2^n m_b m_e (nk)^2)$. Vsako izmed $n!2^n m_b$ možnosti namreč primerjamo z $\Theta(m_e)$ matrikami, za kar porabimo $(2nk)^2$ operacij. V prihodnje velja poiskati metode, ki na višjem nivoju rekurzije zaznajo, da vse naslednje matrike niso vpete ali so ekvivalentne že generiranim.

3.5 Število matrik

Celotno število permutacijskih matrik je $(2nk)!$, saj lahko v prvo vrstico vstavimo enko na $2nk$ mest, v drugo na $2nk - 1$, in tako naprej do $2nk$-te vrstice. Podobno bomo poskušali prešteti število matrik, ki jih lahko generiramo s pogoji iz tega poglavja.

Najprej preštejmo število matrik ob upoštevanju simetričnosti, naprotnosti in zaznavanju dvoumnih matrik. Pogoji v matriki pomenijo:

- Simetričnost: $A_{ij} \iff A_{ji}$
- Nasprotnost: $A_{ij} \iff A_{NC(j)NC(i)}$
- Zaznavanje dvoumnosti: $A_{ii} = 0$

Simetričnost hkrati pomeni, da so enice v stolpcih, ki imajo enake oznake kot vrstice. Stolpci (ali vrstice) brez enice torej tvorijo ustrezno matriko

manjših razsežnosti. Opazimo, da ob dodajanju relacije v matriko dodamo štiri enice zaradi simetričnosti in nasprotnosti. Izjema je dodajanje enice na mesto $A_{iNC(i)}$, ko dodamo samo dve. Za velikost matrike x lahko v prvi vrstici enico postavimo na $x-2$ mest tako, da dobimo manjšo matriko velikosti $(x-4)\times(x-4)$, in eno mesto, da dobimo matriko velikosti $(x-2)\times(x-2)$. Imamo torej rekurzivno enačbo za velikost matrike x (v našem primeru $2nk$), pri čemer opozarjamo, da je x zaradi definicije vedno sodo število:

$$\begin{aligned} F(x) &= F(x-2) + (x-2)F(x-4) \\ F(2) &= 1, \quad F(0) = 1 \end{aligned} \tag{3.1}$$

Prazna matrika je lahko prazna le na en način, matriko velikosti 2 pa lahko zapolnimo le na en način, ker enic ne smemo postaviti na diagonalo. Iz tega sledijo robni pogoji.

Preverimo, da matriko velikosti 4 lahko zapolnimo na tri načine. Res lahko dvakrat dodamo 4 enice in enkrat dve, kar pa enolično določi položaj ostalih dveh. Rekurzivno izračunamo, da je število relacijskih matrik s temi pogoji za $n=1$, $k=4$ enako 25, za $n=2$, $k=4$ pa 5937.

Ekvivalenco matrik zaradi zapletenosti še ne znamo vključiti v rekurzivno formulo. Z vključitvijo bi dobili novo funkcijo F, ki jo lahko uporabimo pri izračunu števila ob upoštevanju vpetosti.

Tudi pogoj vpetosti je prezahteven, da bi izpeljevali število matrik z vpetimi n k-členskimi verigami $G(n,k)$ za splošna parametra n in k. Lahko pa povemo, da za $n=1$ pogoj vpetosti ni definiran, za $n=2$ pa lahko imamo dve stanji - ali sta verigi povezani ali ne. Če nista, je, kot da imamo dve neodvisni matriki z $n=1$. $G(n,k)$ je število matrik s povezanima verigama, torej velja:

$$G(2,k) = F(4k) - F^2(2k) \tag{3.2}$$

Brez upoštevanja ekvivalence izračunamo, da je $G(2,4)=5312$. Program 3.2, ki upošteva pogoj ekvivalence, pa izdela 19 matrik za $n=1$, $k=4$ in 767 matrik za $n=2$, $k=4$.

