Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Svalina

Razvoj podatkovne baze za poskuse z DNA-mikromrežami

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: prof. dr. Blaž Zupan
Somentor: prof. dr. Kristina Gruden

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

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: 00366/2013
Datum: 04.02.2013

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

Kandidat: **MIHA SVALINA**

Naslov: **RAZVOJ PODATKOVNE BAZE ZA POSKUSE Z DNA-MIKROMREŽAMI**
DEVELOPMENT OF A DNA MICROARRAY DATA REPOSITORY

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomski nalogi razvijte podatkovno bazo in pripadajočo spletno aplikacijo za delo s podatki o genskih izrazih, ki so pridobljeni s tehnologijo DNA-mikromrež. Podatkovna baza naj omogoča vpis opisnih podatkov o eksperimentu, prenos podatkov na strežnik ter pregledovanje vseh podatkov v podatkovni bazi. Izdelajte tako strežniški del aplikacije kot tudi ustrezno aplikacijo na odjemalcu. Aplikacija naj omogoča tudi osnovno pregledovanje in vizualizacijo podatkov o genskih izrazih.

Mentor:

prof. dr. Blaž Zupan

Dekan:

prof. dr. Nikolaj Zimic

Somentor:

prof. dr. Kristina Gruden

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Miha Svalina, z vpisno številko **63070161**, sem avtor diplomskega dela z naslovom:

Razvoj podatkovne baze za poskuse z DNA-mikromrežami

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Blaža Zupana in somentorstvom prof. dr. Kristine Gruden,
- 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, 21. maj 2013 Podpis avtorja:

Za strokovno pomoč in nasvete pri diplomskem delu se zahvaljujem mentorju prof. dr. Blažu Zupanu.

Somentorici prof. dr. Kristini Gruden in dr. Ani Rotter se zahvaljujem za odgovore na vsa moja vprašanja o bioloških poskusih.

Zahvala gre tudi staršem, ki so me tekom študija moralno in finančno podpirali. Še posebno se zahvaljujem Tatjani za podporo, potrpežljivost in popravljanje slovničnih napak.

Kazalo

Povzetek

Na Nacionalnem inštitutu za biologijo (v nadaljevanju NIB) pri poskusih uporabljajo tehnologijo DNA-mikromrež. Podatke o poskusih shranjujejo v obliki datotek. Vsak raziskovalec pri tem uporablja svojo obliko datotek. Da bi vsakomur na NIB-u omogočili iskanje po vseh podatkih poskusov, je nastala potreba po centralizirani aplikaciji z možnostjo upravljanja s podatki. V ta namen smo razvili aplikacijo Fitobase. Aplikacija omogoča opis in urejanje poskusov z DNA-mikromrežami ter nalaganje in iskanje po vseh definiranih datotekah poskusov. Omogoča tudi grafične prikaze izraženosti genov, kar uporabniku dodatno pomaga pri analizi genov. S pomočjo genskega izražanja v različnih bioloških kontekstih lahko biologom pomagamo razumeti obnašanje gena pri različnih pogojih in tako natančneje določiti njegovo funkcijo.

Abstract

Researchers at National Institute of Biology (NIB) use DNA microarray technology in their experimental work. Experimental data is currently saved in the form of files and just about every researcher at the Institute uses its own file format. To support searching through all experimental data by everyone at NIB, there was a need to develop a centralised application for DNA microarray data management. We have addressed this task by developing an application called Fitobase. The application allows user to describe the parameters of the experiment, upload the data, and search through the data sets stored in Fitobase. It also provides graphical visualisation of gene expression, supporting a simple means of data analysis. With analysis and comparison of gene expressions across different biological contexts, Fitobase can help biologists to understand the behaviour of the model organisms at different conditions and to hypothesise on gene function.

Poglavje 1

Uvod

Na Nacionalnem inštitutu za biologijo (NIB) si že dolgo želijo podatkovno bazo, ki bi shranjevala poskuse s področja transkriptomike. Opravljajo poskuse z DNA-mikromrežami (angl. *microarray experiment*) in qPCR (quantitative real time polymerase chain reaction). Največkrat ima raziskovalec poskus shranjen lokalno na svojem računalniku. Na NIB-u si želijo, da bi bili poskusi organizirani v skupno središče, do katerega bi lahko dostopali vsi raziskovalci. Tu bi si lahko ogledali že opravljene poskuse in rezultate posameznega poskusa. Vse poskuse so tako organizirali na skupnem datotečnem strežniku (angl. *file server*), ki shranjuje poskuse, opisane z množico datotek. Največkrat raziskovalec shrani pomanjkljiv poskus. Manjkajo ključne datoteke, ki so potrebne za minimalen opis poskusa. Manjka na primer datoteka z neobdelanimi rezultati poskusa, kot jo vrne stroj, s katerim smo izvedli poskus, ali datoteka s podatki o vzorcih (angl. *sample name*) o uporabljenih genih ali pa datoteka s pridobljenimi rezultati statističnih analiz, ki primerja okužbo organizma z delovanjem virusa po na primer 12-ih urah v primerjavi z okužbo po 48-ih urah. Raziskovalec uporablja tudi svojo obliko datotek. Datoteki s podatki o genih manjkajo opisi genov ali pa se imena genov ne ujemajo s tistimi v datoteki z rezultati. Datoteka z rezultati lahko vsebuje tudi poljubne stolpce. Raziskovalcu, ki je razvil poskus, so se zdeli pomembni, za analizo pa niso ključnega pomena. Ob ustreznem minimal-

nem naboru datotek poskusa posledično omogočimo iskanje po datotekah. Da bi poenostavili iskanje, je zaželjeno, da vpeljemo enotno obliko datotek. Vsaki datoteki poskusa definiramo svojo obliko. Oblika predpisuje dovoljen tip datoteke, imena obveznih stolpcev ter njihov vrstni red in dovoljene vrednosti. Z uvedbo enotne oblike datotek omogočimo konsistentno in hitrejše branje vsebine datotek. Hkrati novim raziskovalcem olajšamo raziskovanje poskusov.

Pregledali smo obstoječe rešitve podatkovnih baz, ki shranjujejo poskuse s področja transkriptomike, in od njih odvzeli, kar je najbolj ustrezalo našim potrebam za razvoj aplikacije. Razvili smo spletno aplikacijo *Fitobase*, ki omogoča opis in urejanje poskusov z DNA-mikromrežo ter nalaganje in iskanje po datotekah poskusa. Definirali smo minimalen nabor datotek, potrebnih za opis poskusa z DNA-mikromrežo. Za vsako datoteko smo definirali tudi obliko vsebine datoteke (angl. *data format*). Dodatno smo obogatili funkcionalnost aplikacije s prikazom izraženosti genov (angl. *gene expression*) v obliki stolpičnih grafov (angl. *bar chart*). Tako smo omogočili biologom ne le shranjevanje in iskanje po poskusih z DNA-mikromrežo, temveč tudi analizo posameznih genov.

Poglavje 2

Uporabljene tehnologije in orodja

Za učinkovito izgradnjo podatkovne baze smo pri delu uporabili veliko različnih orodij. Poskusili smo uporabiti čim manj orodij, saj s tem zmanjšamo število orodij, ki jih aplikacija potrebuje za delovanje. Posamezno orodje smo uporabili v skladu z njegovim namenom uporabe.

2.1 SEEK

SEEK je odprtokodna spletna platforma, namenjena iskanju, deljenju in izmenjavi podatkov, modelov in procesov v sistemski biologiji. SEEK (`http://www.sysmo-db.org/seek`) je bil prvotno razvit v okviru projekta SysMO-DB (`http://sysmo-db.org`) kot podpora konzorciju SysMO. Konzorcij SysMO je evropska raziskovalna iniciativa za sistemsko biologijo mikroorganizmov.

SEEK v zadnjem času pridobiva priljubljenost tudi pri drugih konzorcijih po Evropi (Virtual Liver, EviMalar, Unicellsys). Razvija se s tesnim sodelovanjem uporabnikov. Uporabniki razvijejo svojo funkcionalnost, ki lahko kasneje preide v osnovno funkcionalnost naslednjih različic.

SEEK temelji na modelu JERM (Just Enough Results Model), ki govori o minimalni količini informacij, potrebnih za opis vsebine SEEK, in relacij med

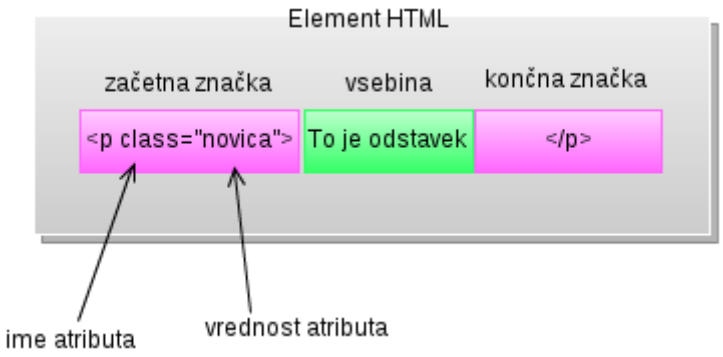
njimi. JERM za vsak tip podatka, kot so poskus z mikromrežo, opis poskusa s področja metabolomike in standardni protokoli (SOP), predstavlja minimalno podatkovno shemo, s katero se konzorcij SysMO strinja glede deljenja in izmenjave. To nas pripelje do predlog JERM (JERM templates). Projekt SysMO-DB uporablja JERM, kadarkoli je možno, s čimer omogoči hiter izvoz in objavo podatkov konzorcija SysMO v javno dostopnih repozitorijih.

Koda SEEK je dostopna na URL naslovu `http://code.google.com/p/sysmo-db`. Demonstracijo si lahko ogledamo na URL naslovu `https://demo.sysmo-db.org`.

NIB je v letu 2012 gostoval članico ekipe projekta SysMO-DB, Olgo Krebs (`https://seek.sysmo-db.org/people/136`). Pri njeni predstavitvi platforme SEEK (`http://www.sysmo-db.org/node/71`) smo ugotovili, da mnoge rešitve, ki so jih v okviru njihovega projekta že razvili, ustrezajo našim. Tako smo podedovali in nadgradili predloge JERM (`https://seek.sysmo-db.org/help/templates`) za področje transkriptomike.

2.2 HTML

HTML (HyperText Markup Language) je označevalni jezik za izdelavo spletnih vsebin. Dokument HTML je predstavljen z elementi HTML, s katerimi oblikujemo vsebino dokumenta. Element je sestavljen iz začetne značke, vsebine elementa in končne značke, kot prikazuje slika 2.1 za element odstavek (`p`). Začetna značka ima obliko <`ime elementa`>, končna pa </`ime elementa`>. Elementu lahko dodamo atribute (naprimer `class`, `id`, `width`), ki se dodajo v začetno značko. Z elementi HTML gradimo strukturo in obliko celotnega dokumenta.



