UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tomo Česnik

# Izgradnja podatkovne baze strukturnih podobnosti proteinov

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Blaž Zupan
Somentor: doc. dr. Janez Konc

Ljubljana, 2011

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01743/2011
Datum: 15.03.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **TOMO ČESNIK**

Naslov: **IZGRADNJA PODATKOVNE BAZE STRUKTURNIH PODOBNOSTI PROTEINOV**

**PROTEIN STRUCTURAL SIMILARITY DATABASE: THE ARCHITECTURE AND IMPLEMENTATION**

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Računsko ocenjevanje podobnosti proteinov je lahko časovno zelo zamudno. Še posebej, če želimo izbrani protein primerjati z desetimi tisočimi ostalimi znanimi proteini. V diplomski nalogi predlagajte arhitekturo podatkovne zbirke, ki bi shranjevala vnaprej izračunane podobnosti med proteini. Podatkovno bazo implementirajte, v njo shranite izračune podobnosti za pare več kot 70.000 znanih proteinov iz repozitorija Protein Data Bank in implementirajte spletni dostop do baze. Za izračune podobnosti uporabite algoritem ProBiS.

Mentor:

prof. dr. Blaž Zupan

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Somentor:

doc. dr. Janez Konc

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

# IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Tomo Česnik,

z vpisno številko 63050020,

sem avtor diplomskega dela z naslovom:

Izgradnja podatkovne baze strukturnih podobnosti proteinov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  prof. dr. Blaža Zupana
  in somentorstvom
  doc. dr. Janeza Konca
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 24.09.2011 Podpis avtorja:

# Zahvala

Zahvaljujem se mentorjema, prof. dr. Blažu Zupanu in doc. dr. Janezu Koncu, za vse uporabne informacije in pomoč pri izvedbi.
Zahvaljujem se prof. dr. Dušanki Janežič za koristne in zanimive diskusije.
Zahvaljujem se Ani Bajc za popravljanje sloga pisanja in za motivacijo.
Posebna zahvala gre tudi moji družini in vsem ostalim, ki so me kakorkoli podpirali in spodbujali pri študiju.

# Kazalo

# Seznam uporabljenih kratic in simbolov

**3D** - Three Dimensional
**AJAX** - Asynchronous JavaScript and XML
**CSS** - Cascading Style Sheets
**DB** - DataBase
**DNS** - Domain Name System
**FTP** - File Transfer Protocol
**HTML** - Hypertext Markup Language
**HTTP** - HyperText Transfer Protocol
**JSON** - JavaScript Object Notation
**JVM** - Java Virtual Machine
**KI** - Kemijski Inštitut
**PDB** - Protein Data Bank
**PHP** - Hypertext Preprocessor
**ProBiS** - Protein Binding Sites
**SQL** - Structured Query Language
**SSH** - Secure Shell
**SVN** - Subversion (Repository)
**XML** - Extensible Markup Language

# Povzetek

Diplomsko delo temelji na algoritmu ProBiS, katerega namen je računanje strukturnih podobnosti proteinov. Taki postopki so v splošnem dolgotrajni, zato je bil glavni cilj dela vnaprejšnje računanje vseh primerjav proteinov in shranjevanje rezultatov v podatkovno bazo. Diplomska naloga se začne z opisom algoritma ProBiS in predstavitvijo spletne strani, preko katere deluje. Osrednji del se osredotoča na načrtovanje in izvedbo porazdeljevanja računanja, ter izgradnjo podatkovne baze in njeno nadaljno optimizacijo s postopkom denormalizacije. Sledi opis izdelave uporabniškega vmesnika za iskanje po podatkih in primer njegove uporabe. V zaključku je implementirana rešitev ovrednotena, podanih pa je še nekaj dodatnih možnosti za izboljšanje sistema. Razvita podatkovna baza ProBiS je nova rešitev na področju strukturne primerjave proteinov in v veliki meri izboljša uporabniško izkušnjo spletnega strežnika ProBiS.

## Ključne besede:

ProBiS, protein, podatkovna baza, porazdeljevanje računanja, denormalizacija, uporabniški vmesnik

# Abstract

The work described in this thesis builds upon ProBiS algorithm for comparison of protein structures. Such comparisons can be very time consuming. The main goal of our project was their pre-computation and development of the ProBiS database. Thesis begins with a description of ProBiS algorithm and a web page, through which it operates. Our main contribution is construction of appropriate architecture of the system and its implementation using distributed computation, and construction of a database and its further optimization through denormalization. We have also implemented the user interface for searching the database. Our newly developed ProBiS database is a novel resource for protein structural comparisons and greatly improves the user experience of ProBiS web server.

### Key words:

ProBiS, protein, database, distributed computation, denormalization, user interface

# Poglavje 1

# Uvod

Skozi zgodovino smo ljudje z razvojem znanosti postajali vedno bolj sposobni spoznavati in opisovati svet okoli nas. Z opazovanjem, testiranjem, preizkušanjem in drugimi eksperimentalnimi metodami, se nenehno veča količina podatkov, potreba po njihovi interpretaciji in prenašanju novih informacij na prihodnje rodove. Razvoj računalništva je ponudil izjemno možnost, ki je s časom prešla že v potrebo, da se veliko nalog s področja znanosti z uporabo računalniške obdelave in shranjevanja podatkov močno olajša. Ker pa nekateri eksperimenti še vedno trajajo veliko časa in ker smo ljudje običajno nepotrpežljivi, je razvoj šel še stopničko višje z algoritmi za predvidevanje in moduliranje rezultatov določenih eksperimentov. Taki algoritmi pa so lahko izredno zapleteni in njihov čas izvajanja zelo dolg. Za uporabnika, ki si želi te podatke uporabljati konstantno in vsakodnevno, je seveda vsako čakanje nezaželeno in predstavlja problem. Rešitev tega problema je vnaprejšnje poganjanje algoritmov in shranjevanje rezultatov, ki so nato takoj dostopni. Vendar je zahtevnost algoritmov lahko še vedno tako velika, da bi izračuni na enem samem računalniku lahko trajali leta ali celo desetletja. Uporaba računalniških omrežij in gruč računalnikov omogoča rešitev takih problemov, seveda pod pogojem, da je problem dovolj vzporeden.

Vse to je zelo splošno in lahko velja za vsa področja znanosti, tudi za področji kemije in farmacije, s katerima je povezana ta diplomska naloga. V tem delu je glavni poudarek predvsem na molekularnem modeliranju, oziroma natančneje, na sistemu ProBiS. Ta je bil razvit z namenom lažjega modeliranja vezavnih mest in lastnosti proteinov, kot eden od načinov uporabe pa se kaže tudi olajšan razvoj novih zdravil [6]. To diplomsko delo je precej interdisciplinarne narave, vendar smo poskušali teme, ki ne sodijo v sklop računalništva, pojasniti na čimbolj osnoven in razumljiv način. Opis in osnove problema

so predstavljeni v poglavju 2. Namen raziskovalnega dela diplomske naloge je bila izgradnja računalniške podatkovne baze, katera bi vsebovala vnaprej izračunane rezultate primerjav vseh proteinov med seboj. To je omogočilo večjo interaktivnost, uporabnost in s tem boljšo uporabniško izkušnjo sistema ProBiS. V istem poglavju je opisan tudi algoritem ProBiS in gruča računalnikov na Kemijskem inštitutu, ki je bila uporabljena za računanje.

Pri delu je bil v veliki meri pomemben način shranjevanja podatkov, zato so v poglavju 3 opisane podatkovne baze v splošnem, podrobneje pa podatkovna baza MySQL, ki jo je algoritem ProBiS že predhodno uporabljal. Opisanih je tudi nekaj vrst modernejših podatkovnih baz, ki se vse več uporabljajo za shranjevanje velikih količin podatkov. Drugi pomemben vidik poleg prostora je v računalništvu seveda čas, zato je v istem poglavju podan pregled osnov porazdeljenega računanja in gruč računalnikov. Zraven so dodani še opisi nekaj programskih jezikov in ostalih tehnologij, ki so bile uporabljene pri delu.

V poglavju 4, ki je tudi najpomembnejše in verjetno najzanimivejše, sta opisani priprava in potek dela ter samo računanje primerjav. Poleg tega so predstavljeni problemi, s katerimi smo se soočali, in njihove rešitve. Za poizvedovanje po pridobljenih podatkih je bil pripravljen poseben uporabniški vmesnik, katerega lastnosti in izdelava so prav tako vključene v vsebino poglavja.

V zadnjem poglavju so opisane še sklepne ugotovitve, ki smo jih glede na zastavljene cilje tudi ovrednotili. K temu so dodane še ideje za nadaljnje delo oziroma izboljšave. Podanih je tudi nekaj predlogov alternativnih rešitev, ki bi v prihodnosti še bolje opravljale samo delo shranjevanja in iskanja podatkov.

# Poglavje 2

# Pregled problema in obstoječih rešitev

Za določitev smernic in ciljev pri opravljanju dela se je najprej potrebno seznaniti s področjem dela in trenutnimi rešitvami, ki so na voljo. Ker je področje dela v tem primeru zelo interdisciplinarno, je to poglavje namenjeno predvsem podajanju in razlaganju pojmov s področja kemije in področja farmacije, ki so neposredno povezani z obravnavanim problemom. V nadaljevanju je nato podrobneje opisan sistem ProBiS, katerega izboljševanje je bilo osnova in bistvo našega dela. Samo delovanje algoritma ProBiS je nato predstavljeno še malo bolj podrobno, saj je po našem mnenju zanimivo tudi za računalniške strokovnjake, ker se v samem sistemu uporabljajo tudi zahtevne metode iz te stroke.

## 2.1 Problemska domena

Proteini so velike molekule, sestavljene iz nabora 20 standardnih aminokislin, ki so nanizane v več sto aminokislin dolge verige. Aminokislina je na splošno vsaka molekula, ki vsebuje tako amino (-NH2) kot karboksilno (-COOH) funkcionalno skupino. Proteini se sintetizirajo na ribosomu in so na začetku linearna veriga aminokislin, nato pa se zvijejo v točno določeno 3D strukturo, določeno s tem zaporedjem aminokislin. Proteini v organizmu vršijo različne naloge: od kataliziranja kemijskih reakcij, prenašanja sporočil iz zunajceličnega prostora v celico, do definiranja trdnosti, celovitosti in oblike celice. Proteini so tudi sposobni medsebojne komunikacije, saj lahko prilagajajo svoje funkcije glede na trenutne razmere v celici. Ob trku se lahko namreč začasno ali trajno povežejo. Kadar se vežejo s točno določenimi predeli svojih površin

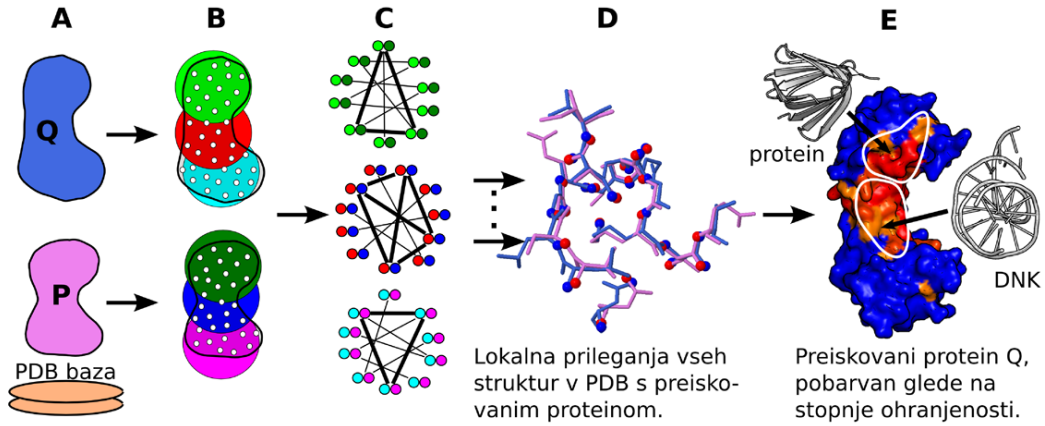
(vezavnimi mesti), ki so si komplementarna, tvorijo proteinske komplekse, pri čemer se spremenijo funkcije udeleženih proteinov. Vsak protein lahko vstopa v več takih interakcij, kar lahko najlažje predstavimo v obliki matematičnega grafa, kjer so točke proteini, povezave pa interakcije med proteini. Tak graf si lahko predstavljamo tudi kot analogijo z družinskimi drevesi, ki se uporabljajo za prikaz relacij med družinskimi člani. Pri modeliranju proteinskih interakcij igra pomembno vlogo površina proteina, manjšo pa notranjost, zato je tudi pomembna sama 3D struktura, v katero se protein zvije. Vezavna mesta proteinov se skozi evolucijo spreminjajo počasi in so ohranjena v strukturah sorodnih proteinov, vsaka mutacija aminokisline v vezavnem mestu pa skoraj vedno povzroči porušenje funkcionalno aktivne strukture proteina. Mutacija s pozitivnim učinkom na preživetje organizma se zgodi zelo redko, oziroma je pozitivna samo v posebnih okoliščinah. Aminokisline, ki niso udeležene pri vezavi molekul ali stabilizaciji proteina, lahko mutirajo precej bolj svobodno, saj nimajo tako velikega vpliva na funkcijo proteina. Zato se lahko proteini z enako biokemijsko funkcijo in lokalno podobnimi vezavnimi mesti močno razlikujejo v globalni prostorski postavitvi elementov. V praksi to pomeni, da lahko pri primerjanju strukture neznanega proteina z znanimi strukturami iz centralne baze proteinov, odkrijemo podobnosti, ki praviloma sovpadajo z vezavnimi mesti, kar nam omogoča tudi napoved funkcije preiskovanega proteina [6].