Poglavje 4

Reševanje problema

Do sedaj smo definicije in izpeljave zastavili splošno, saj z določitvijo parametrov ne bi nič pridobili. Pri praktičnem reševanju problema se omejimo na kvadratne mreže, pri $n = 2$ ter $k = 4$.

4.1 Naivni pristop

Vsaka ustrezna matrika bo omogočila popolnoma vloženo periodično verižno strukturo, v kateri bo vsaka konkretna veriga imela neko obliko (postavitev v prostoru). Naivni pristop je, da imajo vse ponovitve enake verige tudi enako obliko, kar v splošnem ni res. Generiramo vse možne oblike, ki jih lahko verige zavzamejo, in vse možne linearne transformacije. Nato preverimo, ali se oblike dajo vložiti v mrežo tako, da jo zapolnijo. Mreža mora biti dovolj velika, da se izrazi osnovni vzorec. Iz njihove strukture potem tvorimo matriko. Ker imajo lahko strukturno enake verige v mreži več oblik, ne najdemo vseh rešitev. Tudi dvoumnosti in stabilnosti ne znamo preveriti, saj lahko verige v drugačnih oblikah tvorijo aperiodične ali nestabilne mreže, kar pomeni, da mogoče najdemo napačne rešitve.

4.2 Leno vlaganje

Pri lenem vlaganju bomo mrežo gradili postopoma. Težava takih pristopov je, da ne vemo, ali se bodo algoritmi končali in vrnili rešitve. Da zagotovimo zaustavitev, predlagamo omejitev območja, na katerem iščemo periodično vložitev. V območju $Q(r)$ so vsa vozlišča mreže, ki so na manhatenski razdalji od središčnega vozlišča manjši od r, in povezave teh vozlišč. S tem izgubimo nekaj periodičnih vlaganj z večjimi periodami, če ta obstajajo. Ker bomo v praksi sestavljali mreže z dvema verigama, prevelike periode niso uporabne.

Za periodične postavitve na območju $Q(r)$ lahko povemo, ali se bodo nadaljevale v neskončnost. Pokažemo pa lahko, da na robnih vozliščih vedno lahko najdemo drugačno vložitev, ki se morda ne nadaljuje v neskončnost. Velja:

Izrek 4.1 *Če za matriko A uspešno zgradimo in vložimo verižno strukturo v kvadratno mrežo do območja $Q(r)$, bo na istem območju obstajala vsaj še ena drugačna verižna struktura.*

Dokaz. Za matriko A imamo zgrajeno in vloženo verižno strukturo na območju $Q(r)$. Imamo opravka s kvadratno mrežo, zato $Q(r)$ vedno vključuje vozlišče t z dvema robnima povezavama, ki imata eno vozlišče izven območja. Opazujemo nadaljevanja verig na teh povezavah izven območja. Na primer, da se nadaljevanje ene od verig vrne v mrežo in tako dovoljuje samo eno možnost vlaganja za člena na povezavi. Del te verige, ki se vrne mrežo, lahko sprostimo iz mreže in na njegovo mesto vstavimo del nove enake verige. To lahko storimo za obe robni povezavi vozlišča t. Sedaj lahko zamenjamo člene med povezavama in tako dobimo drugačno verižno strukturo na območju $Q(r)$. Za A bomo zaznali dvoumnost.

□

Iz izreka 4.1 vidimo, da moramo zapolniti še širše območje $Q(r + e)$. Če na območju $Q(r)$ takrat obstajata dve različni vložitvi, je matrika dvoumna. Razširitev območja je racionalna posplošitev preverjanja obstoja

verižne strukture v neskončnost, saj se z neskončnostjo v praksi ne ukvarjamo. Označimo še $R = r + e$.

Drugačen je pristop, ki smo ga poimenovali leno vlaganje. Mrežo gradimo na območju $Q(R)$ s povezovanjem členov, vlagamo pa le, ko preostane členu le ena možnost za vložitev. Tak algoritem smo izbrali za izdelavo in podrobnejšo predstavitev.