Slika 2.1: Struktura elementa HTML.

2.3 PHP

PHP (Hypertext Preprocessor) je skriptni jezik, ki se uporablja za razvoj dinamičnih spletnih strani. Cilj jezika je omogočiti razvijalcem hitro pisanje dinamičnih spletnih vsebin. Veliko sintakse si je jezik sposodil od programskih jezikov C, Jave in Perl. Jezika se zelo hitro naučimo, posledica je hiter razvoj aplikacije [6]. Avtor jezika je Rasmus Lerdorf, danski programer. Kasneje, ko je jezik začela uporabljati množica programerjev, se je odločil in omogočil dostop do spreminjanja kode jezika vsem, ki so sodelovali. To je še dodatno pospešilo njegovo uporabo in razširjenost. Kmalu je število razvijalcev strmo naraslo. Posledično je imel vsak svoje ideje za nadaljnji razvoj (naprimer nove funkcije, podatkovne strukture in kontrolni stavki). Izoblikoval se je ožji krog razvijalcev, ki je odločal o tem, katera funkcionalnost se doda v jezik in katera ne. Če hočemo danes dodati svojo funkcionalnost v uradno različico PHP-ja, je prvi korak opis funkcionalnosti v RFC-ju (`https://blogs.oracle.com/opal/entry/the_mysterious_php_rfc_process`). Nato sledi glasovanje razvijalcev projekta PHP in predstavnikov skupnosti PHP, ki so jih izbrali razvijalci projekta PHP (`https://wiki.php.net/rfc/voting`).

Pri programiranju veliko razvijalcev za določeno komponento aplikacije uporabi že obstoječe knjižnice, ki so jih napisali drugi razvijalci. Izkaže se, da so najbolj uporabljene knjižnice tudi najbolj testirane in posledično najbolj zanesljive. Pogosto se zgodi, da nekomu, ki je uporabil že obstoječo knjižnico, njen nabor funkcij ne zadošča. Njegov problem reši delno. Razvijalec se odloči, da bo knjižnico razširil in dodal potrebno manjkajočo funkcionalnost. Različne knjižnice pišejo različni razvijalci, vsak s svojim stilom pisanja kode. Stil pisanja kode definiramo z množico lastnosti, kot so zamik vrstice, imenovanje razredov, spremenljivk, konstant, funkcij in metod, kodni nabor, v katerem je izvorna datoteka shranjena, oblika kontrolnih stavkov (`if, while, else, for, foreach, try, catch`) ter maksimalna dolžina ene vrstice. Prvi pogled na stil pisanja kode knjižnice nam veliko pove o sami strukturiranosti in urejenosti kode. Da bi se vsi razvijalci držali enotnega stila pisanja kode, se je ustanovila skupina *Framework Interop Group*, prvotno *PHP Standards*

Group. Skupina je definirala priporočila za pisanje kode, opisana v dokumentih PSR-0, PSR-1, PSR-2 in PSR-3 (`http://php-fig.org/faq`). Skupnost PHP jih šele v zadnjem času sprejema in upošteva. Tako omogočimo standardizacijo in poskušamo doseči visoko stopnjo kvalitete izvorne kode. Poleg tega branje izvorne kode različnih knjižnic postane hitrejše in posledično omogoči hitrejši razvoj.

Ker je PHP zasnovan s ciljem hitrega razvoja spletnih vsebin, je praksa pokazala, da to mnogokrat nasprotuje učinkovitemu vzdrževanje kode ali kasnejši nadgradnji aplikacije. To se največkrat pokaže, ko programer logiko aplikacije vmeša v kodo HTML. Menim, da v kodo HTML sodi samo izpis rezultatov v obliki <?=$out?>. V primeru, da imamo enak del logike aplikacije napisan v večih datotekah, je potrebno ob spremembi logike aplikacije spremeniti kodo v vseh datotekah. Da bi rešili problem učinkovitega vzdrževanja kode, so se razvila različna ogrodja za razvoj spletnih vsebin s programskim jezikom PHP. Trenutno najbolj uporabljena so Zend, CakePHP, Symfony in Codeigniter. Ta ogrodja vsebujejo vse potrebne mehanizme za odpravo te težave, če jih razvijalec ustrezno uporabi. Praktično lahko govorimo o ogrodjih MVC (Model-Pogled-Kontroler). MVC je vzorec, ki zmanjšuje čas programiranja in vzdrževanja kode [2]. Vzorec MVC, kot ga uporablja ogrodje CakePHP, skupaj s komponento razpošiljevalec, je prikazan na sliki 2.2 [8].

Slika 2.2: Uporaba vzorca model-pogled-kontroler skupaj s komponento razpošiljevalec, kot ga uporablja ogrodje CakePHP.

2.3.1 Laravel

Laravel je ogrodje PHP za spletne obrtnike (kot navaja uradni moto ogrodja). Njegov avtor, Taylor Otwell, je razvil ogrodje aprila 2011. Po tem, ko je raziskoval različna ogrodja PHP, je iskal nekaj, kar bi bilo v koraku s trenutno različico PHP-ja (različica 5.3). Iskal je dobro dokumentacijo, ki uporabnika hitro pripelje do razvoja njegove aplikacije. Razvil je intuitivno in preprosto, a vendar prilagodljivo ogrodje. Namenjeno je tistim, ki se prvič srečujejo z razvojem spletnih vsebin in razvijalcem z izkušnjami. Za namestitev potrebujemo osnovno različico PHP 5.3 in razširitev PHP MCrypt. Kot večina ogrodij deluje po vzorcu MVC (Model-View-Controller). Vgrajen ima tudi lasten ORM (object-relational mapping), s katerim je delo z relacijsko podatkovno bazo konsistentno, hitrejše in varnejše (`http://laravel.com/docs/database/eloquent`). Pri razvoju so zelo uporabni kontrolerji RESTful, saj pripomorejo k večji preglednosti in strukturiranosti kode. To so opcijski kontrolerji, s katerimi lahko ločimo zahteve HTTP GET in POST. Poglejmo primer uporabe za uporabniški prijavni kontroler, prikazan spodaj. Kontroler vsebuje metodo *getLogin()*, s katero prikažemo formo HTML, in metodo *postLogin()*, s katero naredimo validacijo forme. Ob neuspešni validaciji forme uporabnika ponovno preusmerimo na izpolnitev forme.

```
class User_Controller extends Base_Controller
{
  public $restful = true;

  # prikazemo formo HTML
  #
  public function get_login()
  {
    $this->layout->content = View::make('user.profile);
  }

  # validacija podatkov
  # Ob napaki uporabnika preusmerimo na ponoven vpis forme
  #
  public function post_login()
  {
    if (!Auth::attempt(Input::only(['username', 'password'])))
      return Redirect::to_action('user@login')
        ->with('login_errors', true)->with_input();

    return Redirect::to_action('user@panel');
  }
}
```

2.4 Python

Python je skriptni, objektno usmerjen, splošno namenski programski jezik. Uveljavlja čisto in enotno sintakso. Ko govorimo o spremenljivkah v Pythonu, govorimo o objektih. Python nima spremenljivk, ima le objekte [1]. V Pythonu je vse predstavljeno z objektom (seznami, slovarji, datoteka, niz, število, funkcija in modul). Ko v Pythonu izvedemo spodnjo kodo, ustvarimo objekt tipa `int`, z imenom `a`.

```
>>> a = 2
```

Kot programski jezik Java Python kodo prevede v vmesno kodo in zažene v navideznem stroju. Vmesna koda generira datoteko s končnico *pyc* (angl. *compiled python bytecode*). Vmesno kodo lahko zaženemo na različnih operacijskih sistemih, in sicer z nameščenim virtualnim strojem za dani operacijski sistem.

Python se uporablja za reševanje kateregakoli področja programiranja: za programiranje grafičnih uporabniških vmesnikov, spletnih storitev, za implementacijo strežnika s podporo različnim protokolom, obdelavo slik, zvoka in video vsebin, v kombinaciji s potrebnimi knjižnicami. Ko del kode ni dovolj učinkovit za naše potrebe, kritičen del kode razvijemo v programskem jeziku C, s katerim se Python povezuje.

Primeren je za pisanje različnih skript, ki tečejo v družini operacijskih sistemov UNIX. Zelo se je razširil za pisanje sistemskih skript v distribucijah Linux (SUSE, Debian, Fedora, Ubuntu, Gentoo, ...).

2.5 Loxun

Python za delo z dokumenti tipa XML (Extensible Markup Language) podpira standarda DOM in SAX. Prvi hrani dokument v drevesni strukturi, ki je v celoti shranjen v glavnem pomnilniku. Pri velikih dokumentih lahko tu nastane težava, saj nam dokument zasede toliko glavnega pomnilnika, kot je njegova velikost. V ta namen je bil razvit SAX [3]. Uporabljamo ga tako, da podamo funkcije, ki se prožijo ob posameznih dogodkih (začetek in konec značke) [1].

Loxun je modul za Python, s katerim ustvarjamo dokumente XML. Dokumentu poljubno dodamo imenske prostore (angl. *namespace*). Loxun je bil ustvarjen z namenom ustvarjanja velikih dokumentov XML [7].

Glavne lastnosti modula Loxun so:

- majhna poraba glavnega pomnilnika (dokument se zapisuje v datoteko v realnem času, zato ni potrebe po hrambi dokumenta v glavnem pomnilniku);
- podpora imenskim prostorom (na imenske prostore se sklicujemo s sintakso *prostor:značka*);
- mešanje tipa Unicode in 8-bitnih nizov (Loxunu podamo tip Unicode ali 8-bitni niz. Loxun 8-bitni niz pretvori v tip Unicode);
- avtomatsko pretvorba posebnih znakov (angl. *automatic escaping*) (posebni znaki, na primer <, `&`);
- robustnost (Med kreiranjem dokumenta so napake nemudoma javljene. Napaki sta manjkajoči zaključek značke ali nedefinirani imenski prostori.).