Za določanje 3D strukture proteinov se uporabljajo metode rentgenske difrakcijske kristalografije in nuklearne magnetne resonance. Ker sta ti metodi precej zahtevni, je znanih relativno malo proteinskih struktur. Centralna baza proteinov, dosegljiva na naslovu `http://www.pdb.org`, vsebuje vse do zdaj odkrite proteinske strukture (trenutno okrog 73.000), vsako leto pa se poveča za nekaj manj kot 8.000 struktur [2, 6]. Vsakemu proteinu iz te baze je določeno ime, ki je sestavljeno iz štirih številk in črk angleške abecede [14]. Definiran je tudi poseben format tekstovnih datotek, ki se uporabljajo za shranjevanje struktur raziskanih proteinov. Take datoteke imajo končnico `.pdb`, vsebujejo pa zapis položaja atomov in molekul v samem proteinu. Predstavljajo standard, zato jih uporabljajo vsi programi, ki se na kakršenkoli način ukvarjajo s tem področjem [15].

## 2.2 Algoritem ProBiS

Še bolj kot odkrivanje novih proteinov, zaostajajo raziskave, ki določajo biokemijske funkcije že odkritih proteinov. Algoritem ProBiS je bil razvit z na-

menom iskanja vezavnih mest na proteinih [6]. Njegovo delovanje je mogoče videti na sliki 2.1.




Slika 2.1: Poenostavljen prikaz delovanja algoritma ProBiS. A: Preiskovana proteinska struktura (Q) se primerja z vsemi ostalimi (P) iz baze PDB. B: Površina vsake strukture je interno predstavljena kot proteinski graf, ki je razdeljen na manjše podgrafe. C: Vsak podgraf enega proteina se primerja z vsemi podgrafi drugega proteina in za vsako tako primerjavo se tvori produktni graf, v katerem se poiščejo maksimalne klike. D: Te predstavljajo optimalna lokalna prileganja proteinskih struktur. E: Rezultat je 3D površina proteina, obarvana glede na strukturno ohranjenost površinskih aminokislin.

ProBiS je zasnovan na učinkovitem algoritmu na osnovi teorije grafov, ki omogoča primerjave površinskih struktur proteinov na lokalnem nivoju. Grafi so množice vozlišč in povezav med njimi, obe vrsti elementov pa lahko vsebujeta različne atribute. Grafu z atributi pravimo tudi označeni graf. Pri algoritmu ProBiS vozlišča predstavljajo točke v prostoru, njihovi atributi pa fizikalno-kemijske lastnosti funkcionalnih skupin aminokislin. Povezave med vozlišči so določene glede na njihovo medsebojno razdaljo (če je razdalja dovolj majhna, povezava obstaja, v nasprotnem primeru pa ne). Podobnost med dvema takima grafoma je sorazmerna z velikostjo njunega največjega skupnega podgrafa. Tako se najprej iz dveh proteinskih grafov tvori t.i. produktni graf, na katerem se poišče maksimalna klika, to je popolnoma povezan graf, ki predstavlja skupni imenovalec oziroma največjo podobnost med izvornima grafoma. ProBiS tako preiskovano strukturo zaporedno prilega z vsako iz baze in izračuna stopnjo ohranjenosti za vse aminokisline v preiskovanem proteinu. Tiste z visoko stopnjo ohranjenosti so odgovorne za funkcijo proteina in običajno

pripadajo vezavnim mestom, zato lahko za proteine s podobnimi vezavnimi mesti sklepamo, da imajo podobno funkcijo. Algoritem ProBiS za primerjave ne uporablja dejanske celotne baze PDB, ampak samo njen manjši del, ki ga izbere na podlagi gručenja vseh proteinov. Proteine, ki so si v 95 odstotkih podobni po strukturi, združi v skupno gručo, za primerjave pa uporablja samo najbolj reprezentativnega predstavnika iz vsake gruče. S tem principom se iskalni prostor iz 73.000 zmanjša na okrog 27.000 struktur (toliko gruč namreč nastane pri tem postopku). Ta postopek se izvede 1-krat tedensko, ko sistem ProBiS preveri, če so bile v centralno bazo PDB dodane kakšne nove strukture in posodobi svoj seznam gruč [4].

## 2.3 Spletni strežnik ProBiS

S ProBiS ni pojmovan samo algoritem za računanje podobnosti proteinov, ampak tudi spletna stran, prek katere ga lahko uporabljamo. Dosegljiva je na spletnem naslovu `http://probis.cmm.ki.si/`. Uporabnik v okence vpiše PDB identifikator proteina in verige oziroma naloži svojo datoteko v formatu PDB. Ti podatki se nato pošljejo na strežnik, ki v ozadju zažene algoritem ProBiS, ki primerja zahtevani protein z vsemi ostalimi, rezultate pa shrani v ločeno podatkovno bazo MySQL. Ta proces lahko traja od nekaj deset minut do nekaj ur, odvisno od velikosti primerjanega proteina. Izdelana podatkovna baza zaseda od 5 MB pa vse do 100 MB in več prostora na disku. Ob uspešnem zaključku svojega dela, sistem posreduje uporabniku povezavo na rezultate prek elektronskega sporočila oziroma kar prek spletne strani, če je ta še vedno aktivna na uporabnikovi strani. Pri kliku na to povezavo, ProBiS prebere rezultate iz podatkovne baze, ter ustvari stran z njihovo predstavitvijo. Glavni del dinamično ustvarjene strani je vtičnik JMol, prek katerega je mogoče grafično, v 3D prostoru, videti preiskovani protein, pobarvan glede na njegovo strukturno podobnost z ostalimi v bazi. Na desni strani zaslona so prikazana prileganja preiskovanega proteina z najdenimi strukturno podobnimi proteini iz baze [5]. Stran z rezultati je prikazana na sliki 2.2.

Podatkovna baza z rezultati je sestavljena iz treh tabel (več o podatkovnih bazah v poglavju 3.1):

- main - vsebuje stolpce z imeni parov primerjanih proteinov in verig ter identifikator za povezavo z ostalima tabelama.

- cluster - vsebuje podatke o posameznem ujemajočem delu proteinov, ti so sestavljeni predvsem iz ocen o kakovosti ujemanja ter rotacijskih in

pozicijskih vrednosti v 3D prostoru. Vsebuje tudi povezavo na identifikator iz main tabele. Na eno vrstico iz main tabele se nanaša med 0 in 5 vrstic iz te tabele.

- result - vsebuje identifikatorje aminokislin v določenem ujemajočem delu proteinov. Ta tabela je neposredno povezana s cluster tabelo, na eno vrstico iz cluster tabele se nanaša okrog 100 do celo 1000 vrstic iz te tabele.

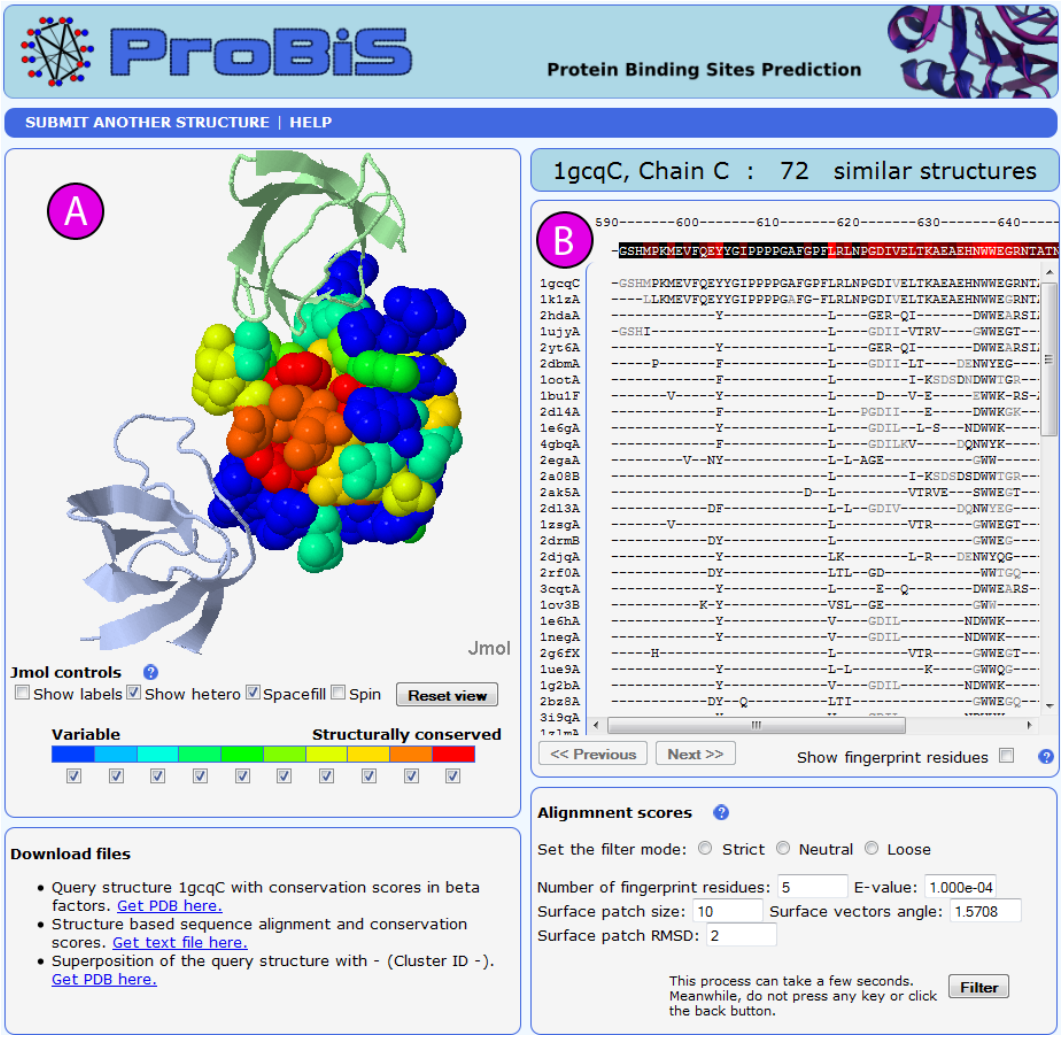
Strukturo podatkovne baze in povezave med njenimi tabelami je mogoče videti na sliki 2.3. Po prejemu zahtevka za pregled rezultatov, sistemu sploh ni potrebno iskati po tako zgrajeni podatkovni bazi, saj vsi podatki v njej pripadajo točno enemu, preiskovanemu proteinu.

Največji problem samega sistema je časovna zahtevnost računanja primerjav preiskovanega proteina z bazo vseh proteinov. Zaradi tega, ker lahko tako računanje traja tudi nekaj ur, je sistem precej neinteraktiven in uporabnik je primoran čakati veliko preveč časa na svoje rezultate. Pospešitev zahtevnega procesa na način predhodnega računanja vseh primerjav proteinov je bila tudi glavna motivacija za naše delo.