Leno vlaganje gradi verižne strukture. Pri tem izkorišča dva principa. Prvi je, da mora biti v končni verižni strukturi vsak člen povezan s členom iz trenutne delno zgrajene verižne strukture ali z novim členom, ki ga dodamo. Drugi pa je, da se lahko neskončni graf verižne strukture ustrezne relacijske matrike le na en način vloži v mrežo, če upoštevamo simetrije mreže, zato člene verižne strukture vlagamo le, ko jim ostane le ena možnost vložitve.

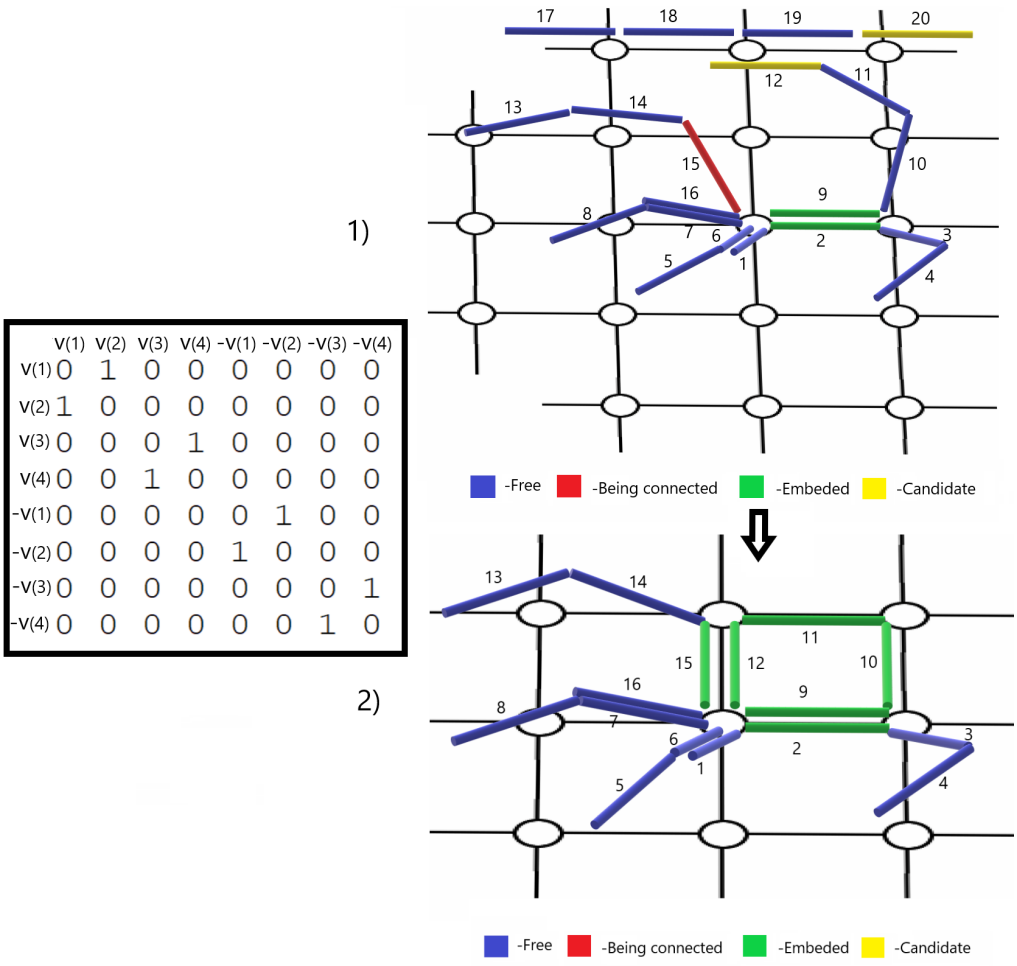
Algoritem je predstavljen v kodi 4.1. V psevdokodi je *RelMatrika* relacijska matrika iz poglavja 1.1.2. *VerižStrukt* vključuje delno vložen graf verižne strukture in mrežo, ki smo ju definirali v poglavju 2. V *RešitStrukt* shranimo podatke za preverjanje enakosti dveh vložitev.