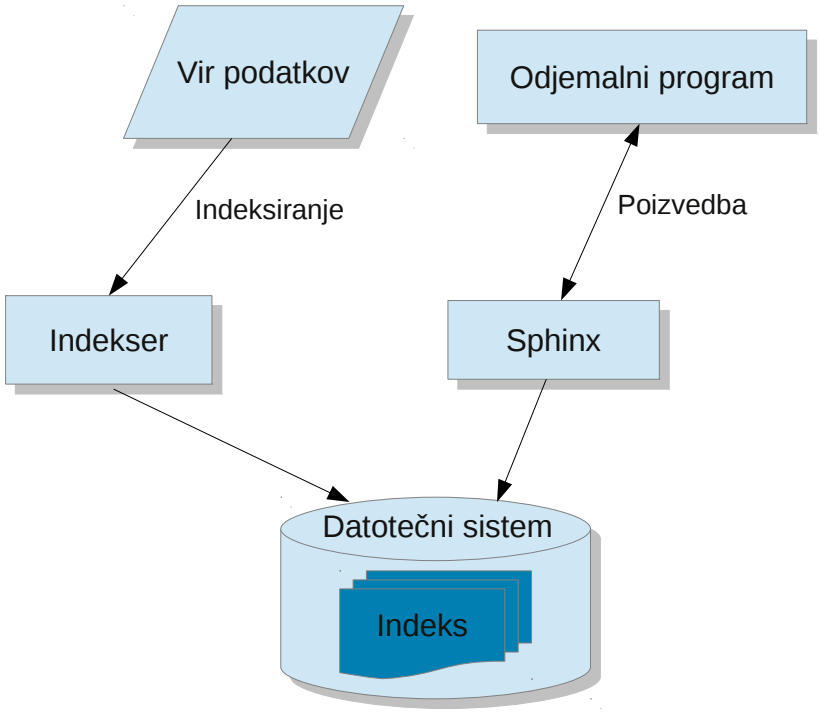
2.6 MySQL

MySQL je odprtokodni sistem za upravljanje relacijskih podatkovnih baz. MySQL atraktivnost izhaja iz preproste namestitve, uporabe in različic za operacijske sisteme Windows in UNIX (Linux, FreeBSD, Solaris, MAC OS, ...). Za načrtovanje podatkovne baze lahko uporabimo orodje MySQL Workbench (`http://mysql.com/products/workbench`). Orodje nam omogoča modeliranje, načrtovanje, pisanje poizvedb in administracijo podatkovne baze. Priporočljivo je načrtovanje podatkovne baze z orodjem Workbench, saj pri razumevanju tabel ali spremembi sheme vizualni pogled na tabele hitro razjasni vse povezave med njimi. Tako hitro ugotovimo pomen posameznih tabel.

Pri kreiranju podatkovne baze velikokrat pozabimo na pomembne nastavitve. Orodje MySQL Workbench nam pomaga ustvariti podatkovno bazo in doda vse pomembne parametre, katere morda pozabimo. Tak postopek upošteva vse priporočljive nastavitve podatkovne baze MySQL.

2.7 Sphinx

Sphinx (SQL Phrase Index) je odprtokodni iskalnik po tekstu (angl. *full text search server*), napisan v programskem jeziku C++. Načrtovan je z naslednjimi cilji: omogočiti hitro in kvalitetno iskanje po indeksiranih dokumentih ter integracijo z drugimi sistemi. Poganjamo ga lahko na operacijskih sistemih Windows in UNIX (Linux, FreeBSD, Solaris, MAC OS, ...). Arhitektura odprtokodnega iskalnika Sphinx je prikazana na sliki 2.3.



Slika 2.3: Arhitektura odprtokodnega iskalnika Sphinx.

Da Sphinx posreduje odgovor našim poizvedbam, ustvari posebno podatkovno strukturo, prirejeno za iskanje besed. Strukturi pravimo indeks. Procesu, ki naredi indeksno datoteko, pravimo indeksiranje (angl. *indexing*). Trenutno je implementiran en tip indeksa, ki je načrtovan za maksimalno indeksiranje in iskanje.

Podatki, ki jih Sphinx indeksira, lahko izvirajo iz različnih virov: relacijske podatkovne baze, standardnega vhoda, datoteke XML, HTML, ... Sphinx gleda na vir kot strukturiran dokument. Govorimo o podobnosti s tabelami v relacijski podatkovni bazi. Vsaka vrstica tabele ustreza dokumentu, vsak stolpec atributu (`http://sphinxsearch.com/docs/current.html#sources`).

Obstajata dva načina vzdrževanja vsebine indeksa. Prvi je datotečni indeks (angl. *disk-based index*), pri katerem vzdržujemo particije indeksa, ki smo jih ustvarili ročno. Prvo particijo imenujemo glavni indeks, drugo particijo pa indeks spremembe (angl. *delta index*). Nove dokumente vedno dodajamo v indeks spremembe. Po določenem času indeks spremembe združimo z glavnim indeksom. Takšen pristop omogoča, da nam ob majhnih spremembah ni potrebno ponovno indeksirati celotnega vira podatkov. Pristop je zelo učinkovit pri virih z veliko količino podatkov. Predstavljajmo si spletni forum, ki vsebuje 1.000000 arhiviranih prispevkov. Dnevno v indeks spremembe dodajamo nove prispevke in ga združimo z glavnim indeksom (`http://sphinxsearch.com/docs/current.html#delta-updates`).

Drugi način vzdrževanja vsebine indeksa je indeks v realnem času (angl. *real time index*). Podpira posodabljanje in brisanje že obstoječih dokumentov ter vnašanje novih. Vsaka operacija povzroči direktno spremembo indeksa. Pri obdelavi velike količine podatkov je indeks v realnem času manj učinkovit, saj gre za veliko vhodno-izhodnih operacij s trdim diskom.

Količino maksimalne zasedenosti glavnega pomnilnika s strani procesa Sphinx lahko nadzorujemo z dvema spremenljivkama. Prva je maksimalno dovoljeno število vzporednih poizvedb, druga pa je maksimalno število rezultatov, ki jih ena poizvedba vrne. Vsaka poizvedba za svoje delovanje potrebuje nekaj blokov glavnega pomnilnika. Spremenljivki igrata pomemben faktor pri optimizaciji latence iskanja.

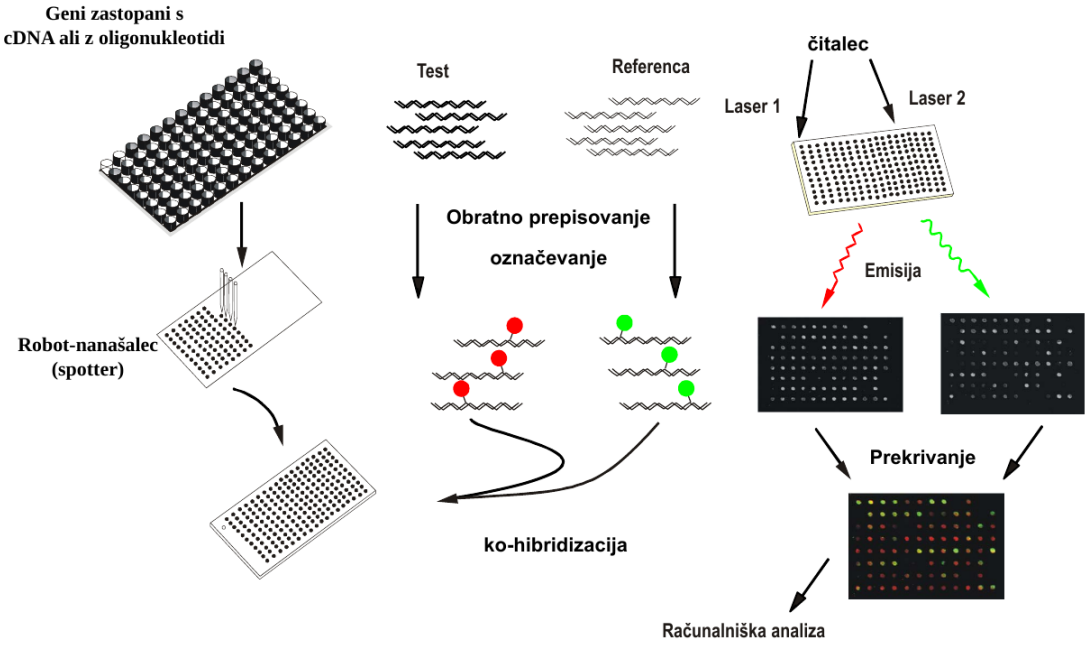
Poglavje 3

Poskusi z DNA-mikromrežami

DNA-mikromreže so zbirka sistematično urejenih molekul DNA (oligonukleotidov, PCR produktov, cDNA ali genomske DNA) imenovanih sonde, ki so kovalentno vezane na dvodimenzionalno podlago, ki je pogosto kar objektno stekelce. Pri mikromreži s celotnim genomom organizma, je vsak gen zastopan vsaj z eno točko, v kateri so na podlago vezane molekule DNA z znanim zaporedjem nukleinskih baz, ki je specifično za določen gen. Analitika na DNA-mikromrežah je osnovana na principu Watson-Crick-ove selektivne hibridizacije med komplementarnimi tarčnimi zaporedji molekul DNA in zaporedji sond.

Običajno se iz biološkega vzorca izolira RNA ali DNA, ki jo pred nanosom na mikromreže fragmentiramo in označimo s fluorescentno molekulo. mRNA molekule moramo pred označevanjem zaradi nestabilnosti prepisati v komplementarne DNA (cDNA) molekule z obratno transkripcijo in verižno reakcijo s polimerazo (RT-PCR). Zmes označenih fragmentov vzorčnih nukleinskih kislin nato hibridiziramo z imobiliziranimi molekulami DNA na mikromreži. Po spiranju nevezanih fragmentov odčitamo fluorescentne signale z laserskim optičnim čitalcem (angl. *scanner*). Hibridizacijo in spiranje lahko izvajamo ročno ali na avtomatski hibridizacijski postaji, ki zagotavlja bolj ponovljive rezultate, zaščito pred mehanskimi poškodbami čipa in odpravi probleme pri spiranju. Komplementarna vzorčna DNA se z imobilizirano DNA poveže z

dovolj močnimi vezmi, da se v procesu spiranja ne odstrani z mikromrež. Kot rezultat odčitavanja dobimo računalniško sliko, kjer je jakost hibridizacijskega signala ponazorjena z intenziteto slikovnih pik [4, 5]. Postopek merjenja izražanja genov z DNA-mikromrežami prikazuje slika 3.1.



Slika 3.1: Ekspresijsko profiliranje z DNA-mikromrežami – dvojno označevanje.