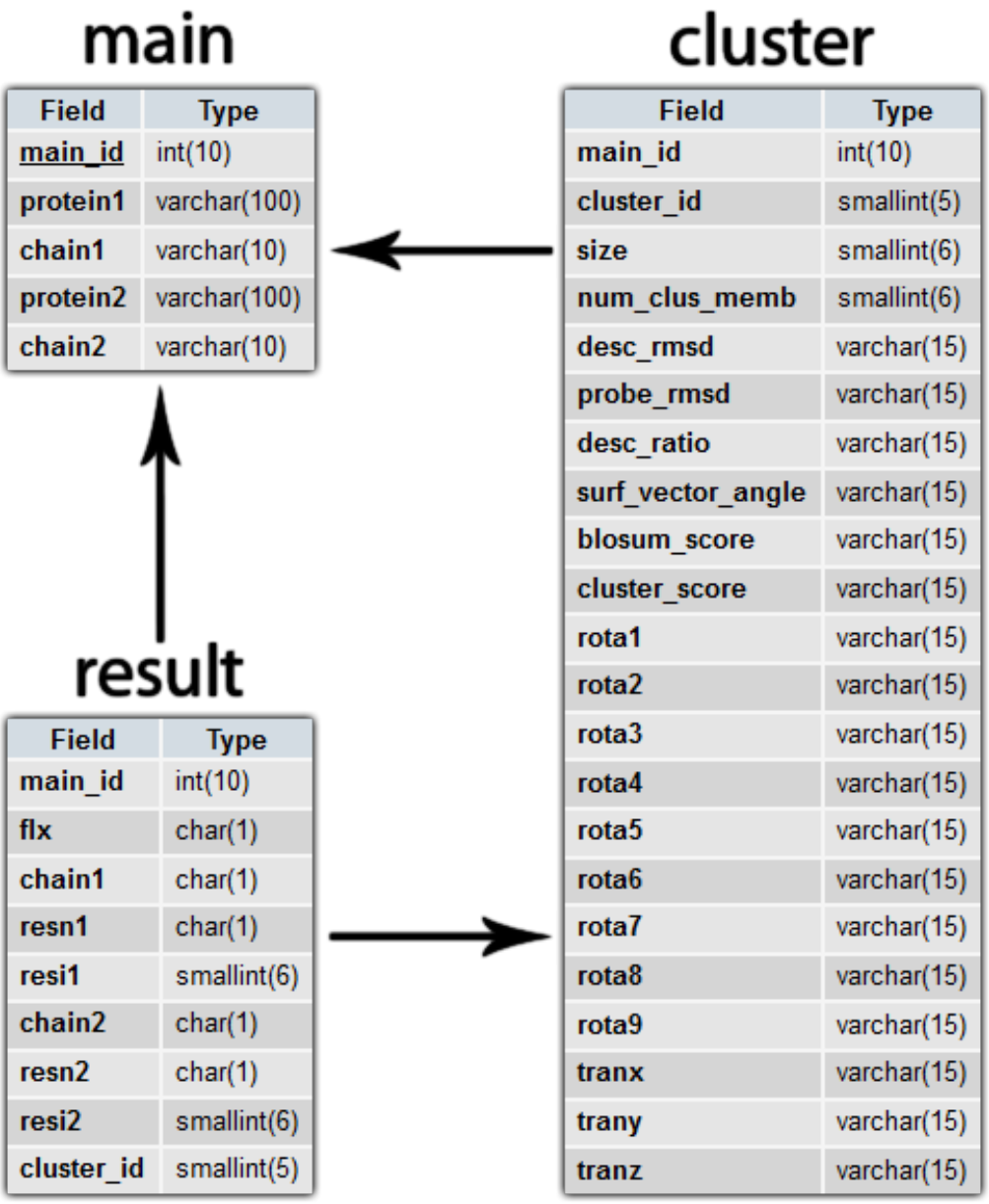
## 2.4 Gruča računalnikov na KI

Spletni strežnik ProBiS je nastanjen na enem izmed računalnikov Kemijskega inštituta. En sam strežnik pa je premalo za zahtevne izračune podobnosti med proteini, zato je le ta povezan z gručo 17 ostalih računalnikov, ki so združeni pod imenom Kavka. Vsi notranji računalniki so povezani preko istega mrežnega stikala in posledično direktno dosegljivi med seboj, kar je pomembno zaradi vzporednega računanja problemov. Na zunaj je dosegljiv le glavni strežnik, na katerem je nastanjena spletna stran. Spletni strežnik ob prejemu zahtevka za računanje primerjav določenega proteina z ostalimi, postavi tak zahtevek v vrsto. Iz vrste ga nato prevzame prvi prosti računalnik iz prej opisane gruče, ki tudi opravi izračune [5]. Za namene diplomske naloge smo omejili računanje tako poslanih zahtevkov uporabnikov na 3 računalnike, tako da smo za naše izračune lahko uporabljali 14 ostalih računalnikov.

Na vsakem od računalnikov tečeta operacijski sistem Linux in strežnik MySQL. Poleg tega je vsak izmed računalnikov tudi dobro strojno opremljen, saj vsebuje 4 štiri-jedrne procesorje Intel Xeon, 24 GB pomnilnika in vsaj 1 TB diskovnega prostora. Ker smo imeli na voljo 14 strežnikov, vsakega s 16 efektivnimi procesnimi enotami, smo lahko za računanje primerjav med vsemi proteini uporabili vsega skupaj 224 procesnih enot [5].




Slika 2.2: Stran z rezultati, ki jo sistem ProBiS ustvari za primerjave preiskovanega proteina z bazo proteinov. A: 3D struktura preiskovanega proteina, katerega atomi so pobarvani glede na njihovo strukturno ohranjenost v bazi proteinov. B: Prileganja proteinov iz baze s preiskovanim proteinom, ki si jih je mogoče ogledati v 3D prostoru, s klikom na črkovna zaporedja.




Slika 2.3: Struktura tabel in povezave med tabelami v podatkovni bazi, ki jo ProBiS ustvari pri računanju primerjav proteina z bazo vseh proteinov.

# Poglavje 3

# Uporabljena orodja in metode

Sistem ProBiS, ki smo ga izboljševali, je zelo raznolik, zato je bilo tudi pri našem delu uporabljenih več različnih principov in orodij. Naslednje poglavje tako govori o programskih jezikih, shranjevanju podatkov in procesiranju na splošno. Najprej so opisane podatkovne baze in razlogi za njihovo uporabo. Osredotočili smo se predvsem na relacijske podatkovne baze in MySQL, ker je bila prav ta uporabljena pri našem delu. Opisanih je tudi nekaj drugih, novejših podatkovnih baz, ki se vse več uporabljajo predvsem pri velikih spletnih storitvah in socialnih omrežjih. Predstavljale bi lahko drugačen pristop, ki bi ga bilo mogoče uporabiti za dodatno izboljšanje našega sistema. V nadaljevanju je nato v splošnem opisano porazdeljeno računanje in gruče računalnikov. Na koncu poglavja se spustimo še v opis različnih programskih jezikov in standardov, ki so bili uporabljeni pri praktičnem delu diplomske naloge.

## 3.1 Podatkovne baze

Podatkovna baza je organizirana zbirka podatkov skupaj z njenim opisom. Ko gre za računalniško shranjevanje podatkov, običajno govorimo kar o sistemu za upravljanje s podatkovno bazo. Tak sistem je dejansko programska oprema, ki skrbi za shranjevanje, branje, brisanje in urejenost podatkov. Sami podatki oziroma način njihovega shranjevanja je v splošnem prezapleten za uporabo brez take programske opreme. Programi, ki jo uporabljajo, postanejo z njo neodvisni od shranjevanja in strukturiranja podatkov, zato lahko iste podatke uporablja prek istega vmesnika več različnih programov. Na trgu je trenutno veliko produktov za upravljanje s podatkovnimi bazami (Oracle, MS SQL, MySQL, Access, DB2, Postgres ...), ki so si v osnovi podobni, vendar v praksi običajno nezdružljivi med seboj. Obstaja več vrst podatkovnih baz, ki se

v grobem razlikujejo glede na njihov logični podatkovni model (mrežni, hierarhični, relacijski, objektni . . . ) [8]. V našem primeru se bomo osredotočili na relacijski podatkovni model, saj ga uporablja tudi podatkovna baza MySQL, ki jo za svoje potrebe shranjevanja podatkov izkorišča algoritem ProBiS.

Relacijski podatkovni model je bil razvit leta 1970 in je pomenil revolucijo na tem področju, saj je nadomestil veliko starejših modelov. Njegovi prednosti sta, da je definiran formalno in osnovan na matematičnih strukturah ali relacijah. Te so predstavljene s tabelami, ki so človeku dobro razumljive. Vsaka tabela ima stolpce, ki predstavljajo atribute relacije in vrstice, od katerih vsaka predstavlja eno n-terico relacije. Vrstni red stolpcev in vrstic je nepomemben, pomembno je le, da so si vrstice med seboj različne. Vsaka relacija definira svoj ključ, ki je dejansko podmnožica atributov, ki določajo vsako n-terico. Tuji ključ je podobno podmnožica atributov, ki je enaka ključu neke druge relacije, zaradi česar se lahko tabele tudi povezujejo med seboj. Za poizvedovanje po relacijah sta definirana 2 formalna jezika, to sta relacijska algebra in relacijski račun. V praksi se večinoma uporablja poizvedovalni jezik SQL, ki je ekvivalenten prej omenjenima jezikoma in ga uporablja tudi MySQL. Je enostaven, nepostopkoven, standardiziran in transformacijsko usmerjen jezik. Sestavljata ga dve skupini ukazov: za definiranje strukture podatkovne baze ter za poizvedovanje in ažuriranje samih podatkov [8].

Pri relacijskih podatkovnih bazah je v veliki meri pomembno dobro načrtovanje primernih relacij, ki ustrezajo potrebam podatkovne domene. Primerne relacije določa nekaj lastnosti, kot so: minimalen nabor atributov, minimalna redundanca med atributi (z izjemo tujega ključa, ki služi za povezavo med relacijama in je lahko predstavljen večkrat) ter zajemanje logično povezanih atributov v isto relacijo. Tak postopek imenujemo normalizacija. Prednosti normaliziranih podatkovnih baz sta boljša izraba prostora na diskih ter enostavnejše delo s podatki. Poleg tega pri dobro normaliziranih bazah ne more priti do ažurnih anomalij. Obstaja več stopenj normalnih oblik:

- Prva normalna oblika (1NO) - v eni celici tabele je samo ena vrednost.
- Druga normalna oblika (2NO) - veljati mora 1NO, poleg tega pa so v vsaki tabeli vsi atributi, ki niso del ključa, funkcionalno odvisni od celotnega in ne samo od dela primarnega ključa.
- Tretja normalna oblika (3NO) - veljati mora 2NO, polega tega pa med atributi, ki niso del primarnega ključa, ni odvisnosti.
- Četrta normalna oblika (4NO) - veljati mora 3NO, poleg tega pa ne smejo obstajati atributi, ki bi bili odvisni od vrednosti primarnega ključa.

Denormalizacija je obraten postopek kot normalizacija. Gre za zavestno kršenje pravil normalizacije, z namenom doseganja boljše učinkovitosti, s čimer poslabšamo fleksibilnost in neodvisnost podatkov [10].

### 3.1.1 MySQL

MySQL je sistem za upravljanje z relacijsko podatkovno bazo, ki deluje kot strežnik in zagotavlja več-uporabniški dostop do večjega števila podatkovnih baz. Napisan je v programskih jezikih C in C++, deluje na veliko različnih sistemskih platformah, hkrati je njegovo delovanje podprto v večini najbolj uporabljenih programskih jezikov. Je tudi odprtokoden in brezplačen in zaradi tega izredno popularen v svetu. Uporabljajo ga aplikacije YouTube, Wikipedia, Google, Facebook in še veliko ostalih. Prav tako je uporabljen v ogrodjih za razvijanje spletnih strani in drugih storitev, kot so Wordpress, Joomla, phpBB, Drupal, Zend Framework ... Za povezovanje na strežnik MySQL lahko uporabimo kar ukazno vrstico, knjižnice za posamezne programske jezike ali pa katerega od bolj sofisticiranih uporabniških vmesnikov, kot so MySQL Workbench, Adminer, Navicat, phpMyAdmin ... Slednjega smo zaradi svoje brezplačnosti in preprostosti tudi uporabili pri našem delu za izdelavo sheme tabel, poganjanje krajših poizvedb in izpis statistik [10].

### 3.1.2 NoSQL

NoSQL ni bil uporabljen pri izdelavi te diplomske naloge, opisan je le z namenom mogočih prihodnjih izboljšav. Seveda se s tem pojavlja vprašanje zakaj ravno NoSQL in kaj ta princip sploh pomeni. NoSQL je razred sistemov za upravljanje s podatkovnimi bazami. Je novejši princip, prvič se ga pod takim imenom omenja leta 1998. Od klasičnih relacijskih podatkovnih baz se razlikuje v nekaj pomembnih lastnostih: podatki ne potrebujejo fiksnih tabelaričnih shem, sistemi NoSQL se izogibajo stičnim operacijam in se dobro skalirajo po horizontalnem obsegu. Prav tako ne uporabljajo jezika SQL za delo s podatki (od tod tudi ime). Uporaba sistemov NoSQL je v porastu, saj se vedno več uporabljajo predvsem pri velikih spletnih storitvah, kot so socialna omrežja, spletne trgovine, oddaljeno shranjevanje podatkov ... Obstaja več vrst podatkovnih modelov NoSQL, vsaka vrsta pa realizira drugačno shranjevanje in povezovanje podatkov:

- Dokumentni model - shranjevanje v dokumente XML ali JSON, ki nimajo fiksne sheme in so med sabo neodvisni. Uporablja se v izvedbah MongoDB, CouchDB, RavenDB ...

- Model grafa - gre za shranjevanje podatkov s pomočjo matematičnih grafov (vozlišča in povezave med njimi). Delo s takimi grafi ponavadi poteka preko nekega začetnega vozlišča, na katerem začnemo z iskanjem v globino oziroma v širino. Najbolj značilni predstavniki so Neo4j, HyperGraphDB ...

- Stolpčni model - podatki so shranjeni v tabeli, vendar so njeni stolpci definirani dinamično. Je eden od vodilnih modelov, ki je veliko prispeval pri nastanku gibanja NoSQL. Izvedbe tega modela uporabljajo Google, Facebook, Twitter, Digg, Reddit ... Najbolj značilni predstavniki pa so HBase, Cassandra, Hypertable, BigTable ...

- Model ključ-vrednost - deluje na osnovi globalnih razpršenih tabel, pri katerih s ključem vrnemo vrednost. Izvedbe: Voldemort, Dynomite, v določeni meri HBase in Cassandra.

Skoraj vse izvedbe NoSQL imajo običajno dobro podporo za porazdeljeno shranjevanje podatkov (na več ločenih strežnikih), s čimer lahko na preprost način razširimo zmožnosti sistema [16].