Funckija *evalRelMatrix* prejme tri parametre: *RelMatrika*, *VerižStrukt* in *RešitStrukt*. Funkcija glede na parametre označi relacijsko matriko kot NEMOGOČA, DVOUMNA, NESTABILNA ali USTREZNA.

evalRelMatrix najprej v verižni strukturi poišče prost člen *povezovanČlen*, ki ga je potrebno še povezati. Če takšnega člena ni, se rekurzija zaključi in opredeli vrsta matrike (prvi `if` stavek v psevdokodi). Pri tem uporabimo logiko, predstavljeno v razdelku 2.5. Če pa takšen člen obstaja, v *VS* poiščemo vse člene, ki so povezljivi s *povezovanČlen* in jih shrani v *kandidati*. Opozarjamo, da so to lahko vsi prosti členi v *VS* in povezljiv člen v novi verigi, ki jo dodamo. Funkcija v preostanku rekurzivno preišče možnost vlaganja. Pri vrnitvi iz rekurzivnega sestopa ob najdeni dvoumnosti ali nestabilni strukturi zaključimo, ob ustreznosti se vrnemo na območje $Q(r)$, kjer nadaljujemo z iskanjem, in v primeru nemogoče strukture le nadaljujemo.

```
evalRelMatriko(RelMatrika RM, VerižStrukt VS, RešitStrukt RS)
  povezovanČlen = VS.najdiČlenZaPovezati(RM);

  if (povezovanČlen ne obstaja) // ustavitev rekurzije
    if (!VS.preveriStabilnost()) return NESTABILNA;
    if (RS ne obstaja)
      RS = new RS(VS);
      return USTREZNA;
    if (RS.preveriEnakost(VS)) return USTREZNA;
    return DVOUMNA;

  kandidati = VS.najdiKandidate(povezovanČlen, RM);
  kandidati +=
    nasprotni člen iz VS.novaNasprotnaVeriga(členZaPovezat, RM)
  najdenaRešitev = false;
  vrnjeno = NEMOGOČA;
  for each kandidat in kandidati
    if (VS.poveživloži(povezovanČlen, kandidat, RM))
      vrnjeno = evalRelMatriko(RM, VS, RS);
      VS->revertPoveživloži(relMatrix);

    if(vrnjeno == USTREZNA)
      if (VS.razdaljaDoCentra(povezovanČlen) >= r)
        return USTREZNA;
      najdenaRešitev = true;
    else if (vrnjeno == DVOUMNA || vrnjeno == NESTABILNA)
      return vrnjeno;

  return najdenaRešitev ? USTREZNA : NEMOGOČA;
```

Koda 4.1: Psevdokoda algoritma, ki razvrsti relacijsko matriko. Začetni vhod je relacijska matrika, *VerižStrukt* z eno verigo, ki ima en člen vložen v središče mreže, in prazna *RešitStrukt*.



	V(1)	V(2)	V(3)	V(4)	-V(1)	-V(2)	-V(3)	-V(4)
V(1)	0	1	0	0	0	0	0	0
V(2)	1	0	0	0	0	0	0	0
V(3)	0	0	0	1	0	0	0	0
V(4)	0	0	1	0	0	0	0	0
-V(1)	0	0	0	0	0	1	0	0
-V(2)	0	0	0	0	1	0	0	0
-V(3)	0	0	0	0	0	0	0	1
-V(4)	0	0	0	0	0	0	1	0

Slika 4.2: Prikaz delovanja programa v četrtem koraku rekurzije za matriko, ki je na sliki.