DNA-mikromreže omogočajo sledenje izražanja velikega števila genov, lahko tudi celotnega transkriptoma, če so poznana zaporedja vseh genov, ki se izražajo v preučevanem organizmu. Pri rastlinah so DNA-mikromreže zelo uporabne na primer za študij odziva na škodljivce oziroma povzročitelje bolezni, saj omogočajo vpogled v celostno stanje izražanja genov določenega rastlinskega organa ali tkiva v različnih časovnih točkah. S primerjavo vzorcev napadenih in zdravih rastlin lahko identificiramo gene udeležene v rastlinski obrambi ter njihove regulatorne elemente, kar je ključno v raziskavah obrambnih signalnih poti (Wan in sodelavci, 2002). V rastlinski biologiji se poleg študij izražanja rastlinskih genov, DNA-mikromreže uporabljajo tudi za detekcijo povzročiteljev bolezni (Boonham in sodelavci, 2003; Bystricka in sodelavci, 2005), z njimi pa lahko preučujemo tudi spremembe v izražanju genov v mutiranih ali gensko spremenjenih rastlinah (Aharoni in Vorst, 2002; Lodha in Basak, 2011). Pri rastlinah je največ študij izražanja genov z DNA-mikromrežami narejeno na navadnem repnjakovcu (pregled v Aharoni in Vorst, 2002), katerega celotni genom je bil objavljen že na prelomu tisočletja (Arabidopsis Genome Initiative, 2000). Objavam rezultatov poskusov z DNA-mikromrežami pri navadnem repnjakovcu po številu objavljenih poskusov sledijo ekonomsko pomembne poljščine: riž, koruza, ječmen, pšenica, soja, krompir, vinska trta, paradižnik in tobak (podatki iz podatkovne zbirke ArrayExpress, Parkinson in sodelavci, 2011) [4].

3.1 Predstavitev poskusa

Poskus z DNA-mikromrežo je predstavljen z množico datotek. Minimalni nabor datotek, ki sestavljajo poskus, so datoteka s podatki o vzorcih (angl. *Phenodata*), datoteke s surovimi podatki (angl. *Rawdata*), datoteka s podatki o genih (angl. *Annotation*), datoteka z normaliziranimi vrednostmi (angl. *Normalized*), datoteke z rezultati (angl. *Result*) in datoteka s podatki o datotekah z rezultati (angl. *Metaresult*). Množica datotek predstavlja celoto. Med seboj so odvisne, ena brez druge predstavljajo nepopoln poskus.

Datoteke *Phenodata*, *Annotation*, *Normalized*, *Result*, *Metaresult* so tekstovne datoteke. Njihovo vsebino predstavlja tekst. Vsebujejo več ključnih vrednosti, ki jih med seboj ločimo z znakom tabulator (`\t`). Ko za ločitev vrednosti uporabimo znak tabulator, nastanejo stolpci. Ključne vrednosti posamezne datoteke se nahajajo v začetnih stolpcih. Vse datoteke so kodirane v načinu UTF-8 brez BOM. Konec vrstice datoteke predstavlja eden od znakov:

- LF (angl. *line feed*);
- CR+LF (angl. *carriage return + line feed*).

Datoteka *Rawdata* je poljubna datoteka. Zanjo ne veljajo nobena pravila kodiranja znakov ali načini zapisa konca vrstice. Datoteko lahko predstavlja dokument word, excel, pdf, datoteka tipa XML, HTML, datoteka poljubnega programa, ...

3.1.1 Datoteka s podatki o vzorcih

Datoteka s podatki o vzorcih (angl. *Phenodata*) je tekstovna datoteka. V njej definiramo vzorce (angl. *Sample name*), ki smo jih uporabili pri poskusu. Vsakemu vzorcu pripada ena ali več datotek *Rawdata*. Ime datoteke *Rawdata* skupaj z vzorcem zapišemo v eni vrstici. Ko vzorcu pripada več datotek *Rawdata*, se ime vzorca ponovi v večih vrsticah.

Prva vrstica datoteke vsebuje dva stolpca. Prvi stolpec je `Filename`, drugi pa `Sample name`. Dodani so lahko poljubni dodatni stolpci, ki vsebujejo vse vrednosti znakov standarda Unicode. Prvo vrstico imenujemo `meta vrstica`. Stolpci `meta vrstice` predstavljajo pomen njihovim vrednostim navpično po vrsticah. Stolpec `Sample name` nakazuje definiranje vzorcev, stolpec `Filename` pa imena datotek *Rawdata*. Primer veljavne datoteke s podatki o vzorcih prikazuje slika 3.2. Veljaven niz znakov za prvi in drugi stolpec predpisuje regularni izraz, `^[a-zA-Z0-9-_]+$`. Vrednost posamezne vrstice stolpca je veljavna, ko velja, da od začetka do konca niza vsebuje kombinacije znakov od a do z, A do Z, 0 do 9, znak - ali znak _. Minimalna dovoljena dolžina niza je 1.

Filename	Sample name	fosf.(ug/ml)	time (min)	
NUID-0000-0027-2305.cel	sa1-control-20min	0		20
NUID-0000-0027-2306.cel	sa1-fosf-1-20min	1		20
NUID-0000-0027-2307.cel	sa1-fosf-4-20min	4		20
NUID-0000-0027-2308.cel	sa1-fosf-1-40min	1		40
NUID-0000-0027-2309.cel	sa1-fosf-4-40min	4		40
NUID-0000-0027-2310.cel	sa2-control-10min	0		10
NUID-0000-0027-2311.cel	sa2-fosf-4-40min	4		40
NUID-0000-0027-2312.cel	sa3-control-0min	0		0
NUID-0000-0027-2313.cel	sa3-fosf-4-10min	4		10

Slika 3.2: Veljavna datoteka Phenodata.

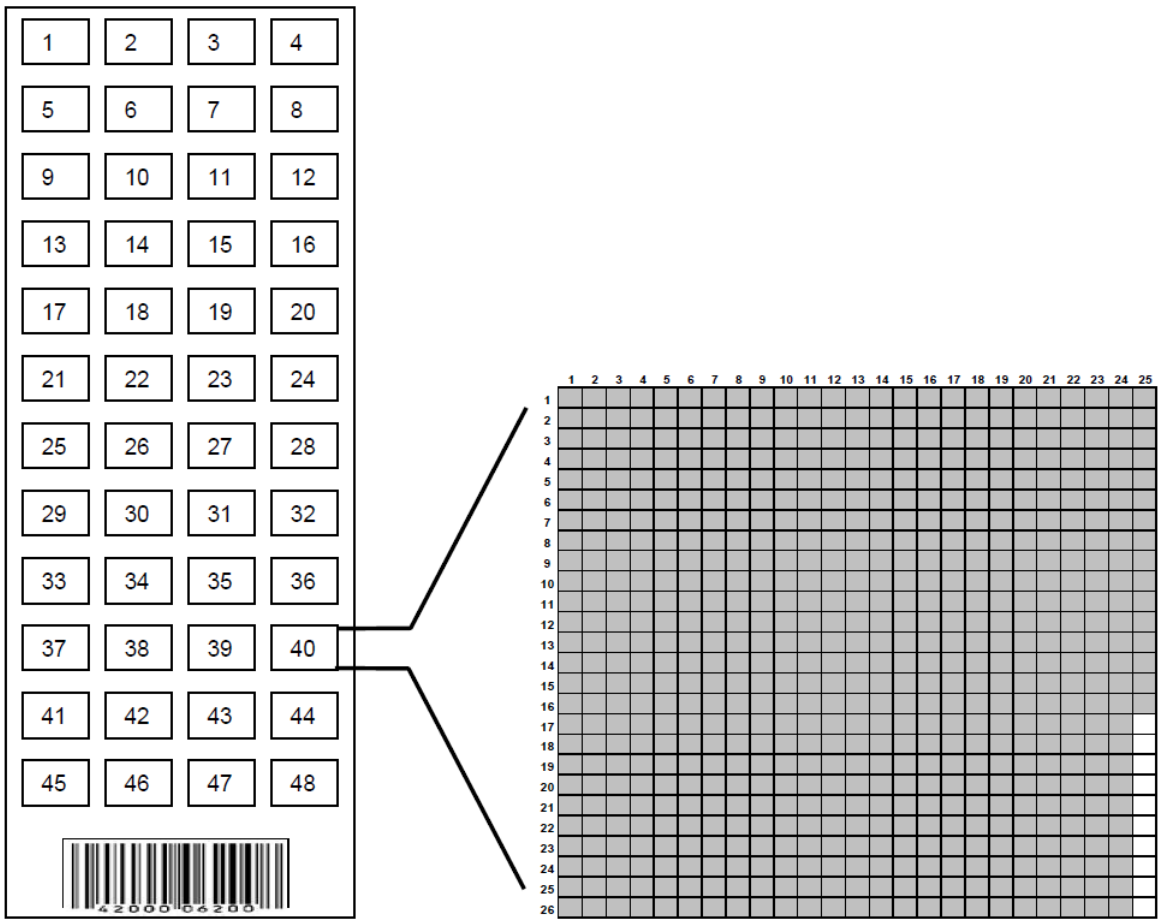
3.1.2 Datoteka s surovimi podatki

Datoteka s surovimi podatki (angl. *Rawdata*) vsebuje neobdelane rezultate poskusa in predstavlja temelj začetka analize rezultatov poskusa. Datoteko dobimo z DNA-mikromrežnim čitalnikom (angl. *microarray scanner*). S pomočjo datoteke *Rawdata* ustvarimo datoteko *Normalized* (3.1.4).

V datoteki *Phenodata* (3.1.1) vsakemu vzorcu pripada eno ali več imen datotek *Rawdata*. Poskus mora vsebovati vse datoteke *Rawdata*, ki so naštete v stolpcu `Filename` datoteke *Phenodata*.

3.1.3 Datoteka s podatki o genih

Datoteka s podatki o genih (angl. *Annotation*) je tekstovna datoteka. Vsebuje imena genov in njihove opise, ki so bili nanešeni na DNA-mikromrežo. Datoteko zagotovi proizvajalec DNA-mikromreže. Prvo vrstico datoteke imenujemo `meta vrstica` (3.1.1). Vrstica vsebuje dva stolpca. Prvega z imenom `geneID` in drugega z imenom `description`. Prvi stolpec označuje ime, drugi pa opis gena. Stolpca skupaj predstavljata par. V `meta vrstici` so dovoljeni še dodatni poljubni stolpci. Najpogostejša dodatna stolpca sta `Column` in `Row`. Predstavljata koordinate genov na DNA-mikromreži. Poleg genov dobimo tudi njihovo razporeditev na DNA-mikromreži. Zaradi zanesljivosti rezultata poskusa je enak gen na DNA-mikromrežo nanešen večkrat (angl. *redundancy*). Na DNA-mikromreži načrtovalec poskusa svoje vzorce razdeli v bloke, kar prikazuje slika 3.3 za krompirjevo cDNA mikromrežo. V datoteki *Annotation* se tako pojavlja tudi stolpec `Block`, kar prikazuje slika 3.4.