## 3.2 Porazdeljeno računanje

Porazdeljeno računanje se ukvarja z reševanjem računskih problemov na porazdeljenih sistemih. To so računalniški sistemi, sestavljeni iz več avtonomnih računalnikov, ki komunicirajo prek računalniškega omrežja, z namenom doseganja istega cilja. Na takih sistemih teče porazdeljen program, katerega naloga je med drugim tudi razdelitev problema na več manjših, ki so nato lahko rešeni na vsakem izmed posameznih računalnikov. Tipične lastnosti porazdeljenih sistemov so:

- sistem mora sam poskrbeti za obravnavanje napak na posameznih računalnikih,

- struktura sistema, pri čemer mislimo na število računalnikov, mrežno topologijo, itd., ni vnaprej poznana in se lahko spreminja,

- vsak posamezen računalnik pozna le svoj del podatkov in ima omejen pogled na sistem kot celoto.

Porazdeljeno računanje ima veliko podobnosti z vzporednim računanjem, saj pri obeh procesorji hkrati in vzporedno računajo določen problem. Obe vrsti

računanja ločimo po kriteriju, da pri vzporednem računanju procesorji uporabljajo isti pomnilnik in si prek njega tudi prenašajo informacije, pri porazdeljenem računanju pa ima vsak procesor svoj pomnilnik, informacije z ostalimi pa si deli s pomočjo posredovanja sporočil, kar tipično poteka preko omrežja [1, 7].

Obstaja več arhitektur porazdeljenega računanja, med katerimi je najbolj standardna arhitektura strežnik-odjemalec, ki smo jo uporabili tudi pri izdelavi naše aplikacije za porazdeljeno računanje. Gre za arhitekturo, ki loči med dvema entitetama:

- strežnik - deluje kot ponudnik storitev in virov, čaka na povezave in zahtevke,
- odjemalec - ne deli svojih virov, sproži zahtevek po viru oziroma storitvi.

Taka arhitektura se uporablja za zelo veliko aplikacij in protokolov: HTTP, FTP, SSH, SVN, DNS, podatkovne baze in še bi lahko naštevali. Njena uporabnost izvira predvsem iz mnogih prednosti, ki jih prinaša, kot je lažje vzdrževanje, centralno hranjenje podatkov in centralni nadzor (večja varnost), veliko dobro razvitih in dodelanih implementacij, delovanje z različnimi odjemalci, ki imajo lahko različne sposobnosti ... Ena od najpomembnejših slabosti, ki jo je potrebno vzeti v obzir je, da zaradi velikega števila sočasnih zahtevkov strežnik lahko postane preobremenjen, njegovo delovanje pa se poslabša, kar lahko v skrajnem primeru privede do prenehanja delovanja in posledično nedosegljivosti storitve [1].

## 3.3 Programski jeziki

Programski jeziki so si v splošnem med seboj podobni, predvsem v tem, da je z vsakim mogoče rešiti določen računski problem, če le ustreza pogojem izračunljivosti [13]. Razlikujejo pa se v nekaj pomembnih lastnostih, kot so: hitrost izvajanja, zahtevnost uporabe, preglednost, zahtevnost vzdrževanja kode, podpora programskim knjižnicam, itd. V naslednjih podpoglavjih so opisani programski jeziki, ki so bili uporabljeni pri izdelavi različnih aplikacij in skript v tej diplomski nalogi. Trudili smo se uporabiti predvsem tiste jezike, ki so bili v času pisanja standardni in najpogosteje uporabljani jeziki za reševanje določenih problemov.

### 3.3.1 Java

Java je eden novejših, objektno usmerjenih programskih jezikov, ki je bil leta 1995 razvit v Sun Microsystems (danes Oracle). Sintaksa je podobna C in C++, le da ima Java poenostavljen objektni model, ki na primer določa, da vsak razred lahko deduje samo od enega drugega razreda. Ima tudi manj nizkonivojskih konstruktov, na primer ni kazalcev in direktnega dela s pomnilnikom. Aplikacije v Javi se prevedejo v strojno kodo navideznega računalnika (Java Virtual Machine - JVM). Posledica tega je, da jih lahko poganjamo na kateremkoli takem navideznem računalniku neodvisno od arhitekture dejanskega računalnika. Java je bila sicer v svojih začetkih kritizirana kot zelo počasna, vendar se je z novimi različicami, predvsem pa z uvedbo navideznega stroja HotSpot, ki uporablja naprednejše tehnike delovanja, kot na primer dinamično prevajanje, ki je dejansko združeno statično prevajanje in interpretacija, stanje tako izboljšalo, da jo danes na nekaterih hitrostnih področjih primerjajo celo s C++. V današnjih časih se Java zaradi svojih odlik uporablja na zelo veliko področjih, kot so strežniške aplikacije, spletne strani, uporabniški vmesniki, igre, aplikacije za mobilne telefone, itd. [3]

Pri našem delu smo Javo zaradi njene preprostosti uporabili za hiter razvoj aplikacije za porazdeljevanje izračunov strukturnih primerjav med proteini, ki je opisana v poglavju 4.3.

### 3.3.2 C++

C++ je splošen programski jezik, razvit leta 1979 kot nadgradnja jezika C. Dodaja mu razrede, virtualne funkcije, prepisovanje operatorjev, večkratno dedovanje, predloge in delo z izjemami. Je eden najbolj široko uporabljanih programskih jezikov, saj se uporablja za izdelavo sistemske programske opreme, gonilnikov, vgrajenih sistemov, visoko zmogljivih aplikacij tipa strežnik-odjemalec, iger, itd. Omogoča uporabo več programskih paradigm (deklarativno, funkcionalno, proceduralno, modularno in objektno programiranje), kar ga naredi težjega za učenje in razumevanje, v primerjavi s katerimi od novejših programskih jezikov [11, 12].

Algoritem ProBiS je v celoti napisan v C++, kar je pomembno zaradi same hitrosti izvajanja. Pri praktičnem delu smo morali ProBiS modificirati na način, da dovoljuje zapisovanje in branje iz drugače oblikovane podatkovne baze; problem in rešitev sta opisana v poglavju 4.5.

### 3.3.3 PHP

PHP je splošno namenski skriptni jezik, ki je bil v začetku (leta 1995) ustvarjen za razvoj dinamičnih spletnih strani. S tem namenom ga je mogoče vstaviti v dokument HTML, ki ga spletni strežnik, ko je izbrana stran zahtevana, interpretira s primernim modulom PHP (za strežnik Apache, ki je bil uporabljen v našem delu, je to modul `mod_php`). Kasneje je bil nadgrajen tudi z možnostjo poganjanja iz ukazne vrstice. PHP je preprostejši kot večina ostalih jezikov, saj ni strogo objektno usmerjen, pri deklaraciji spremenljivk na primer ni potrebno navesti tipa spremenljivke ... Obenem je počasnejši od Jave in C++, vendar se ga zaradi svoje preprostosti uporablja za večino spletnih strani [9].

PHP se uporablja tudi na strežniku ProBiS za prikaz dinamičnih spletnih strani. V diplomski nalogi je bil uporabljen pri izdelavi gradnika za iskanje po podatkovni bazi.

### 3.3.4 HTML, CSS, JavaScript

Vse tukaj opisane tehnologije veljajo danes za standard na področju programiranja za splet. HTML je označevalni jezik spletnih strani. Sestavljen je iz več oznak, ki definirajo različne elemente in s tem strukturirajo sam dokument oziroma ustvarijo tako imenovani objektni model dokumenta. Spletni brskalniki preberejo tak strukturiran dokument in ustvarijo njegovo vizualno podobo. Pri tem si pomagajo tudi s stilskimi predlogami CSS, ki določajo različne stile posameznim elementom oziroma skupini le teh. JavaScript je skripten, objektno usmerjen, dinamičen jezik, namenjen predvsem uporabi v različnih brskalnikih. Lahko se ga vstavi v dokument HTML s pomočjo oznake `<script>`. V današnjem času se veliko uporablja za ustvarjanje dinamičnih in interaktivnih spletnih strani, saj lahko z njim direktno modificiramo prej opisani objektni model dokumenta. Poleg tega se uporablja za asinhrono komunikacijo med brskalnikom in strežnikom, kar je bolj znano kot princip AJAX [9].

Zgoraj naštete tehnologije so bile pri diplomski nalogi uporabljene za izdelavo vizualnega gradnika za iskanje po zgrajeni podatkovni bazi. Sledeči gradnik je bil izdelan kot skripta JavaScript, ki na mestu vstavitve v HTML doda nekaj svojih elementov (okenca za vpis iskalnih besed in gumb za potrditev). Uporabnik lahko poda tudi svojo stilsko CSS datoteko, v kateri določi izgled in velikost samega gradnika. Več o tem je opisano v poglavju 4.6.

# Poglavje 4

# Razvoj podatkovne baze ProBiS

Glavni cilj te diplomske naloge je računanje vseh primerjav proteinov med seboj in s tem izboljšanje interaktivnosti in uporabnosti spletnega strežnika ProBiS. Zaradi velike kompleksnosti računanja in velikosti pridobljenih podatkov, se je bilo za uspešno izvedbo treba poslužiti kar nekaj različnih tehnologij in pristopov s področja računalništva. V tem poglavju je opisan postopek našega dela. Začeli smo z analizo in načrtovanjem, ki sta pomemben del implementacije, zaradi določitve ciljev in metod. Nadaljevali smo z izvedbo, testiranjem in vrednotenjem. Ker z rezultati sprva nismo bili zadovoljni, smo uvedli določene spremembe in se poslužili nekoliko drugačnega pristopa shranjevanja podatkov. Na koncu smo implementirali še uporabniški vmesnik za poizvedovanje po novo formiranih podatkih. V istem vrstnem redu kot je zgoraj našteto, si po vrsti sledijo tudi podpoglavja v tem poglavju.

## 4.1 Načrtovanje

Razviti je bilo potrebno podatkovno bazo ProBiS, ki odpravlja slabost trenutnega sistema, da mora uporabnik za vsak preiskovani protein vsakič znova poganjati algoritem ProBiS, ki protein primerja z vsemi ostalimi, podatke pa shrani v novo zgrajeno, ločeno podatkovno bazo. Vsaka takšna operacija traja v povprečju 20-60 minut, velikost vsake nove baze pa znaša med 5 MB in 100 MB, nekatere so celo še večje. Na porabo prostora in časa seveda vpliva velikost preiskovanega proteina, natančneje število njegovih aminokislin [5]. Če grobo povzamemo, da vsako primerjanje proteina s 27.000 ostalimi traja v povprečju 40 minut na enem računalniku, zasede pa 60 MB prostora, lahko ocenimo, da bi izdelovanje podatkovne baze po principu vsi proti vsem na 14 prostih strežnikih trajalo okrog 53 dni, končna velikost take baze pa bi znašala 1620

GB, če bi podatke shranjevali tako kot do sedaj, v podatkovne baze MySQL, vsak protein v svojo. Za nas je bila najbolj kritična predvsem časovna komponenta, saj smo podatke hoteli pridobiti čimbolj hitro. Prostorska zahtevnost ni predstavljala velikega problema, saj smo ji zaradi dobre strojne opreme lahko preprosto zadostili.

Ugotovili smo tudi, da s takim principom uvajamo redundanco podatkov, saj se primerjava med proteinoma vedno zgodi dvakrat: pri primerjanju prvega z drugim in drugega s prvim. Pri uporabi obstoječega sistema shranjevanja podatkov tudi nimamo fleksibilnosti, če bi želeli kdaj v prihodnosti spreminjati ali brisati podatke. Vsako od teh operacij je namreč potrebno vedno opravljati na dveh mestih hkrati za posamezno primerjavo, oziroma na 27.000 mestih za en protein. Vse to so bili razlogi, ki so botrovali odločitvi za implementacijo posebne aplikacije za razporejanje računanja na vseh prostih strežnikih tako, da se primerjave med različnima proteinoma ne bi nikoli ponavljale. Obenem smo se odločili za uporabo skupne podatkovne baze, v kateri bi imeli za vsako primerjavo shranjen le en set vrednosti. S tem bi lahko prihranili polovico časa (26 dni) in prostora (810 GB) iz prvotnega izračuna, kar bi bil lep napredek, tako glede porabe prostora kot časa. Hkrati bi olajšali tudi ostale operacije nad podatki. Poudariti je potrebno tudi to, da je bil eden izmed zelo pomembnih pogojev izvedbe zahteva, da se po končanih izračunih vsi podatki nahajajo le na enem računalniku oziroma enem disku, saj smo morali zagotoviti, da se lahko gruča računalnikov v prihodnje uporablja za reševanje drugih problemov. V praksi je to pomenilo, da načrtovana podatkovna baza ni smela biti porazdeljena po gruči računalnikov, ampak je morala biti vsebovana na enem samem. Algoritem ProBiS že v osnovi zapisuje rezultate v podatkovno bazo MySQL, zato se je bilo preprosteje odločiti zanjo kot za kakšno alternativo, saj tako ni bilo potrebno posegati v samo kodo in delovanje ProBiS-a, ampak je bilo mogoče takoj začeti z računanjem.