Slika 4.2 vizualizira korak rekurzije med izvajanjem programa na primeru v četrtem koraku rekurzije za matriko na sliki. Program dobi za vhod strukturo 1), na kateri so zeleni členi že vloženi. Rdeč člen je najbližje središču, zato ga določimo za *povezovanČlen*. Rumeni členi so kandidati za vezavo, vključno z novo verigo. To niso vsi četrti členi verig, saj sta člena 4 in 8 na napačni razdalji in v primeru vezave ne bi tvorila kvadrata. Na 2) je struktura, ki nastane, če za vezavo izberemo člen 12. Paru (12, 15) sedaj ostaneta le dve možnosti za vložitev. Ker se to zgodi prvič, ju vložimo v zgornjo povezavo, zatem pa še ostale člene, ki jim sedaj preostane le ena možnost za vložitev. Funkcijo kličemo ponovno na strukturi 2).

V preostanku podrobneje opisujemo kompleksnejše funkcije, ki nastopajo v psevdokodi in njihove časovne zahtevnosti:

- *VS.najdiČlenZaPovezati(RM)*: Vrne nepovezan člen iz VS, ki je najbližje središču mreže in ni na razdalji večji od R, ali pa vrne, da tak člen ne obstaja. $O(R^2)$.

- *VS.preveriStabilnost()*: vrne true, če delno vložena verižna struktura na $Q(r)$ ustreza pogojema stabilnosti iz te naloge in false drugače. $\Theta(R^2)$.

- *RS(VS)*: inicializacija RS. Parameter je VerižStrukt, od katerega shranimo mrežo z vloženimi členi, da lahko po potrebi preverimo enakost z drugo najdeno mrežo. $\Theta(R^2)$.

- *RS.preveriEnakost(VS)*: vrne `true`, če sta RS in VS enaki verižni strukturi, ter `false` sicer. Enakost preveri tako, da za nek premik najde ponovljeno središče iz RS v VS in preveri ali je tudi okolica na $Q(r)$ enaka. Če se prekrivajoča se območja v celoti ujemajo v vsaj eni od osmih simetrij mreže, je enakost najdena. $O(R^4)$.

- *VS.najdiKandidate(povezovanČlen, RM)*: vrne vse člene iz verižne strukture s katerimi se *povezovanČlen* povezuje glede na RM. Osnova je sprehod skozi vse člene v verižni strukturi in preverjanje povezljivosti v relacijski matriki. Funkcijo lahko dodatno izboljšamo, da ne vrne

členov, ki so na napačni razdalji in bi tvorili neveljavno obliko ali vozlišče s stopnjo večjo kot štiri. $\Theta(R^2)$.

- *VS.novaNasprotnaVeriga(povezovanČlen, RM)*: v verižno strukturo doda novo verigo po definiciji verige iz naloge ter vrne člen, ki se glede na relacijsko matriko povezuje s *povezovanČlen*. $\Theta(k)$.

- *VS.poveziVloži(povezovanČlen, kandidat, RM)*: Če je možno, poveže oba člena v verižni strukturi, sicer vrne false. Preveri še, ali je za novo verižno strukturo možna vložitev tako, da vlaga nevložene člene, ki imajo sosednji člen vložen, potem pa za njihove naslednike stori isto. Če ima tak člen samo eno možno vložitev, ga trajno vloži in morebiti poveže z drugim členom na istem mestu v mreži. Če člen nima vložitve, funkcija vrne false. Ko vse člene uspešno preveri, vrne true. Ocena časovne zahtevnosti je $O(4^k R^2)$, saj moramo za največ R^2 členov rekurzivno do globine $k-1$ preveriti obstoj vložitve na največ 4 mesta na vsaki stopnji rekurzije. $O(4^k R^2)$