Slika 3.3: Shema TIGR krompirjevih cDNA mikromrež. cDNA je natisnjena v 48 blokih (levo), v vsakem (desno) je 25 stolpcev in 26 vrstic točk cDNA.

geneID	description	Block	Column	Row	ID
STMGV34	Putative phospholipid-transporting ATPase 8 [Arabidop				
STMHK82	Hypothetical protein F12E4_320 [Arabidopsis thaliana				
STMIB28	Predicted by genscan [Arabidopsis thaliana (Mouse-ear				
STMIO76	Epoxide hydrolase [Nicotiana tabacum (Common tobacco)				
STMJA34	Hypothetical protein P0566A10.33-2 [Oryza sativa (jap				
STMCB13	None	1	1	1	1
STMCN61	Oligouridylate binding protein-like protein [Solanum				
STMDC19	Do not use. Not validated.			1	3
STMDR67	None	1	4	1	1
STMEK13	Do not use. Not validated.			1	5
STMEW61	Histone H2B [Arabidopsis thaliana (Mouse-ear cress)]				

Slika 3.4: Primer veljavne datoteke Annotation, ki vsebuje še najpogostejše stolpce Block, Column in Row.

3.1.4 Datoteka s podatki o normaliziranih vrednostih

Datoteka s podatki o normaliziranih vrednostih (angl. *Normalized*) je tekstovna datoteka. Vsebuje normalizirane vrednosti izraženosti genov, ki pripadajo vzorcem, definiranim v datoteki *Phenodata*. Prisotna je `meta vrstica` (3.1.1), v kateri je prvi stolpec `geneID`, vsi naslednji stolpci pa so edinstvena imena vzorcev (angl. *Sample name*). Imena vzorcev v datoteki *Normalized* predstavimo z množico A, imena vzorcev v datoteki *Phenodata* pa z množico B. Imena vzorcev iz množice A se morajo ujemati z imeni iz množice B. Veljati mora $A \subset B$. Vse vrstice stolpcev z edinstvenimi imeni vzorcev vsebujejo numerične vrednosti, ki predstavljajo izraženost gena pri določenem vzorcu. Dovoljena numerična vrednost je tudi `NaN` (angl. *not a number*). Z vrednostjo `NaN` nadomestimo manjkajočo vrednost ali pa je `NaN` rezultat operacije števil v plavajoči vejici (standard IEEE 754).

Datoteka *Normalized* je ustvarjena s pomočjo datotek *Rawdata*. Njena obdelava vključuje pomoč z različnimi statističnimi programskimi orodji, kot je R (`http://www.r-project.org`). Slika 3.5 prikazuje odsek veljavne datoteke *Normalized*, pri čemer je prisotna numerična vrednost `NaN`.

geneID	Igor-30min-3	Igor-30min-4	
STMCE78	NaN	NaN	-0.119892073
STMCE79	-0.368017036	-0.195585187	
STMCE80	-0.860617677	-0.080809204	
STMCE81	0.349051374	-0.020659175	NaN
STMCE82	-0.803090662	-0.110064347	
STMCE83	-1.088019812	0.374125687	NaN
STMCE84	-0.033098588	0.02532242	-0.061716532
STMCE85	NaN	NaN	NaN NaN

Slika 3.5: Odsek veljavne datoteke Normalized.

3.1.5 Datoteka z rezultati

Datoteka z rezultati (angl. *Result*) je tekstovna datoteka, ki primerja izraženost genov dveh ali več vzorcev med seboj. Podatki o izraženosti genov posameznega vzorca so vzeti iz datoteke *Normalized* (3.1.4). Rezultati se pridobijo z različnimi statističnimi orodji, kot je R (`http://www.r-project.org`).

Datoteka vsebuje `meta vrstico` (3.1.1). Prvi stolpec `geneID` vsebuje imena genov, uporabljenih pri primerjavi. Imena genov se morajo ujemati z geni, definiranimi v datoteki *Annotation*. Dovoljeni so dodatni poljubni stolpci. V odvisnosti od vrste statistične obdelave najpogosteje sledijo stolpci `logFC`, `AveExpr`, `P.Value` in `Cluster`, kot prikazuje slika 3.6.

geneID	logFC	AveExpr	t	P.Value
STMCG89	2.60725073	6.492736138	6.358687657	0.000187686
STMDE91	1.209601346	6.903924715	4.636019899	0.000307798
STMDH14	-2.410133421		8.837360838	-5.564318428
STMD082	2.133436879	9.342505462	5.363796066	0.000598763
STMH080	1.232102815	9.319662509	4.195542131	0.000660759
STMIX05	-1.019551607		10.10957413	-4.070968442
STMEU77	-0.724985028		8.586761024	-3.957917805
STMHS01	2.045448903	8.525015815	4.604359496	0.001180239
STMED79	1.837846022	11.0375154	4.543475057	0.001289869

Slika 3.6: Odsek veljavne datoteke Result.

3.1.6 Datoteka s podatki o datotekah z rezultati

Datoteka s podatki o datotekah z rezultati (angl. *Metaresult*) je tekstovna datoteka, v kateri so našteta imena datotek *Result* (3.1.5). Glede na vrsto statistične obdelave razlikujemo tri tipe datotek:

- primerjava (angl. *comparison*);
- profil (angl. *profiles*);
- grupiranje v skupine po določenem kriteriju (angl. *clustering*).

Datoteka vsebuje `meta vrstico` (3.1.1). Prvi stolpec je `Filename`. Vrstice prvega stolpca definirajo imena datotek *Result*. Dovoljeni so dodatni poljubni stolpci.

Pogosto ime datoteke *Result* je `vzorec_čas_vzorec_čas`. Primer imena `igor_12h_VS_igor_30m` nam v grobem sporoča, da gre za primerjavo rezultatov okužbe vzorca `igor` po 30-ih minutah in 12-ih urah. Da `igor` predstavlja vzorec, se lahko prepričamo v datoteki *Phenodata*. Datoteka *Metaresult* največkrat vsebuje dodatni stolpec `Compare`, ki nam podrobno opiše posamezno datoteko *Result*, kar prikazuje slika 3.7.

```
Filename»   Compare»    Comments
des_b03P-des_b03m.txt»  Desiree-PVY 3dpi (bottom) vs Desiree-mock 3dpi (bottom).
nah_b05P-nah_b05m.txt»  NahG-PVY 5dpi (bottom) vs NahG-mock 5dpi (bottom).
```

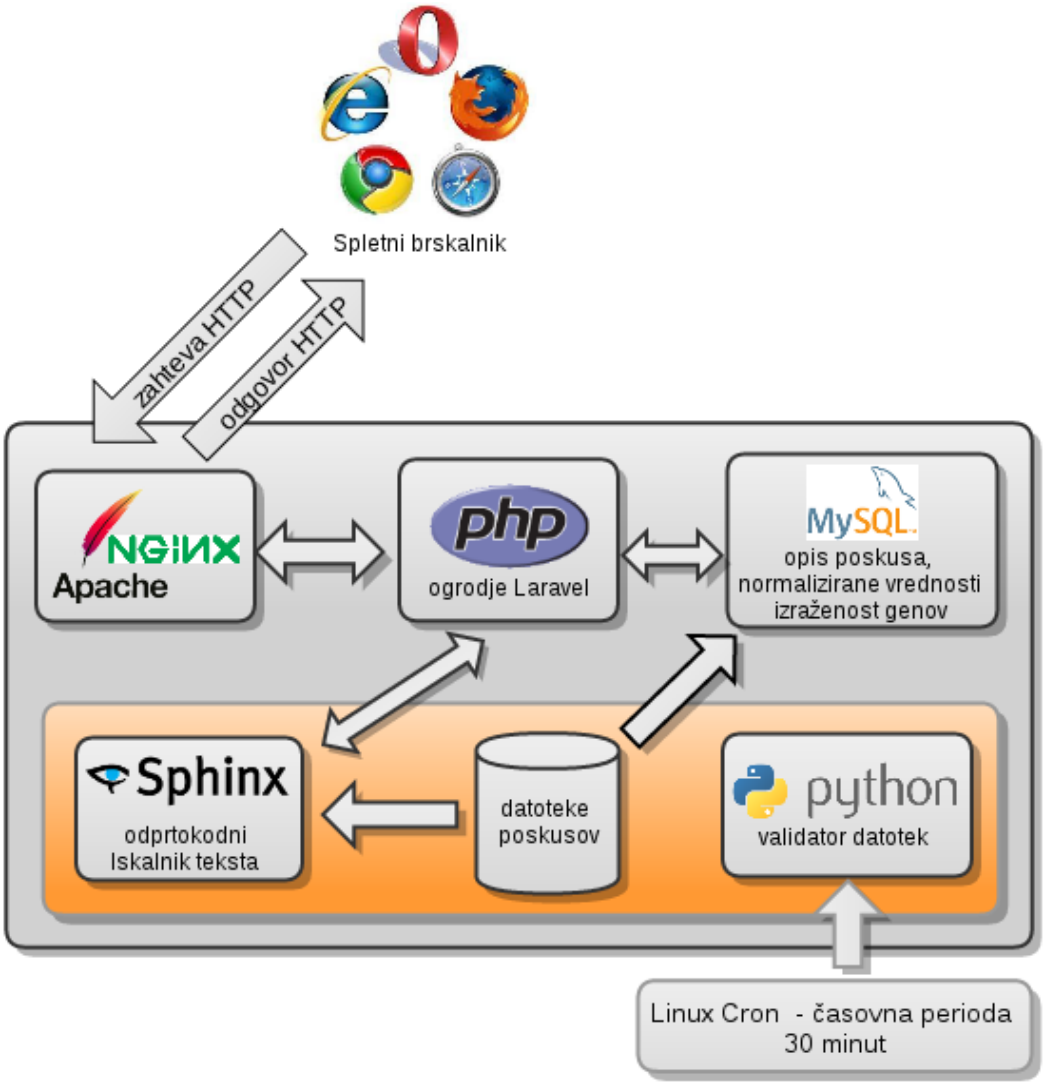
Slika 3.7: Primer datoteke Metaresult, tipa primerjava.

Poglavje 4

Aplikacija Fitobase

Fitobase je spletna aplikacija, ki za svoje delovanje uporablja skupek odprtokodne programske opreme, imenovan sklad LAMP, in odprtokodni iskalnik teksta Sphinx (2.3). LAMP je akronim, katerega posamezne črke predstavljajo operacijski sistem Linux, strežnik http Apache, podatkovno bazo MySQL (2.6) in programski jezik PHP (2.3). Posamezne plasti sklada LAMP lahko nadomesti druga programska oprema, ki še vedno opravlja enako funkcijo. V praksi zasledimo LAPP (Linux, Apache, PostgreSQL, PHP), LNMP (Linux, Nginx, MySQL, PHP), WAMP (Windows, Apache, MySQL, PHP), ... Za programski jezik se uporablja tudi Perl ali Python, strežnik http pa nadomešča nginx (`http://http://nginx.org/`) ali lighttpd (`http://lighttpd.net`).