## 4.2 Priprava gruče računalnikov

Za namen izgradnje enotne podatkovne baze ProBiS je bilo najprej potrebno pripraviti in konfigurirati gručo 14 računalnikov, ki smo jo imeli na voljo. Med poglavitne naloge tega sklopa je spadalo:

- nadgrajevanje programske opreme računalnikov na isto različico,
- kopiranje datotek, pomembnih za delovanje algoritma ProBiS na vsakega od računalnikov (to je bilo potrebno zaradi razbremenitve centralnega

strežnika, saj bi v nasprotnem primeru morali za vsako izvajanje ProBiS-a kopirati datoteke iz centralnega strežnika, kar bi ga lahko potencialno preobremenilo),

- konfiguracija MySQL-a na primerne parametre, odvisne od pomnilnika in diska na vsakem od strežnikov (nastaviti je bilo potrebno poti za shranjevanje podatkovne baze na prazen delovni disk ter povečati omejitve za medpomnilnik). Ta del je bil pomemben zaradi podobnega razloga kot prejšnji, podatkov namreč nismo hoteli shranjevati takoj na en sam disk na centralnem strežniku zaradi možnosti preobremenitve le tega, ampak smo jih raje shranjevali na vsak posamezen lokalni disk na vsakem izmed računalnikov posebej.

## 4.3 Implementacija aplikacije za porazdeljeno računanje

Za enakomerno porazdeljevanje in izvajanje računanja primerjav med proteini z algoritmom ProBiS na strežnikih smo v programskem jeziku Java implementirali aplikacijo, ki je razdeljena na dva dela: strežnik in odjemalec. V naslednjih podpoglavjih sta opisana oba dela, njuno delovanje in medsebojno komuniciranje.

### 4.3.1 Strežnik

Delovanje strežnika razdelimo na več sklopov, ki so po vrsti, tako kot si v izvajanju kode sledijo, tudi opisani v nadaljevanju.

Strežnik najprej odpre datoteko za zapisovanje svojega trenutnega stanja in zapiše vanjo čas začetka delovanja. Ta datoteka se uporablja za sprotno spremljanje delovanja strežnika, kar pri dolgotrajnem računanju omogoča odkrivanje morebitnih napak v izvajanju. Vanjo se zapisujejo vsi začetni in končni časi izvajanja računanja na odjemalcih in možne napake, ki se lahko zgodijo med računanjem, kot je na primer izpad katerega izmed odjemalcev. Datoteka uporablja za zapis podatkov format HTML in je med računanjem dosegljiva prek spletne strani ProBiS. Ogledati si jo je mogoče s spletnim brskalnikom, zaradi česar lahko na preprost način spremljamo potek računanja in v primeru napak ponovno vzpostavimo katerega od odjemalcev. Drugi razlog za uporabo te datoteke je obnavljanje v primeru napak ali izpadov strežnika, saj lahko ob ponovni vzpostavitvi strežnika datoteko preberemo in iz nje izluščimo stanje

strežnika pred izpadom, ter ga obnovimo in nadaljujemo z računanjem od tam naprej. Primer prikaza take datoteke v brskalniku je prikazan na sliki 4.1.

*27.06.2011 20:36* -> Definitions file '/home/probis/unstable/newbin/definicije.php' parsed successfully!
*27.06.2011 20:36* -> PServer started in '-a' mode!
*27.06.2011 20:36* -> PDB list '/home/probis/unstable/newbin/tmp/dump3_multiple' parsed successfully! Number of PDBs: **28174**
*27.06.2011 20:36* -> PDB list '/home/probis/unstable/newbin/tmp/dump3_multiple.91' parsed successfully! Number of PDBs: **28136**
*27.06.2011 20:37* -> Finished list parsing!
*27.06.2011 20:37* -> There are **63** structures in new pdb list.
*27.06.2011 20:37* -> There are **28111** structures in same pdb list.
*27.06.2011 20:37* -> There are **25** structures in old pdb list.
*27.06.2011 20:37* -> PServer is listening for new connections...
*27.06.2011 20:38* -> New client CONNECTED: **probis17.local**
*27.06.2011 20:38* -> Client '**probis17.local**' STARTED job (comparing protein '**3zqj**' with **28173** others).
*27.06.2011 20:39* -> Client '**probis17.local**' COMPLETED job (comparing protein '**3zqj**' with **28173** others).
*27.06.2011 20:39* -> Client '**probis17.local**' STARTED job (comparing protein '**3sam**' with **28172** others).
• • •
*27.06.2011 20:42* -> Client '**probis6.local**' started DELETE phase.
*27.06.2011 20:42* -> Client '**probis6.local**' LOST connection!
*27.06.2011 20:42* -> New client CONNECTED: **probis6.local**
*27.06.2011 20:42* -> Client '**probis6.local**' started DELETE phase.
• • •
*27.06.2011 21:01* -> Client '**probis15.local**' disconnected.
*27.06.2011 21:08* -> Client '**probis4.local**' disconnected.
*27.06.2011 21:08* -> PServer stopped successfully!

Slika 4.1: Primer strežniške datoteke za zapisovanje trenutnega stanja. Z modro barvo je zabeležen vsak začetek, z zeleno pa vsak konec računanja. Z rdečo barvo je posebej označen primer ko odjemalec izgubi povezavo s strežnikom.

Po vzpostavitvi sistema za zapisovanje pomembnih dogodkov delovanja, strežnik prebere konfiguracijsko datoteko PHP z nastavitvami za delovanje algoritma ProBiS, ki pravilno konfigurira zaganjanje programa ProBiS na naši gruči računalnikov. Ker je aplikacija napisana v Javi, smo morali za branje datoteke PHP in interpretiranje spremenljivk jezika PHP napisati svoj algoritem. Ta prebere vsa imena spremenljivk in njihove vrednosti iz konfiguracijske datoteke PHP in jih shrani v razpršeno tabelo, kjer je ime spremenljivke podano kot ključ za dostop do njene vrednosti. Za pravilno implementacijo tega algoritma je bilo potrebno večkrat pregledati celotno pridobljeno tabelo in jo popraviti, saj so vrednosti nekaterih spremenljivk lahko definirane z vrednostjo drugih spremenljivk.

Nato strežnik prebere datoteko z aktualnimi proteini, ki jih bomo primerjali med seboj; njihove kode PDB in verige se shranijo v seznam. Datoteka z aktualnimi proteini se vsak teden ob istem času na novo ustvari glede na nove proteine iz centralne baze PDB in glede na že opisan postopek gručenja. Datoteke s starimi seznami proteinov se ne brišejo, namesto tega se ob ustvarjanju nove datoteke preprosto preimenujejo, tako da ima vsaka svoj identifikator, ki

določa kdaj je bila aktualna. Zaradi tega se delovanje strežnika na tej točki loči na tri, med seboj izključujoče dele, glede na vhodne parametre, ki jih uporabnik poda pri zagonu programa:

- "`-n`" - ta opcija vključi način za računanje vseh primerjav. Uporabna je le za prvotno računanje podatkovne baze ProBiS.

- "`-a`" - določa avtomatski način delovanja, ki je uporaben za tedensko posodabljanje podatkovne baze ProBiS. V tem primeru strežnik sam poišče predhodno datoteko s proteini in jo prebere. Njene vrednosti prav tako shrani v ločen seznam. Ta način delovanja se tudi privzame in uporabi v primeru, če uporabnik ne uporabi nobenega od naštetih parametrov pri zagonu programa.

- "`-m <število>`" - določa ročni način delovanja, kjer pri zagonu programa podamo še identifikator starejše datoteke s proteini, ki se nato ravno tako prebere in shrani v ločen seznam. Ta opcija je primerna za posodabljanje podatkovne baze v primeru, če le ta že dolgo časa ni bila posodobljena zaradi neuporabe avtomatskega posodabljanja.

Pri uporabi zadnjih dveh opisanih načinov delovanja, mora strežnik upoštevati 2 seznama proteinov, ki jih najprej še preoblikuje. To stori tako, da iz obeh seznamov izloči enake proteine in jih shrani v nov, tretji seznam, ki določa nespremenjene proteine med obema datotekama. Tako v prejšnjih dveh seznamih ostanejo samo proteini, ki razlikujejo oba seznama. V seznamu aktualnih proteinov so zato shranjeni proteini, ki jih je potrebno strukturno primerjati med seboj. Prav tako je potrebno vsakega izmed njih primerjati tudi z vsemi proteini iz tretjega seznama, saj so le ti ravno tako vsebovani v najnovejši datoteki. Proteinov iz tretjega seznama ni potrebno primerjati med sabo, saj rezultati takih primerjav že obstajajo v podatkovni bazi. Proteine, ki so ostali v starem seznamu, pa je potrebno izbrisati iz podatkovne baze, saj njihovih primerjav ne potrebujemo več, zaradi tega ker so ob tedenskem dodajanju novih proteinov in gručenju izgubili svoj reprezentativni pomen. Pri uporabi prve opcije se edini obstoječ seznam upošteva kot seznam novih proteinov.

Strežnik na tej točki omogoča še dva druga, dodatna in neobvezna načina delovanja, ki ju je ravno tako mogoče določiti z vhodnimi parametri:

- "`-su`" - preskoči fazo posodabljanja podatkovne baze in opravi samo brisanje. Uporabno v primeru napak v fazi brisanja ali v primeru, ko smo že kako drugače izračunali nove primerjave med proteini.

- "`-sd`" - opravi samo posodabljanje podatkovne baze, brisanje pa izpusti. Uporabno v primeru, če bi kdaj v prihodnosti želeli shranjevati čisto vse rezultate.

Opisana načina nista med seboj izključujoča, njuna uporabnost pa je omejena le na primer tedenskega in ročnega posodabljanja podatkovne baze ProBiS.

Strežnik nato preide v stanje čakanja na povezave z odjemalci. Za vsakega odjemalca, ki se poveže z njim, strežnik ustvari novo programsko nit, ki skrbi za komunikacijo med njima. Ob vzpostavitvi nove niti, strežnik odjemalcu najprej pošlje inicializacijsko sporočilo, ki vsebuje nekatere nastavitve algoritma ProBiS, prebrane iz konfiguracijske datoteke, kot so različica algoritma, osnovni argumenti za pravilno zaganjanje in poti do datotek PDB, kjer se nahajajo dejanski podatki o strukturi posameznega proteina. To sporočilo se pošlje samo na začetku komunikacije, saj si odjemalec nastavitve zapomni in jih uporabi pri vsakem nadaljnjem računanju. Ta racionalizacija zmanjša obremenjevanje omrežnih in lokalnih virov v primerjavi s sprotnim pošiljanjem nastavitev. Komunikacijska nit po začetnem pošiljanju nastavitev preide v stanje čakanja na odgovor odjemalca. Odjemalec pošilja le eno vrsto sporočila, ki označuje, da je ta prost in da se mu lahko dodeli več dela. Ko strežnik prejme tako sporočilo, s pomočjo prej pridobljenih seznamov proteinov ustvari novo sporočilo o delu, ki vsebuje podatke, s katerimi proteini je potrebno primerjati določen protein. To sporočilo se pošlje odjemalcu, ki nato začne opravljati svoje delo. Hkrati se iz seznama novih proteinov odstrani aktualni protein, da ga v prihodnje ne pošiljamo več v primerjanje. Ta seznam mora biti zaradi možnega sočasnega dostopa sinhroniziran tako, da lahko do njega hkrati dostopa le ena programska nit, s čimer se izognemo kritičnim napakam, ki bi lahko nastale ob sočasnem brisanju in branju istih vrednosti. Če je seznam novih proteinov prazen in se sporočilo o opravljanju dela ne more več ustvariti, se odjemalcu pošlje sporočilo o brisanju, ki vsebuje seznam vseh proteinov, ki jih je potrebno pobrisati iz podatkovne baze. Komunikacijska nit si zapomni, da je njen pripadajoči odjemalec v stanju brisanja, tako da ob naslednjem sprejetem sporočilu iz odjemalca, le temu sporoči, naj se zaustavi. Hkrati strežniku sporoči, da ni potrebno več čakati na nove povezave odjemalcev. Le ta zato izstopi iz zanke za poslušanje novih povezav in počaka, da se zaustavijo vse ustvarjene niti, nazadnje pa se zaustavi še sam.

### 4.3.2 Odjemalec

Delovanje odjemalca je podobno delovanju strežnika v tem, da ga ravno tako določajo parametri, ki mu jih ob zagonu poda uporabnik. Parametri odjemalca

so:

- "`-h <naslov gostitelja>`" - naslov strežnika s katerim naj se poveže.
- "`-wt <število>`" - število dovoljenih vzporednih računov. Optimalna vrednost parametra je enaka številu procesorjev oziroma številu jeder računalnika, na katerem deluje odjemalec.
- "`-js <število>`" - število primerjav, ki jih algoritem ProBiS opravi v enem življenjskem ciklu programa. Ta vrednost ne vpliva veliko na hitrost delovanja, saj večino časa algoritem ProBiS porabi za računanje strukturnih primerjav proteinov, zelo malo pa za njegovo pravilno vzpostavitev.