4.2.1 Analiza

Največja globina rekurzije je enaka polovici števila členov v mreži, torej $4R^2$. Število kandidatov v vsakem vozlišču rekurzivnega drevesa je vsekakor manjše od števila členov nasprotne vrste, ki jih je največ $\frac{4R^2}{nk}$. Za $c \geq 2$ je $\sum_{i=0}^{n} c^i < c^{n+1}$, zato število vozlišč rekurzije ocenimo s številom listov rekurzivnega drevesa $O((\frac{4R^2}{nk})^{4R^2})$. V vsakem vozlišču neodvisno od globine opravimo največ $O(4^k R^2)$ operacij, saj je *poveziVloži* najzahtevnejša funkcija. Časovno zahtevnost programa lahko omejimo z $O(4^k R^2 (\frac{4R^2}{nk})^{4R^2})$.

Vedno imamo shranjeno verižno strukturo in morebiti še rešitveno strukturo, ki je manjša od verižne strukture. Graf verižne strukture obsega $\Theta(R^2)$ vozlišč in prav toliko povezav. Mreža je velikosti $4(R+k)^2$, matrika pa $(2nk)^2$. V vsakem vozlišču rekurzije shranimo kandidate za povezovanje, katerih število smo določili prej, pa tudi globino rekurzije. Prostorska zahtevnost za shranjevanje kandidatov je $O((\frac{R^4}{nk}))$. Za $R > n, k$ je to tudi prostorska zahtevnost celotnega programa.

4.3 Rezultati

Program je časovno zahteven, zato uporabimo območje z $r = 3$ in $e = 1$. To je dovolj veliko, da so rezultati relevantni. Povzetek uvrščanja matrik z implementacijo algoritma iz prejšnjega poglavja je za $n = 2$ v tabeli 4.1. Za $n = 1$ smo uporabili spremenjen algoritem, ki ob najdeni stabilni verižni strukturi odstrani povezave do območja $Q(r-1)$, medtem ko se program po opisu iz 4.2 v tem primeru vrne le na $Q(r)$. Spremenjen algoritem ne preišče vseh struktur, zato omogoča nepravilno pozitivne rezultate, a deluje hitreje. Rezultati so prikazani v tabeli 4.2.

Matrike smo generirali z algoritmom 3.2. Izmed 19 matrik pri $n = 1$ je 10 nestabilnih in 9 dvoumnih. Pri $n = 2$ je izmed 767 matrik 223 nestabilnih, 78 dvoumnih in 466 nemogočih. Ustreznih matrik nismo našli. Obstajajo lahko pri večjih parametrih r, e, n in k, a bi preverjanje trajalo predolgo, saj program preveri vse možnosti za povezavanje, da dokaže ustreznost. Opazimo,

Type	N	$\overline{t}[ms]$	$t_{max}[ms]$	$t_{min}[ms]$	σ
VSE	767	22000	$3,87 \times 10^7$	4	2.3×10^6
NESTABILNE	223	370	22000	17	1694
DVOUMNE	78	$2,14 \times 10^6$	$3,87 \times 10^7$	932	6.9×10^6
NEMOGOČE	466	6	20	4	2.2
USTREZNE	0	-	-	-	-

Tabela 4.1: Rezultati za $n = 2$.

Type	N	$\overline{t}[ms]$	$t_{max}[ms]$	$t_{min}[ms]$	σ
VSE	19	3366.58	32000	17	7700
NESTABILNE	10	72	170	17	50
DVOUMNE	9	7000	32000	1800	10000
NEMOGOČE	0	-	-	-	-
USTREZNE	0	-	-	-	-

Tabela 4.2: Rezultati za $n = 1$.

da za $n = 1$ tudi nemogočih matrik ni, kar je verjetno zaradi večje gostote možnih povezav.