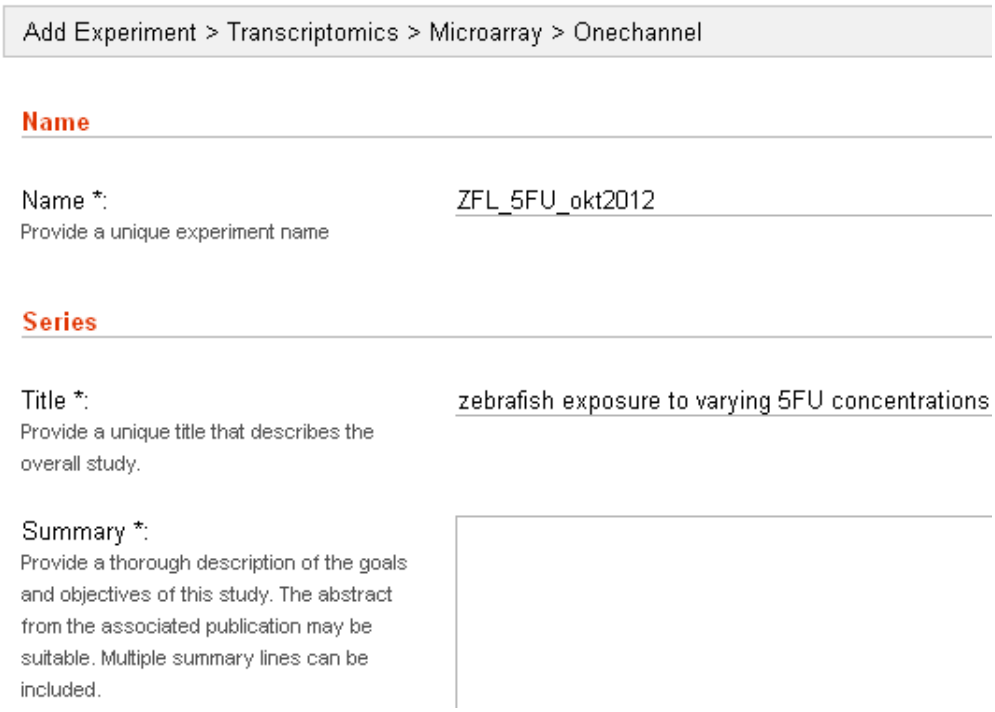
Arhitektura aplikacije je prikazana na sliki 4.1. Aplikacija predstavlja spletni uporabniški vmesnik in nudi uporabniku v prvem koraku opis in urejanje poskusa z DNA-mikromrežo, v drugem koraku pa nalaganje in iskanje po vseh v naprej definiranih datotekah poskusa. Za uporabnika je zagotovo najbolj pomembna predstavitev rezultatov iskanja, zato smo se osredotočili na hitro in kvalitetno iskanje.



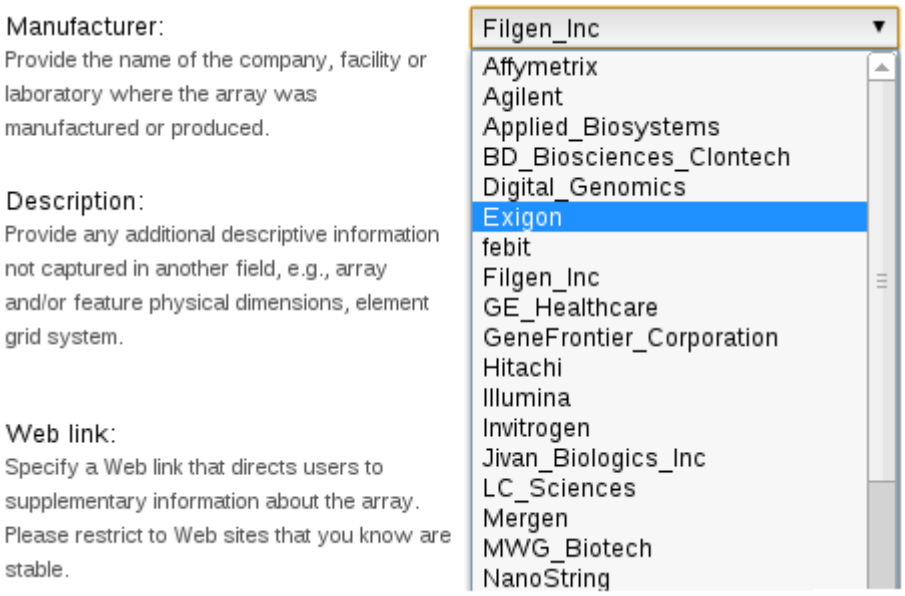
Slika 4.1: Arhitektura aplikacije Fitobase.

4.1 Delovni tok vnosa poskusa

Poskus z DNA-mikromrežami moramo najprej ustvariti in potrebne datoteke naložiti na spletni strežnik, če hočemo po njih iskati. Prvi korak pri ustvarjanju novega poskusa je njegov opis. Omenimo nekaj elementov opisa, kot so ime poskusa, cilji poskusa, uporabljene tehnologije, proizvajalca DNA-mikromreže (Affymetrix, Agilent, Applied Biosystems, Ilumina), organizem, od katerega smo pridobili biološki material in protokole, ki smo jih uporabili. Odsek opisa poskusa prikazujeta sliki 4.2 in 4.3.

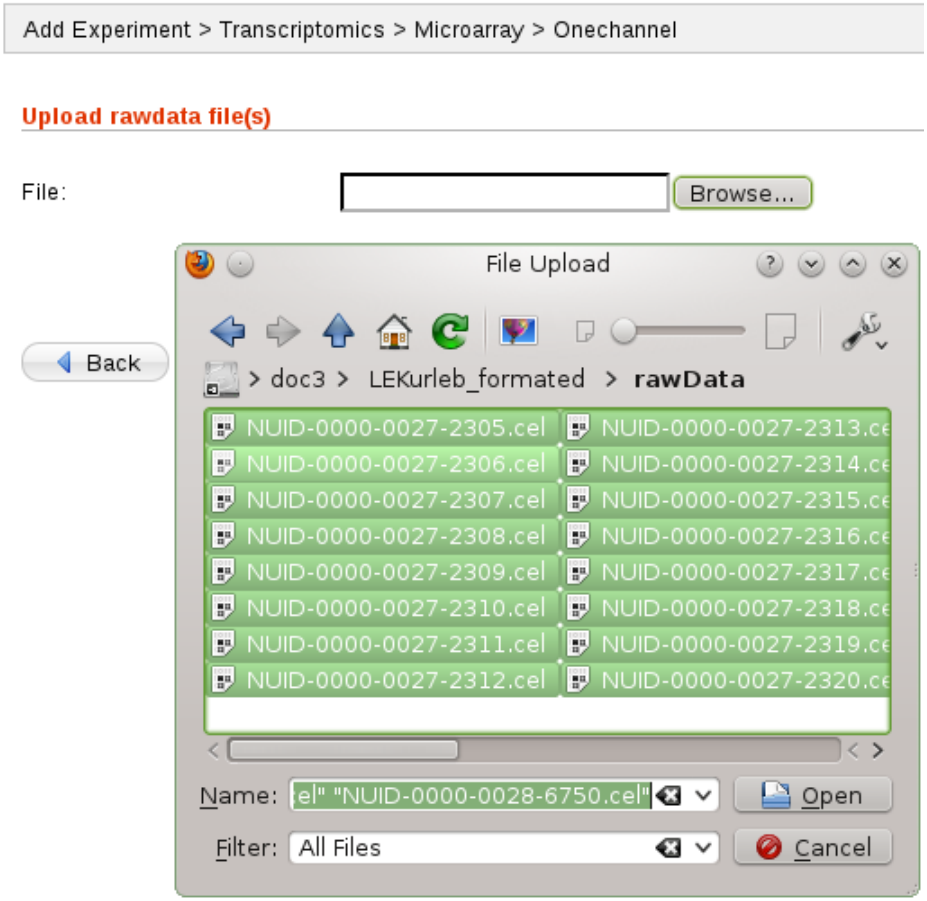


Slika 4.2: Vnos imena, naslova in povzetka poskusa z DNA-mikromrežo.



Slika 4.3: Izbira proizvajalca DNA-mikromreže.

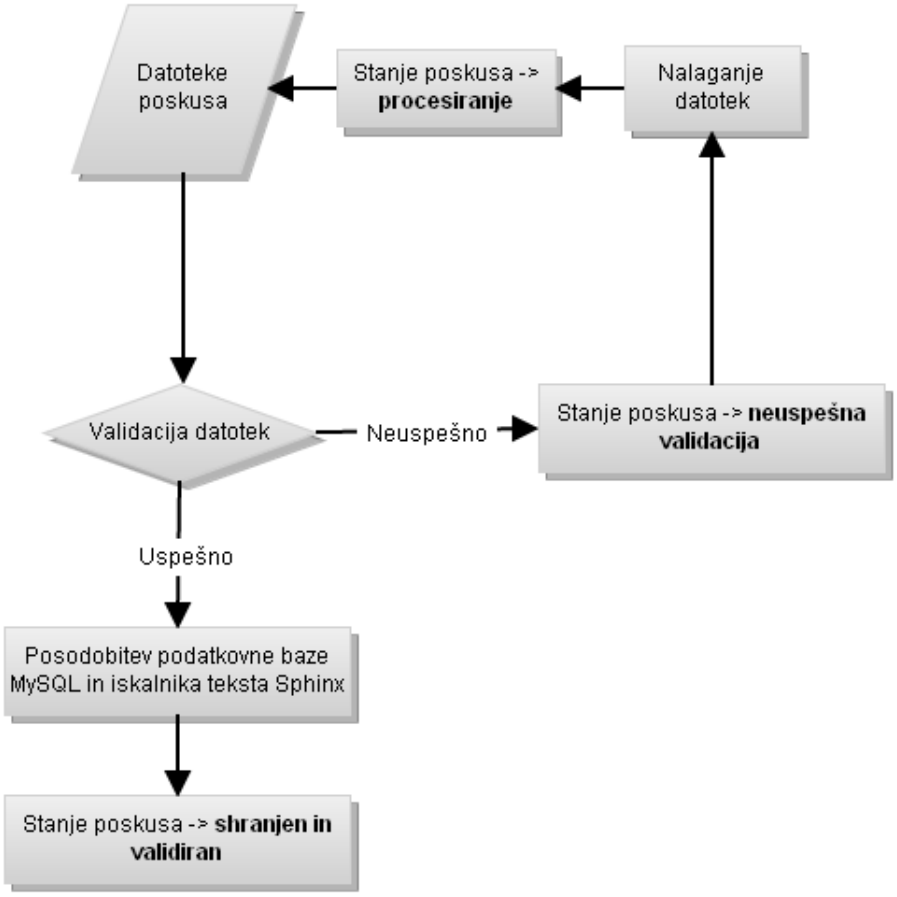
Drugi korak predstavlja nalaganje datotek poskusa, ki smo jih opisali v poglavju 3.1. Datoteke naložimo s pomočjo forme HTML, kot prikazuje slika 4.4 za datoteke *Rawdata* (3.1.2).