Po inicializaciji se odjemalec poveže s strežnikom, ali pa se zaustavi z napako, če strežnik ni dosegljiv. Dosegljiv strežnik se odzove tako, da pošlje inicializacijsko sporočilo, ki ga odjemalec najprej shrani, nato pa ustvari programske delovne niti, katerih število je odvisno od nastavitve uporabnika. Te niti bodo skrbele za zaganjanje instanc algoritma ProBiS, potrebne pa so zaradi tega, ker ProBiS sam po sebi ne podpira večnitnega izvajanja. Mi smo želeli izkoristiti dane večopravilne zmožnosti računalnikov in ProBiS poganjati na vseh razpoložljivih procesorjih. Niti so po inicializaciji pasivne, kar pomeni da zaradi trenutnega pomanjkanja dela spijo in ne odžirajo računalniških virov. Odjemalec strežniku nato po isti povezavi pošlje sporočilo, da je pripravljen na delo in preide v stanje poslušanja. Ko sprejme sporočilo o delu, ga razbije na manjše kose, katerih velikost je odvisna od uporabnikove nastavitve, in jih shrani v seznam. Ta mora biti zaradi možnega hkratnega dostopa sinhroniziran, za kar v Javi poskrbimo s konstruktom `synchronized`, ki omogoča, da v določenem trenutku le ena nit uporablja ta seznam. Hkrati zbudi delovne niti, katerih delovanje izgleda tako:

- Nit iz seznama del vzame naslednje delo in ga ob tem izbriše iz seznama.
- Nato pripravi parametre za zaganjanje algoritma ProBiS, pri čemer si pomaga s podatki o trenutnem delu.
- V naslednjem koraku zažene ProBiS in počaka, da se proces zaključi.
- Po koncu se vrne na začetek (prevzem novega dela iz seznama).

V primeru, da je seznam del prazen, se nit vrne v pasivno stanje in to naznani odjemalcu. Če odjemalec pri tem ugotovi da so že vse niti v pasivnem stanju, kar pomeni da so končale z delom, to sporoči strežniku. Ta

lahko nato ponovno pošlje sporočilo o novem delu, ali pa sporočilo o brisanju. Slednje vsebuje seznam proteinov, katerih podatke je potrebno pobrisati iz podatkovne baze. Ob prejemu le tega se odjemalec zato poveže na lokalno podatkovno bazo MySQL in iz vseh tabel kaskadno (prek njihovih povezav) pobriše zahtevane podatke. Za povezovanje s podatkovno bazo MySQL smo uporabili posebno knjižnico `mysql-connector`, ki je na voljo na spletnem naslovu `http://dev.mysql.com/downloads/connector/j/`. Po uspešnem brisanju odjemalec to sporoči strežniku, ki takoj pošlje nazaj preprosto obvestilo, da je dela zmanjkalo in da se lahko odjemalec zaustavi.

## 4.4 Primerjanje proteinov z algoritmom ProBiS in združevanje rezultatov v enotno podatkovno bazo

V prejšnjem poglavju opisano aplikacijo smo izvajali na naši gruči računalnikov, tako da smo na centralnem strežniku poganjali strežniški del aplikacije za porazdeljevanje dela, ki ni preveč obremenil samega strežnika. Odjemalce smo pognali na 14 ostalih prostih računalnikih. Nastavili smo jih tako, da so uporabljali 15 programskih niti in tako vzporedno zaganjali 15 instanc algoritma ProBiS. Vsak računalnik ima sicer 16 računskih enot in vsaka instanca algoritma ProBiS v polni meri obremeni posamezno izmed računskih enot, zaradi česar smo eno pustili prosto, da bi se nemoteno lahko izvajali ostali procesi operacijskega sistema in samega odjemalca. Število primerjav, ki jih algoritem ProBiS izvede v enem svojem življenjskem ciklu, smo omejili na 20, kar je ustrezalo tudi prvotnemu delovanju algoritma in je preverjeno dobro delovalo.

Izvajanje je brez napak potekalo nekaj več kot 18 dni. Kot rezultat smo dobili okrog 75 GB veliko podatkovno bazo MySQL na vsakem izmed računalnikov, na katerih se je izvajal odjemalec. Iz tega lahko razberemo, da smo v prvotnem izračunu nekoliko precenili povprečno hitrost delovanja algoritma, saj so v povprečju primerjave porabile manj časa, kot smo napovedali (glej poglavje 4.1). Kljub temu bi po starem sistemu računanje trajalo dvakrat dlje, torej 36 dni. Pri napovedi porabljenega prostora smo dejansko porabo podcenili, saj so vse podatkovne baze skupaj zasedale 1050 GB, v primerjavi z 810 GB, ki smo jih napovedali. V tem primeru je bila naša napoved preveč optimistična, vendar zaradi današnjih kapacitet trdih diskov, to ni predstavljalo posebnega problema.

V skladu z našimi zahtevami, ki smo si jih postavili v načrtovanju dela, da

mora biti podatkovna baza enotna in delujoča na enem računalniku, je bilo potrebno pridobljene manjše baze združiti. Tako smo jih najprej izvozili v datoteke in prenesli na en računalnik, kjer smo jih spet vzpostavili. Njihovo združevanje ni bil ravno preprost postopek, saj so tabele med seboj odvisne, zato jih nismo mogli trivialno uvoziti v isto podatkovno bazo. Na ta način bi zaradi primarnih in tujih ključev, ki bi se pri uvozu prepisovali, izgubili soodvisnosti. Za realizacijo združevanja smo zato razvili posebno kratko javansko aplikacijo, ki je kot parameter sprejela ime baze, ki smo jo hoteli združiti v večjo bazo, ter nato po vrsti brala iz nje in vstavljala v novo bazo. Pri tem je popravljala ključe, da so se odvisnosti med tabelami ohranile. Za povezovanje s podatkovno bazo MySQL smo prav tako kot pri odjemalcu v prejšnjem poglavju uporabili posebno knjižnico `mysql-connector`. Pri implementaciji smo pazili tudi na to, da smo zaradi ogromne količine podatkov sproti sproščali vire, ob tem pa še vedno pisali v novo podatkovno bazo s pomočjo tehnik zapisovanja več podatkov hkrati, kar je pospešilo postopek.

Tak način združevanja podatkovnih baz pa se je kljub vsemu naštetemu izkazal za prepočasen. Problem smo rešili tako, da smo za vsako od porazdeljenih baz poiskali maksimalno vrednost primarnega ključa v glavni tabeli in ga prišteli vsem primarnim in tujim ključem vseh tabel vsake naslednje baze. Tako spremenjene podatkovne baze, ki so imele med seboj edinstvene ključe, smo nato direktno uvozili brez dodatnega popravljanja ključev. Zaradi porazdelitve problema in notranjih optimizacij, ki jih pri tem uporablja MySQL, se je ta pristop izkazal za veliko hitrejšega.

Ko so bili vsi podatki končno združeni v enotni podatkovni bazi, smo pričeli s testiranjem le te, tako da smo pognali več testnih poizvedb SQL z različnimi parametri. Te poizvedbe so ustrezale tistim pri končni uporabi podatkovne baze. Nad rezultati časov iskanja in pridobivanja podatkov smo bili precej razočarani. V povprečju je sistem porabil nekaj več kot 10 minut za iskanje novih podatkov in okrog 16 sekund za iskanje primerjav proteinov, po katerih smo predhodno že iskali. Sicer smo v splošnem izboljšali čas pridobivanja željenih podatkov, vendar niti približno na nek, za uporabnika interaktiven čas. Podatkovno bazo smo nato analizirali in prišli do vrednosti, ki so predstavljene v tabeli 4.1. Iz nje je razvidno, da indeksi, ki jih MySQL uporablja za iskanje po podatkih, zasedajo zelo veliko prostora, kar še posebno velja za indeks pri tabeli `result`. Razlog za to je velika selektivnost podatkov v tej tabeli, saj se zelo veliko njenih vrstic navezuje na eno samo vrstico iz tabele `main`, oziroma na eno samo primerjavo dveh proteinov. Ugotovili smo, da je velikost indeksov verjetno glavni razlog za počasno delovanje podatkovne baze, saj bi MySQL deloval mnogo hitreje, če bi bili vsi indeksi shranjeni neposredno v pomnilniku

| Tabela | Velikost [GB] | Velikost indeksa [GB] | Število vrstic |
|---|---|---|---|
| main | 27 | 13,5 | 367.380.000 |
| cluster | 234,5 | 18,8 | 1.250.000.000 |
| result | 788,6 | 241,4 | 13.111.000.000 |

Tabela 4.1: Statistika enotne podatkovne baze. Število vrstic je izračunano samo približno, vendar se le minimalno razlikuje od dejanskega števila.

računalnika. V našem primeru mora sistem pri vsakem iskanju pregledati veliko količino podatkov na disku, kar je zaradi razlik v njunih dostopnih časih precej počasno. Da bi določili kateri del poizvedbe potrebuje največ časa, smo izmerili še čase z okrnjenimi poizvedbami, ki so iskale samo po dveh oziroma po eni sami tabeli. Primeri poizvedb:

- Poizvedba po vseh treh tabelah:

```
SELECT * FROM main M, cluster C, result R WHERE
((M.protein1='1all' AND M.chain1='A') OR
(M.protein2='1all' AND M.chain2='A')) AND
M.main_id=C.main_id AND M.main_id=R.main_id AND
C.cluster_id=R.cluster_id;
```

- Po dveh tabelah (`main` in `cluster`):

```
SELECT * FROM main M, cluster C WHERE
((M.protein1='1all' AND M.chain1='A') OR
(M.protein2='1all' AND M.chain2='A')) AND
M.main_id=C.main_id
```

- Po eni sami tabeli (`main`):

```
SELECT * FROM main M WHERE
(M.protein1='1all' AND M.chain1='A') OR
(M.protein2='1all' AND M.chain2='A')
```

Povprečni časi opisanih poizvedb so zbrani v tabeli 4.2. V njej je mogoče opaziti, da se čas dostopa zelo podaljša, ko dodamo v poizvedbo največjo tabelo. To je tabela `result`, ki ima tudi največji indeks. Dostopni čas je precej dolg

| **Uporabljene tabele** | **Čas dostopa brez medpomnilnika** [s] | **Čas dostopa z medpomnilnikom** [s] |
|---|---|---|
| main | 55 | 0,2 |
| main, cluster | 165 | 0,8 |
| main, cluster, result | 635 | 16 |

Tabela 4.2: Povprečni časi poizvedb po različnih skupinah tabel.

že za dve tabeli in celo za eno samo. Razlog za dolg dostopni čas samo do tabele `main` pa je uporaba `OR` operatorja, zaradi katerega mora MySQL dejansko preiskati dvakratno število vrstic, kot jih tabela v resnici vsebuje, poizvedbe pa tudi ni mogoče dodatno optimizirati. V praksi si običajno želimo doseči odzivni čas pod nekaj sekund, iz naših merjenj časov pa lahko razberemo, da bo za ta problem to skoraj nemogoče. Vendar pa smo enotno podatkovno bazo združevali in testirali na delovnem računalniku, ki ni imel takih sposobnosti kot kasnejši strežniški. Časi na pravem strežniku bi bili seveda v določeni meri krajši, zaradi večje količine pomnilnika (24 GB), ki je na voljo, vendar še vedno ne bi dosegali naših zastavljenih standardov interaktivnosti. Zato smo ta problem želeli rešiti na bolj učinkovit način. Uporabili bi lahko na primer katero izmed NoSQL rešitev, toda brez porazdeljevanja baze po več računalnikih še vedno ne bi mogli zagotoviti njenega hitrejšega delovanja. Vsi večji uporabniki NoSQL svoje podatke namreč običajno shranjujejo porazdeljeno na gruči računalnikov. Ravno tako bi lahko MySQL konfigurirali tako, da deluje porazdeljeno, na gruči računalnikov, vendar to ne bi ustrezalo naši prvotni zahtevi po enotni podatkovni bazi na enem računalniku. Uporabo NoSQL rešitve je oteževala tudi potreba po spreminjanju podatkov v izbrano NoSQL obliko, kar bi bilo v splošnem težje izvedljivo, saj NoSQL sistemi ne uporabljajo poizvedovalnega jezika SQL. Iz istega razloga bi potem morali precej spremeniti tudi sam algoritem ProBiS, da bi znal brati in pisati v take sisteme, pri čemer bi morali še vedno obdržati združljivost za nazaj.

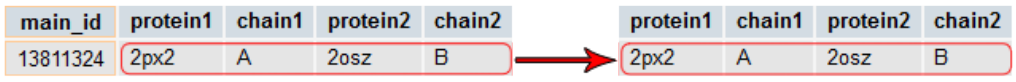
## 4.5 Denormalizacija podatkovne baze ProBiS

Glede na vse opisano smo se zato raje odločili za denormalizacijo enotne podatkovne baze tako, da smo v shemo tabele `main` dodali še dva stolpca s primernimi imeni, za hranjenje tekstovnih tipov podatkov. Nova struktura tabele je prikazana na sliki 4.2. Vse vrednosti iz tabele `cluster`, ki so pripadale

določeni vrstici v tabeli `main`, oziroma primerjavi dveh proteinov, smo shranili v enega od stolpcev in nato isto naredili še z vrednostmi iz tabele `result`. Tako je primerjava dveh proteinov, ki je prej obsegala tipično nekaj sto vrstic v tabeli `result` in manj kot deset vrstic v tabeli `cluster` sedaj zavzela le eno vrstico v novi tabeli `main`. Na ta način smo zreducirali celotno bazo v eno samo tabelo, ki ni bila povezana z ostalimi, zaradi česar je njen čas dosopa bil podoben prej izmerjenemu času dostopa samo do tabele `main`. S tem smo lahko opustili tudi primarne in tuje ključe, ki so prej povezovali tabele, kar nam je dodatno zmanjšalo velikost končne denormalizirane podatkovne baze. Da bi podatke, ki so bili prej ustrezno ločeni, lahko zapisali v isto celico tabele kot dolg tekst, smo jih morali najprej preoblikovati. To smo storili tako, da smo vsako vrstico iz tabel `cluster` in `result` obdali z oklepaji, njene vrednosti atributov pa ločili z vejicami. Vse vrstice, ki so se nanašale na isto vrstico iz tabele `main`, smo ravno tako ločili z vejicami in sestavili skupaj. Postopek je podrobneje predstavljen na slikah 4.3, 4.4 in 4.5.