Če glede na razvrstitev matrik seštejemo čase izvajanja programa in vsoto delimo s številom matrik pripadajoče vrste, dobimo povprečni čas izvajanja $\overline{t}$ za vsako vrsto. Zanimive so razlike v časih izvajanja glede na razvrstitev matrik. Pri $n = 2$ nemogoče matrike najdemo zelo hitro. Za nestabilne matrike porabimo v povprečju več časa, saj preiskujemo do zapolnitve območja, ki vključuje nestabilno vozlišče. Za dvoumne matrike porabimo največ časa, saj po prvi najdeni mogoči verižni strukturi preiskujemo naprej, dokler ne najdemo verižne strukture, ki se dovolj razlikuje od prve, da zagotovo ni njen periodični premik ali simetrija. Največji standardni odklon σ imajo dvoumne matrike, kar pomeni, da so časi izvajanja za take matrike najbolj nepredvidljivi. Maksimalni čas izvajanja t_{max} za dvoumne matrike je dobrih

deset ur. Skupaj s predvideno eksponentno časovno zahtevnostjo programa nam to pove, da bo program za nekatere matrike delal zelo dolgo, če območje dodatno povečamo.

Velja preučiti možnost, da ustrezne matrike sploh ne obstajajo. Dokazi za dvoumnost in nestabilnost so enostavni v obliki verižnih struktur in jih lahko hitro preverimo.

Programa za generiranje in evalvacijo matrik, generirane matrike, rezultati ter verižne strukture, ki dokazujejo razvrstitev matrik, so javno dostopni na `https://github.com/ssandir/PeptideEmbedding`.

Poglavje 5

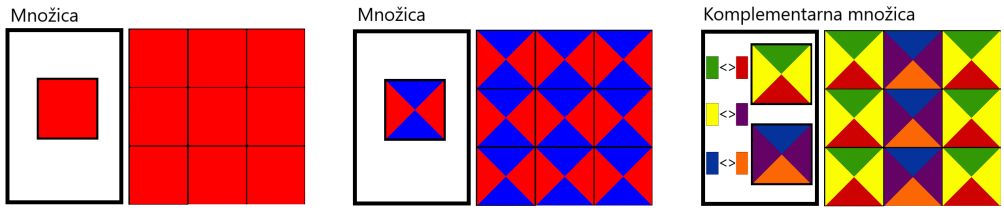
Zaključek in delo za naprej

V nalogi smo obravnavali problem vlaganja verig peptidov v mrežo. Problem smo najprej matematično modelirali. Najprej smo definirali peptide kot člene verig, potem verižno strukturo, sestavljeno iz povezanih verig, in na koncu mrežo, v katero vlagamo verižno strukturo. Relacije med usmerjenimi členi smo predstavili z relacijsko matriko. Da bi zmanjšali iskalni prostor matrik, smo z upoštevanjem biokemijskih lastnosti izločili nekatere matrike. Definirali smo vrste matrik glede na vlaganje verižnih struktur v mrežo. Na podlagi zgoraj opisanega smo predstavili osnovni algoritem za generiranje matrik.

V nadaljevanju smo se omejili na kvadratne mreže in dve 4-členski verigi. Predstavili smo dva pristopa k vlaganju verižnih struktur v mrežo. Pri prvem definiramo obliki obeh verig ter poskušamo z njima periodično ploščičiti celotno ravnino. Izkaže se, da težko preverimo pravilnost in enoličnost vloženosti. Temu ni tako pri drugem pristopu z lenim vlaganjem, ki deluje z omejenim območjem, in lahko relacijske matrike napačno razvrstimo le kot neustrezne. Bolje smo predstavili leni algoritem in ga implementirali. Omejili smo območje iskanja in na območju s polmerom 3 ustreznih matrik nismo našli.

Predstavljena naloga je lahko temelj za nadaljnje raziskave ustvarjanja ravninskih mrež iz verig peptidov. Večina predstavljenih definicij, izpeljav in idej je lahko posplošenih na poljuben n in k. Najprej lahko bolje raziščemo učinkovitejše generiranje matrik. Ker nismo uspeli najti ustreznih matrik, se

je pri reševanju problema v bodoče smiselno poglobiti v možnost, da ustrezne matrike sploh ne obstajajo. Potem lahko še dodatno optimiziramo implementacijo lenega algoritma ter ga priredimo za iskanje dokazov neustreznosti.