Slika 4.4: Nalaganje datotek Rawdata s pomočjo forme HTML.

Ob uspešnem prenosu posamezne datoteke na strežnik lahko začnemo z validacijo vsebine datoteke v skladu z njeno definirano obliko (3.1). Zaradi možnih velikih datotek proces validacije traja zelo dolgo. Če datoteka presega velikost 100 MB, se proces validacije podaljša za več minut. Vnos poskusa tako minimalno povečamo za omenjen čas validacije. Zaželjeno je, da spletna aplikacija uporabniku ponuja interaktivno in odzivno storitev. V ta namen se proces validacije datotek začne ob uspešnem prenosu vseh da-

totek na strežnik, istočasno pa poskus preide v stanje `procesiranje`. Vsebino datotek validiramo neodvisno od delovanja spletne aplikacije (angl. *offline validation*). Proces validacije se zažene časovno, s periodo 30 minut, kar skupaj s programsko kodo opisuje priloga A. Upoštevamo določen vrstni red validacije, saj nekatere datoteke potrebujejo vrednosti že validiranih datotek. Na primer datoteka *Annotation* vsebuje imena genov, ki jih datoteka *Normalized* potrebuje za validacijo. Validacija datoteke *Annotation* se zato opravi najprej. Pri validaciji datotek *Phenodata, Annotation, Result, Metaresult, Normalized* preverjamo vsebino vsake vrstice, kar za datoteko *Phenodata* prikazuje programska koda v prilogi B. Ob neupoštevanju oblike vrstice datoteke zabeležimo napako. Napaka je predstavljena s sporočilom o napaki in vrstico datoteke, ki sporoča mesto napake. Poskus, ki je zabeležil vsaj eno napako pri pregledu vseh datotek, preide v stanje `neuspešna validacija`. V želji po uspešni validaciji uporabnik ponovno naloži popravljene datoteke. Poskus preide v stanje `procesiranje`. Cikel traja, dokler vsebina datotek ne ustreza obliki datotek. Poskus preide v stanje `shranjen in validiran`, ko vsebina datotek popolnoma ustreza obliki datotek in po posodobitvi indeksa odprtokodnega iskalnika teksta Sphinx in podatkovne baze MySQL z vsebino datotek poskusa. Delovni tok vnosa poskusa je prikazan na sliki 4.5.



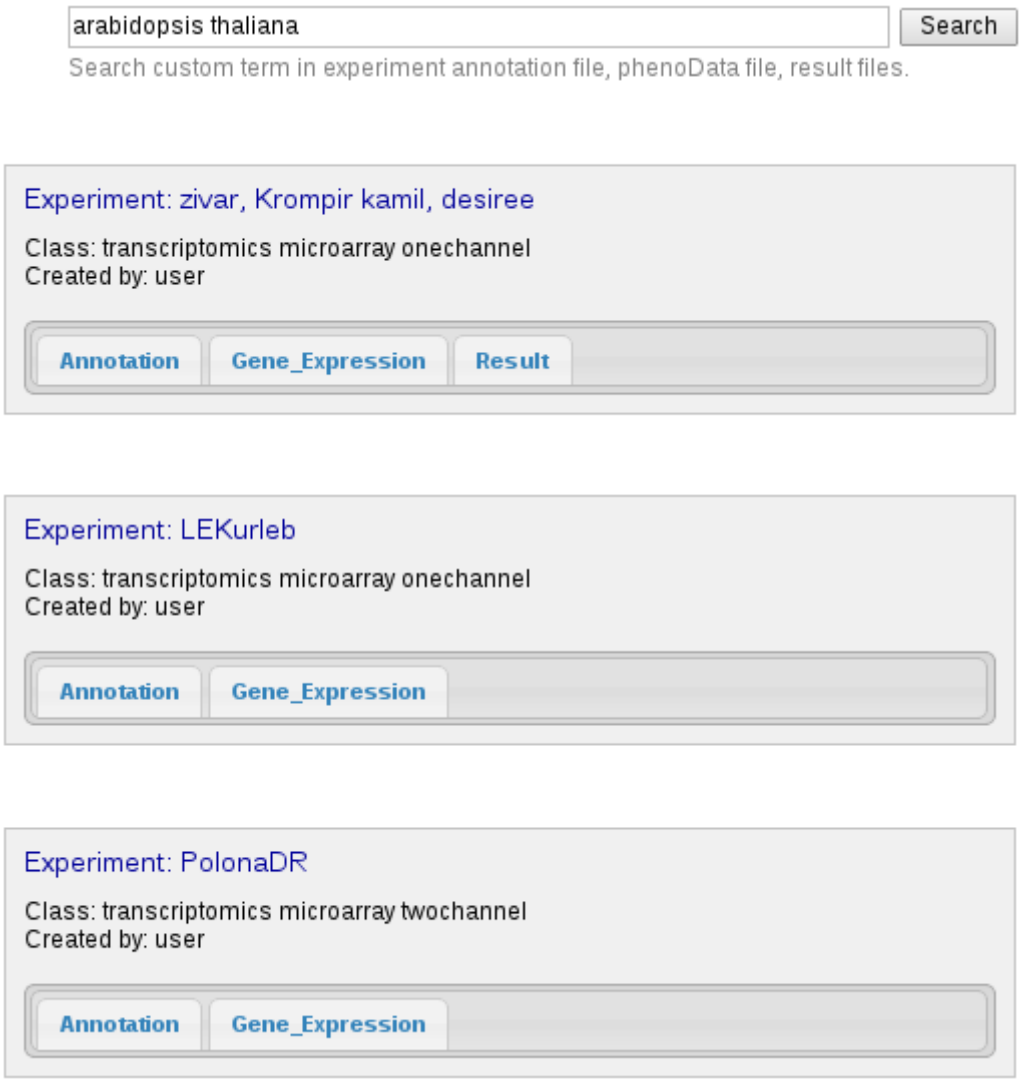
Slika 4.5: Delovni tok vnosa poskusa.

4.2 Datoteke, po katerih iščemo

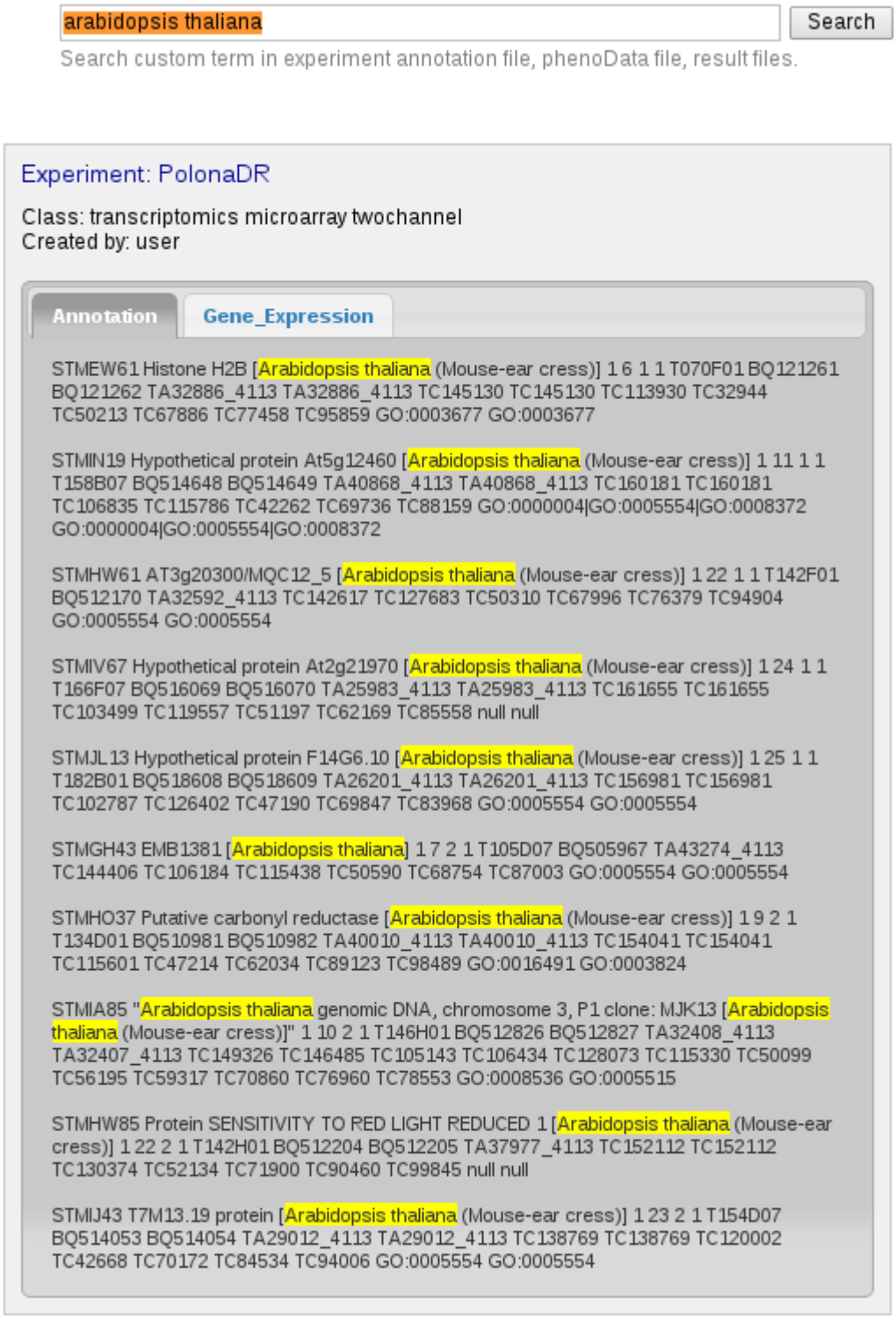
Iščemo po vsebini datotek *Phenodata* (3.1.1), *Annotation* (3.1.3) in *Result* (3.1.5), kar skupaj predstavlja iskalne datoteke. Iskanje po datotekah *Rawdata* (3.1.2) ni smiselno, saj so poljubnega tipa. Prav tako ne iščemo po datotekah *Metaresult* (3.1.6), saj vsebujejo le imena datotek *Result*, in po datotekah *Normalized* (3.1.4), saj vsebujejo le numerične vrednosti.