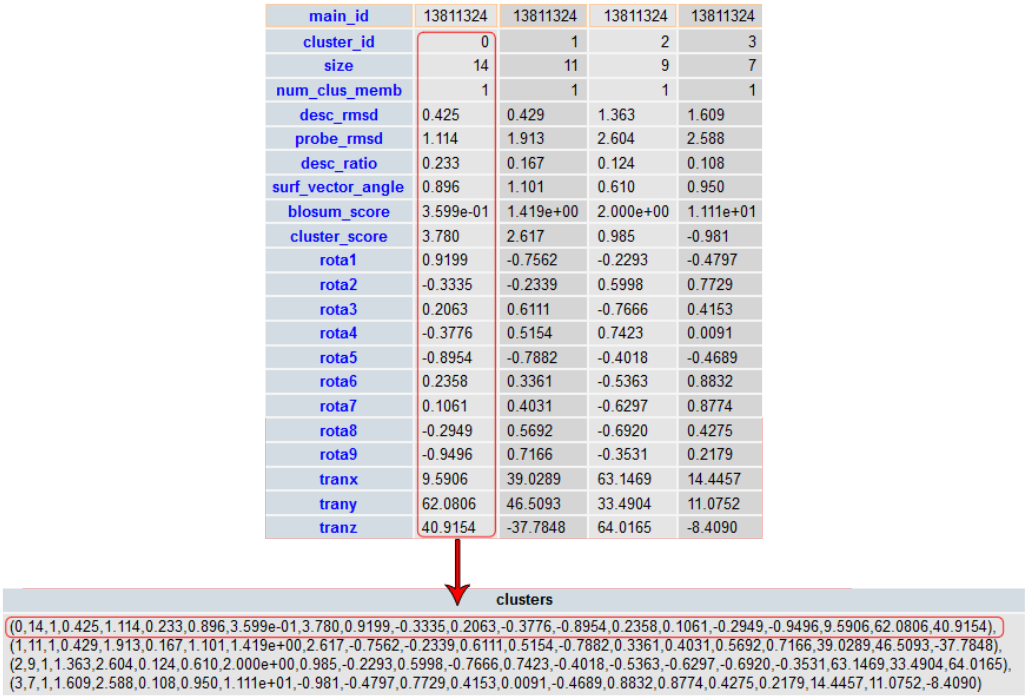
| Field | Type |
|---|---|
| protein1 | char(4) |
| chain1 | char(1) |
| protein2 | char(4) |
| chain2 | char(1) |
| clusters | text |
| results | text |

Slika 4.2: Struktura tabele `main` v denormalizirani podatkovni bazi.




Slika 4.3: Podatki iz tabele `main` se prepišejo v denormalizirano tabelo `main` tako, da se izpusti primarni ključ.

Za postopek denormalizacije podatkovne baze smo napisali kratko javansko aplikacijo, ki je na vsakem od 14 uporabljenih strežnikov izvozila podatkovne



| main_id | 13811324 | 13811324 | 13811324 | 13811324 |
|---|---|---|---|---|
| cluster_id | 0 | 1 | 2 | 3 |
| size | 14 | 11 | 9 | 7 |
| num_clus_memb | 1 | 1 | 1 | 1 |
| desc_rmsd | 0.425 | 0.429 | 1.363 | 1.609 |
| probe_rmsd | 1.114 | 1.913 | 2.604 | 2.588 |
| desc_ratio | 0.233 | 0.167 | 0.124 | 0.108 |
| surf_vector_angle | 0.896 | 1.101 | 0.610 | 0.950 |
| blosum_score | 3.599e-01 | 1.419e+00 | 2.000e+00 | 1.111e+01 |
| cluster_score | 3.780 | 2.617 | 0.985 | -0.981 |
| rota1 | 0.9199 | -0.7562 | -0.2293 | -0.4797 |
| rota2 | -0.3335 | -0.2339 | 0.5998 | 0.7729 |
| rota3 | 0.2063 | 0.6111 | -0.7666 | 0.4153 |
| rota4 | -0.3776 | 0.5154 | 0.7423 | 0.0091 |
| rota5 | -0.8954 | -0.7882 | -0.4018 | -0.4689 |
| rota6 | 0.2358 | 0.3361 | -0.5363 | 0.8832 |
| rota7 | 0.1061 | 0.4031 | -0.6297 | 0.8774 |
| rota8 | -0.2949 | 0.5692 | -0.6920 | 0.4275 |
| rota9 | -0.9496 | 0.7166 | -0.3531 | 0.2179 |
| tranx | 9.5906 | 39.0289 | 63.1469 | 14.4457 |
| trany | 62.0806 | 46.5093 | 33.4904 | 11.0752 |
| tranz | 40.9154 | -37.7848 | 64.0165 | -8.4090 |



Slika 4.4: Podatki iz tabele `cluster` se spremenijo v tekst tako, da se med vsakim izmed njih vstavi vejica, celotna vrstica pa postavi med oklepaje. Različne vrstice med sabo prav tako loči vejica.

baze v datoteke, te datoteke prebrala, nato pa primerno obdelala vsebujoče podatke in vse skupaj zapisala v novo datoteko, primerno za direkten uvoz v MySQL. Pri tem smo pazili, da primerjav med proteinoma, ki niso imele nobenih rezultatov, nismo vključevali v končno datoteko. V nasprotnem primeru bi imeli v denormalizirani podatkovni bazi vrstice brez vrednosti v novo zgrajenih stolpcih, kar ne bi imelo smisla. Ker so bile izvožene datoteke po velikosti podobne podatkovni bazi, jih zaradi omejitev računalniškega pomnilnika nismo mogli naložiti direktno vanj, ampak smo jih morali brati po vrsti in hkrati paziti na sprotno zapiranje virov po branju ene vrstice iz datoteke. To smo lahko storili le tako, da smo datoteke izvozili na način, da so bili podatki urejeni po naraščajočih vrednostih primarnih in tujih ključev, kar nam je omogočilo minimalno porabo pomnilnika, saj ni bilo potrebno dodatno iskanje po podatkih. Na novo pridobljene spremenjene datoteke smo tako uvozili v podatkovno bazo in jo testirali. Velikost podatkovne baze je znašala nekaj manj kot 600 GB, indeks pa je zasedal 14,5 GB. Časi poizvedb so bili po-

| main_id | flx | chain1 | resn1 | resi1 | chain2 | resn2 | resi2 | cluster_id |
|---|---|---|---|---|---|---|---|---|
| 13811324 | N | A | S | 197 | B | N | 355 | 0 |
| 13811324 | N | A | E | 193 | B | E | 351 | 0 |
| 13811324 | N | A | E | 196 | B | E | 354 | 0 |
| 13811324 | N | A | K | 194 | B | E | 352 | 0 |
| 13811324 | N | A | K | 190 | B | Q | 348 | 0 |
| 13811324 | N | A | I | 192 | B | I | 350 | 0 |
| 13811324 | N | A | R | 63 | B | R | 347 | 1 |
| 13811324 | N | A | E | 67 | B | E | 351 | 1 |
| 13811324 | N | A | K | 61 | B | Q | 345 | 1 |
| 13811324 | N | A | V | 66 | B | I | 350 | 1 |
| 13811324 | N | A | T | 7 | B | N | 355 | 2 |
| 13811324 | N | A | Q | 11 | B | Q | 360 | 2 |
| 13811324 | N | A | L | 8 | B | L | 357 | 2 |
| 13811324 | N | A | I | 241 | B | I | 366 | 2 |
| 13811324 | N | A | S | 128 | B | A | 374 | 3 |
| 13811324 | N | A | D | 131 | B | D | 370 | 3 |
| 13811324 | N | A | V | 132 | B | I | 366 | 3 |
| 13811324 | N | A | V | 130 | B | L | 371 | 3 |

| results |
|---|
| (N,A,S,197,B,N,355,0), |
| (N,A,E,193,B,E,351,0), |
| (N,A,E,196,B,E,354,0), |
| (N,A,K,194,B,E,352,0), |
| (N,A,K,190,B,Q,348,0), |
| (N,A,I,192,B,I,350,0), |
| (N,A,R,63,B,R,347,1), |
| (N,A,E,67,B,E,351,1), |
| (N,A,K,61,B,Q,345,1), |
| (N,A,V,66,B,I,350,1), |
| (N,A,T,7,B,N,355,2), |
| (N,A,Q,11,B,Q,360,2), |
| (N,A,L,8,B,L,357,2), |
| (N,A,I,241,B,I,366,2), |
| (N,A,S,128,B,A,374,3), |
| (N,A,D,131,B,D,370,3), |
| (N,A,V,132,B,I,366,3), |
| (N,A,V,130,B,L,371,3) |

Slika 4.5: Pri podatkih iz tabele `result` velja isto pravilo preoblikovanja v tekst, kot pri podatkih iz tabele `cluster`.

dobni pričakovanemu času dostopa samo do stare tabele `main`, saj je testna poizvedba SQL za pridobitev podatkov potrebovala približno 1 minuto brez uporabe medpomnilnika, ob uporabi pa manj kot 1 sekundo.

Ker smo z denormalizacijo v določeni meri porušili podatkovno fleksibilnost in so bili podatki po novem v drugačni obliki, smo morali algoritem ProBiS dopolniti tako, da smo mu dodali nove funkcije, s katerimi mu je omogočeno branje in pisanje v denormalizirano podatkovno bazo. Funkcija za pisanje tako kar ob sprotnem računanju, glede na podana pravila, sestavlja rezultate v primeren tekst za shranjevanje. Funkcija za branje pa zna prebrati tekst iz podatkovne baze in iz njega na podlagi naštetih pravil razločiti posamezne podatke, po katerih lahko tudi filtrira. Realne poizvedbe SQL so prej filtriranje opravljale kar na nivoju podatkovne baze, kar jih je dodatno zakompliciralo in podaljšalo čas izvajanja same poizvedbe. Filtriranje na programskem nivoju je hitrejša in primernejša rešitev. Dodatno programiranje algoritma ProBiS tako ni bilo problematično, saj smo na pretežno enostaven način spremenili stare funkcije in jih vključili kot nove. Izguba podatkovne fleksibilnosti ravno tako ne pomeni večjega problema v našem sistemu, saj se podatki po tem, ko so enkrat ustvarjeni, nikoli več ne spreminjajo.

## 4.6 Izdelava vizualnega spletnega gradnika za pridobivanje podatkov iz podatkovne baze ProBiS

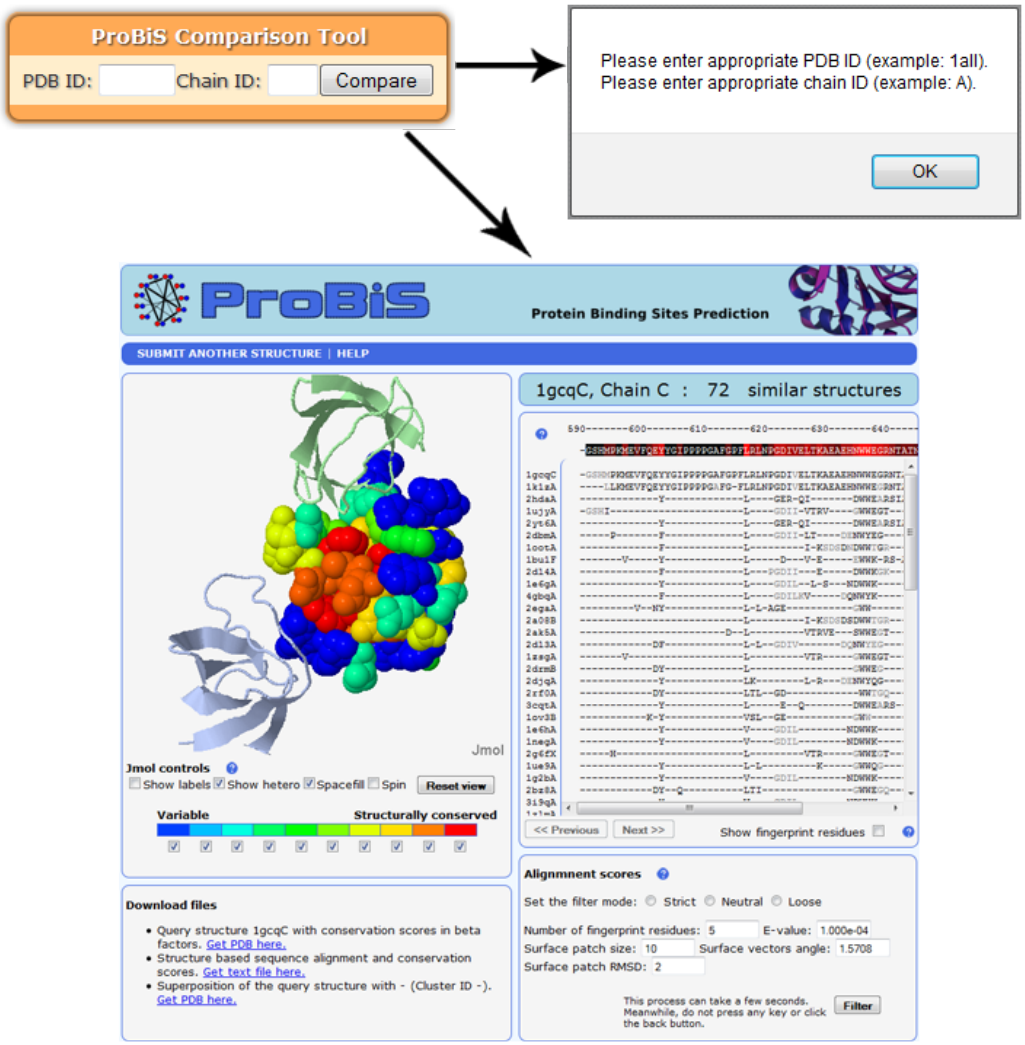
Poleg spremembe podatkovne baze, smo se odločili dodatno implementirati tudi vizualni gradnik za iskanje po njej. Gradnik je moral biti tako fleksibilen, da bi ga lahko preprosto vključili v katerokoli spletno stran. Ta fleksibilnost je bila pomembna predvsem zato, da smo s tem omogočili ostalim institucijam, ki se ukvarjajo s primerjavami proteinov, da vključijo gradnik v svoje spletne strani in si s tem olajšajo dostop do naše storitve. Zaradi tega je napisan v jeziku JavaScript in prosto dostopen prek našega spletnega strežnika. Na določeno spletno stran se ga vključi preprosto tako, da v datoteki HTML na mestu, kjer ga želimo imeti, dodamo sledečo kodo:

```
<script type="text/javascript"
src="http://probis.cmm.ki.si/probisWidget.js"></script>
```