Slika 5.1: Trivialna periodična ploščičenja z Wang ploščicami. Desna množica je prikaz komplementarnega ploščičenja. Pravila za vezavo robov so v tem primeru definirana ob ploščicah.

Kot morebitno zanimivo iztočnico za bodoče raziskovanje omenimo verige s povezanimi konci, ki tvorijo obroče (v primeru $k = 4$ kvadrate). Take verige lahko predstavljajo kar Wang ploščice [9] in za njih obstajajo trivialna ploščičenja. Nekaj takih ploščičenj je prikazanih na sliki 5.1. Aktualna je še uporaba deterministčnih ploščičenj po vzoru raziskav z DNK [10], s katerimi lahko tudi računamo.

Literatura

[1] Gradišar H., Božič S., Doles T., Vengust D., Hafner-Bratkovič I., Mertelj A., Webb B., Šali A., Klavžar S., and Jerala R. Design of a single-chain polypeptide tetrahedron assembled from coiled-coil segments. *Nature Chemical Biology*, 9(6):362–366, 2013.

[2] Greg L. Hura and John A. Tainer. Coiled coils unspring protein origami. *Nature Biotechnology*, 35:1044–1045, 2017.

[3] Ajasja Ljubetič, Fabio Lapenta, Helena Gradišar, Igor Drobnak, Jana Aupič, Žiga Strmšek, Duško Lainšček, Iva Hafner-Bratkovič, Andreja Majerle, Nuša Krivec, Mojca Benčina, Tomaž Pisanski, Tanja Ćirković Veličković, Adam Round, José María Carazo, Roberto Melero, and Roman Jerala. Design of a single-chain polypeptide tetrahedron assembled from coiled-coil segments. *Nature Biotechnology*, 35:1094–1101, 2017.

[4] Daniel Silađi. Računalniške metode za načrtovanje polipeptidnega origamija. Zaključna naloga, Fakulteta za matematiko, naravoslovje in informacijske tehnologije, Univerza na Primorskem, 2017.

[5] Jordan M. Fletcher, Robert L. Harniman, Frederick R. H. Barnes, Aimee L. Boyle, Andrew Collins, Judith Mantell, Thomas H. Sharp, Massimo Antognozzi, Paula J. Booth, Noah Linden, Mervyn J. Miles, Richard B. Sessions, Paul Verkade, and Derek N. Woolfson. Self-assembling cages from coiled-coil peptide modules. *Science*, 340(6132):595–599, 2013.

[6] Vladimir Potapov, Jenifer B. Kaplan, and Amy E. Keating. Data-driven prediction and design of bzip coiled-coil interactions. Article, PLoS Comput Biol, 2015.

[7] Silvia Cavalli, Fernando Albericio, and Alexander Kros. Amphiphilic peptides and their cross-disciplinary role as building blocks for nanoscience. *Chemical Society Reviews*, 39:241–263, 2010.

[8] Jody M. Mason, Kristian M. Müller, and Katja M. Arndt. ipep: peptides designed and selected for interfering with protein interaction and function. In *2nd international meeting on molecular perspectives on protein–protein interactions*, pages 1442–1447. Biochemical Society Transactions, 2008.

[9] Cyril Allauzen and Bruno Durand. Tiling problems. In *The classical decision problem, Appendix A*, pages 407–420. Springer, 1997.

[10] Xuncai Zhang, Yanfeng Wang, Zhihua Chen, Jin Xu, and Guangzhao Cui. Arithmetic computation using self-assembly of DNA tiles: subtraction and division. *Progress in Natural Science*, 19(3):377 – 388, 2009.