Skripta ob vključitvi v dokument HTML doda v dokumentni objektni model nekaj svojih elementov: okvir, okence za vpis imena proteina, okence za vpis imena verige in gumb za pošiljanje poizvedbe. Hkrati vsebuje tudi funkcije za preverjanje pravilnosti vpisanih vrednosti in če so vrednosti napačne ali pa sploh niso vpisane, uporabniku prikaže sporočilo z obvestilom o napaki. V primeru, da vključimo v HTML samo skripto, bo njen izgled zelo osnoven in primitiven. Če želimo, da gradnik izgleda lepše in bolj uporabniku prijazno, moramo v glavo dokumenta HTML vključiti še sledečo vrstico, ki gradniku določi vizualni stil CSS:

```
<link rel="stylesheet" type="text/css"
href="http://probis.cmm.ki.si/probisWidget.css" />
```

Če barvna shema ne ustreza barvam spletne strani, na katero bi uporabnik rad vključil naš gradnik, lahko datoteko CSS preprosto prenese na strežnik, kjer se nahaja njegova/njena spletna stran. Nato le še spremeni parametre v datoteki CSS glede na svoje želje in popravi pot do nje v glavi datoteke HTML. Delovanje skripte poteka na tak način: Ob pritisku na tipko “Compare” se koda PDB in koda verige proteina pošlje na naš strežnik, kjer skripta PHP požene program ProBiS, ki sproži poizvedbo za ta protein v denormalizirani podatkovni bazi ProBiS. Iz dobljenih podatkov naredi spletno stran z rezultati primerjave, na katero je uporabnik nato preusmerjen. Postopek delovanja je prikazan na sliki 4.6.




Slika 4.6: Izgled in delovanje vizualnega gradnika. Ob napačnih vhodnih podatkih se uporabniku prikaže sporočilo o napaki. Ob pravilno vpisanih podatkih, se zahteva posreduje na strežnik, kjer se preišče podatkovna baza ProBiS in ustvari ter prikaže stran z rezultati.

# Poglavje 5

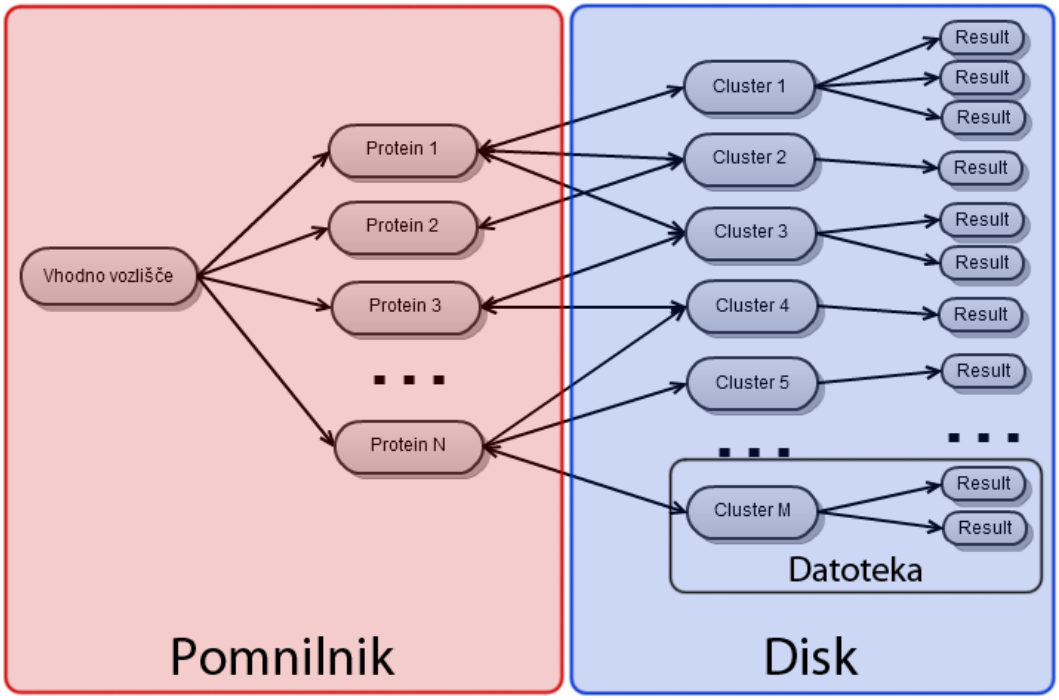
# Sklepne ugotovitve in ideje za nadaljnje delo

V delu je predstavljen postopek porazdeljenega računanja primerjav proteinov in vzpostavljanja enotne podatkovne baze ProBiS za pridobljene podatke. Kot je razvidno iz prejšnjih poglavij, izvajanje računanja ni povzročalo nikakršnih težav. Največji problemi so se pojavljali pri vzpostavitvi podatkovne baze.

Ena od glavnih omejitev je bila zahteva, da se podatkovna baza vzpostavi na enem računalniku oziroma enem disku. Ta omejitev se je izkazala za preveč strogo, saj je bila količina podatkov prevelika, da bi bilo mogoče uspešno in hitro iskati po njih. Sistem bi delal veliko bolje, če bi bila podatkovna baza porazdeljena na več računalnikih, vsak od teh pa bi po prejemu zahtevka preiskal le svoj del podatkov. Na tak način bi se v odvisnosti od števila računalnikov, na katere bi bazo porazdelili, v veliki meri skrajšal čas pridobivanja podatkov. Hkrati bi sicer morali poskrbeti za vmesno stopnjo združevanja vrnjenih podatkov, toda večina sistemov za upravljanje s podatkovnimi bazami, vključno z MySQL, to že samodejno omogoča.

Namesto sistema za upravljanje s podatkovnimi bazami MySQL bi lahko uporabili katerega od ostalih podobnih sistemov, vendar bi se najverjetneje vsi, ki uporabljajo relacijski podatkovni model, izkazali približno enako. Lahko bi uporabili tudi katero izmed novejših rešitev pestre zbirke sistemov NoSQL. Naši podatki so sicer bolj toge narave in ne potrebujejo fleksibilnih tabelaričnih shem, kljub temu pa bi bilo mogoče pridobiti na hitrosti predvsem zaradi uporabe drugačnih načinov shranjevanja podatkov. Predvidevamo, da bi bil za ta namen najbolj uporaben podatkovni model v obliki grafa. Pri tem bi vsako tabelo iz prvotne baze predstavili kot svojo vrsto vozlišča. Tako bi vsak protein in vsaka vrstica iz tabel `cluster` in `result` predstavljala svoje

vozlišče. Vozlišče tipa "protein" bi vsebovalo kodo PDB proteina in kodo verige. Vozlišče tipa "cluster" bi vsebovalo vse vrednosti atributov iz določene vrstice iz tabele `cluster`. Isto bi veljalo za vozlišče tipa "result", le da bi to vsebovalo vse vrednosti iz določene vrstice iz tabele `result`. Vsako "cluster" vozlišče bi definiralo eno primerjavo dveh proteinov, zato bi imelo vedno točno dve dvosmerni povezavi na dve proteinski vozlišči, hkrati pa različno število enosmernih povezav na pripadajoča "result" vozlišča. Povratne povezave bi bile potrebne zato, ker poleg informacije o sami primerjavi vedno potrebujemo tudi informacijo o paru primerjanih proteinov. Vsako vozlišče tipa "protein" bi imelo povezave na vsa "cluster" vozlišča, ki so rezultat primerjave proteina z nekim drugim. Vhodno vozlišče bi v tem primeru predstavljalo le vmesnik za dostop do podatkov, lahko bi bil to preprost seznam ali pa razpršitvena tabela. Vsekakor bi vsebovalo povezave na vse proteine (v našem primeru 27.000 povezav). Primer takega grafa je predstavljen na sliki 5.1.




Slika 5.1: Primer grafa predlagane rešitve.

Ker so poizvedbe v našem primeru vedno istega tipa (definirane so z imenom proteina in verige), bi se lahko na tak način hitro sprehodili po grafu in dodali, prebrali ali pobrisali pripadajoče rezultate vsake primerjave. Za tak

pristop ne bi bilo potrebno uporabiti posebne podatkovne baze NoSQL, ampak bi ga lahko neposredno implementirali v programu ProBiS. Vozlišča bi predstavili z razredi, povezave med njimi pa s kazalci. Direktno v pomnilniku bi hranili samo poti do posameznih primerjav, podatke pa bi lahko imeli shranjene na primer v datotekah XML. Te bi po prejemu zahtevka za prikaz podatkov prebrali in v primerni obliki vrnili uporabniku. Taka implementacija rešitve bi bila zahtevnejša, vendar bi se jo zaradi pospešitve hitrosti poizvedb morda v prihodnosti splačalo uresničiti.

Denormalizirana podatkovna baza je vseeno zelo učinkovita rešitev. V primerjavi s prejšnjo skupno podatkovno bazo je v času dostopa do podatkov hitrejša od slednje za faktor nekaj več kot 10, medtem ko je v primerjavi s prvotno implementacijo sistema ProBiS, ki ni uporabljal podatkovne baze, hitrejša za faktor 30 in več, kar pomeni da je bil naš namen izboljšanja interaktivnosti ob podanih omejitvah v veliki meri uspešen.

# Literatura


[1] G. R. Andrews, *Foundations of multithreaded, parallel, and distributed programming*, Reading, Mass.: Addison-Wesley, 2000, pogl. 1.

[2] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, P.E. Bourne, "The Protein Data Bank," *Nucleic Acids Research*, št. 28, str. 235-242, 2000.

[3] J. Farrell, *Java Programming*, International Edition: Course Technology: Cengage Learning, 2012, pogl. 1.

[4] J. Konc, D. Janežič, "ProBiS algorithm for detection of structurally similar protein binding sites by local structural alignment," *CBioinformatics*, št. 26, zv. 9, str. 1160–1168, 2010.

[5] J. Konc, D. Janežič, "ProBiS: a web server for detection of structurally similar protein binding sites," *Nucleic Acids Research*, št. 38, str. 436–440, 2010.

[6] J. Konc, D. Janežič, "Z računalnikom do vpogleda v evolucijo proteinskih vezavnih mest," *Kemija v šoli in družbi*, letnik 22, št. 3, str. 4-9, 2010.

[7] A. D. Kshemkalyani, M. Singhal, *Distributed Computing*, Cambridge: Cambridge University Press, 2008, pogl. 1.

[8] T. Mohorič, *Podatkovne Baze*, Ljubljana: BI-TIM d.o.o., 2002, pogl. 2, 4, 5, 6.

[9] R. W. Sebesta, *Programming the World Wide Web 2009*, New Jersey: Upper Sadle River: Pearson, 2010, pogl. 2, 3, 4, 9, 10.

[10] L. Ullman, *MySQL*, Berkeley: Peachpit Press, 2006, pogl. 1, 3.

[11] Bjarne Stroustrup's FAQ. Dostopno na: http://www2.research.att.com/∼bs/bs_faq.html

[12] C++ Applications. Dostopno na:
http://www2.research.att.com/∼bs/applications.html

[13] Introduction to programming languages. Dostopno na:
http://www.upf.edu/materials/bib/docs/3371/12463/aaby.pdf

[14] PDB ID(s). Dostopno na:
http://www.pdb.org/pdb/staticHelp.do?p=help/advancedsearch/pdbIDs.html

[15] Protein Data Bank Contents Guide: Atomic Coordinate Entry Format Description. Dostopno na:
ftp://ftp.wwpdb.org/pub/pdb/doc/format_descriptions/Format_v33_A4.pdf

[16] Up Close and Personal with NoSQL. Dostopno na:
http://www.linuxforu.com/developers/up-close-and-personal-with-nosql/