4.3 Predstavitev rezultatov

Iskani niz je lahko vsebovan enkrat ali večkrat, v vsaki vrstici iskalnik datotek vsakega poskusa, ki je v stanju `shranjen in validiran`. Uporabniku kot rezultat prikažemo celotno vrstico, kjer se iskani niz nahaja. Izkazalo se je, da to zadošča potrebam uporabnika. To vrsto iskanja imenujemo `osnovno iskanje`. Iskani niz se lahko hkrati nahaja v vseh treh iskalnih datotekah. V ta namen rezultate vsake iskalne datoteke prikažemo v obliki zavihkov, kar prikazuje slika 4.6. Vsak zavihek vsebuje maksimalno 10 vrstic, kjer se iskani niz nahaja, kar prikazuje slika 4.7.



Slika 4.6: Predstavitev rezultatov v obliki zavihkov za vsako iskalno datoteko posebej.

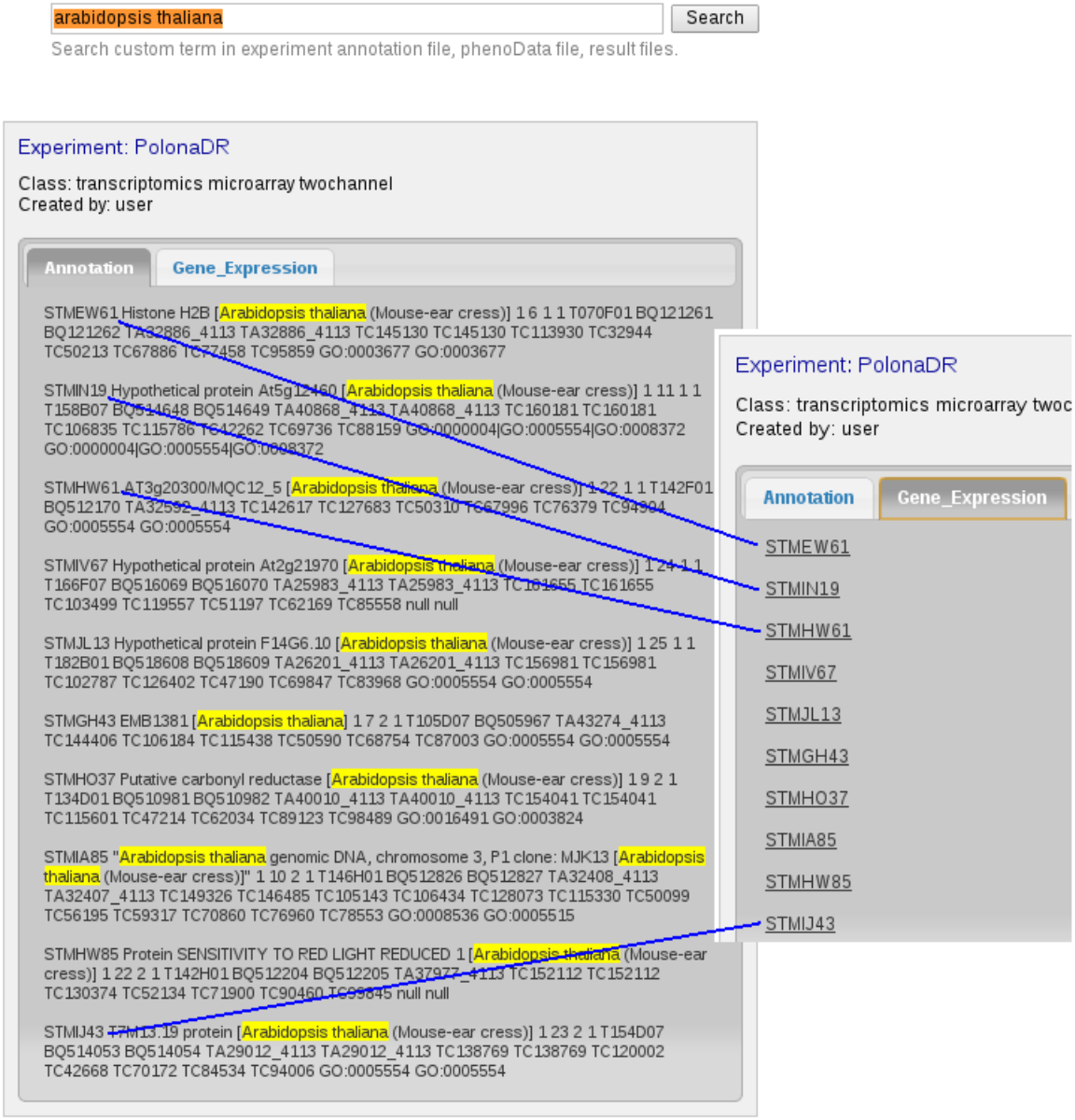


Slika 4.7: Vrstice datoteke Annotation, kjer je najden iskani niz arabidopsis thaliana (rastlina, navadni repnjakovec).

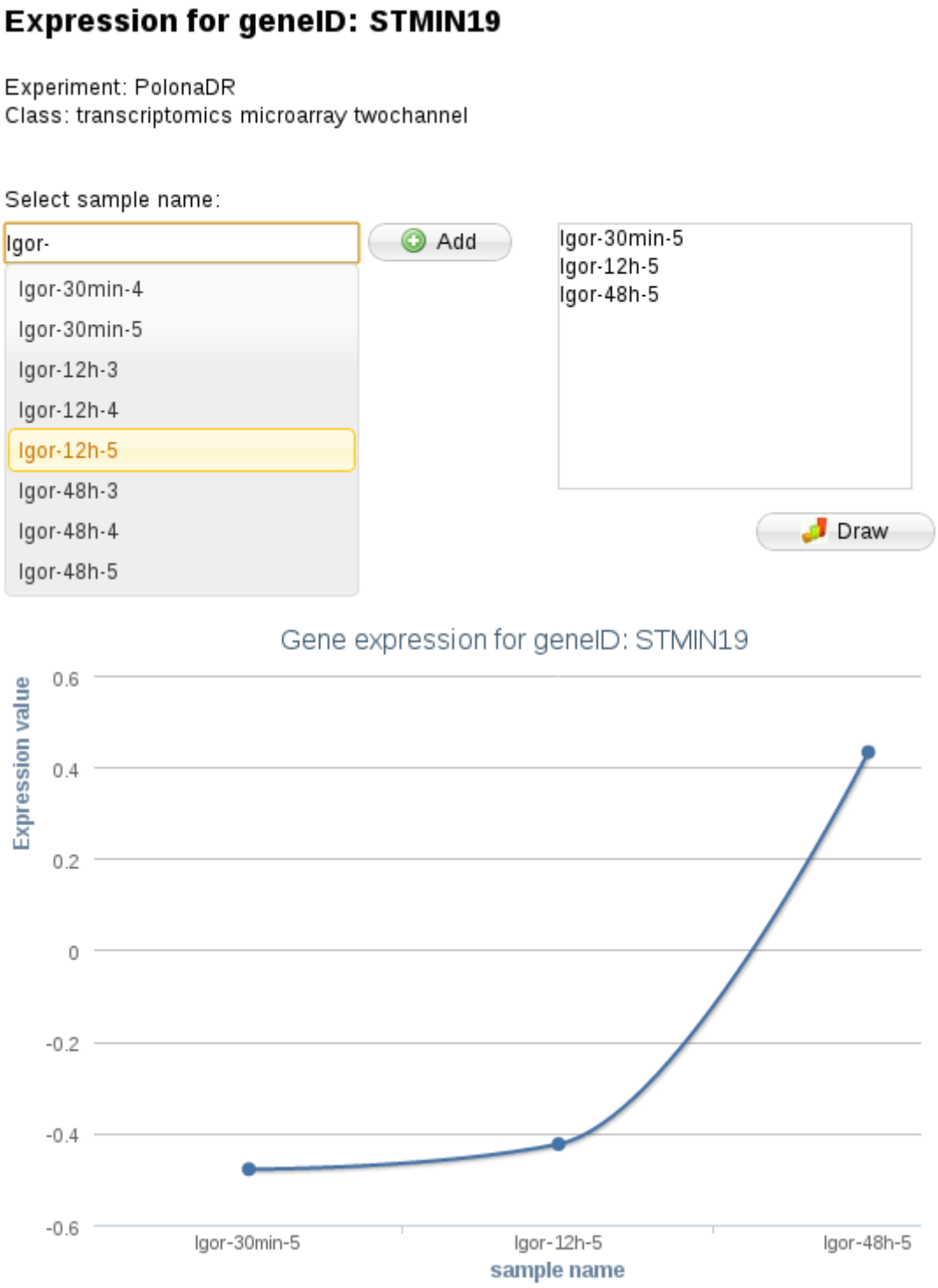
4.3.1 Izraženost genov

S pomočjo datoteke *Normalized* (3.1.4) poleg `osnovnega iskanja` uporabniku nudimo izris izraženosti posameznih vzorcev ali genov določenega poskusa. Za izris gena naštejemo imena vzorcev, za izris vzorca pa imena genov, ki nas zanimajo. Do omenjenih izrisov pridemo posredno preko `osnovnega iskanja`.

Datoteka *Annotation* vsebuje prvi stolpec, drugi stolpec ter dodatne poljubne stolpce s poljubno vsebino. Prvi stolpec vsebuje imena genov, drugi pa opis genov. Ko se iskani niz nahaja v vrsticah datoteke *Annotation*, posamezno najdeno vrstico prikažemo v zavihku `Annotation`, v zavihku `Gene_Expression` pa prikažemo pripadajoča imena genov najdenih vrstic. Celotno vrstico datoteke *Annotation* identificiramo z imenom gena. Zavihek `Gene_Expression` vsebuje imena genov, ki pripadajo najdenim vrsticam v zavihku `Annotation`, kar prikazuje slika 4.8. Vse pogoje za izris izraženosti gena izpolnjujemo, če se najdeno ime gena nahaja v datoteki *Normalized*. Ostane nam še izbira vzorcev. Slika 4.9 prikazuje izraženost gena `STMIN19` po okužbi štirih vzorcev slovenske sorte krompirja igor z virusom. Vzorci `Igor-30min-5, Igor-12h-5` in `Igor-48h-5` predstavljajo časovne značke trajanja okužbe virusa.

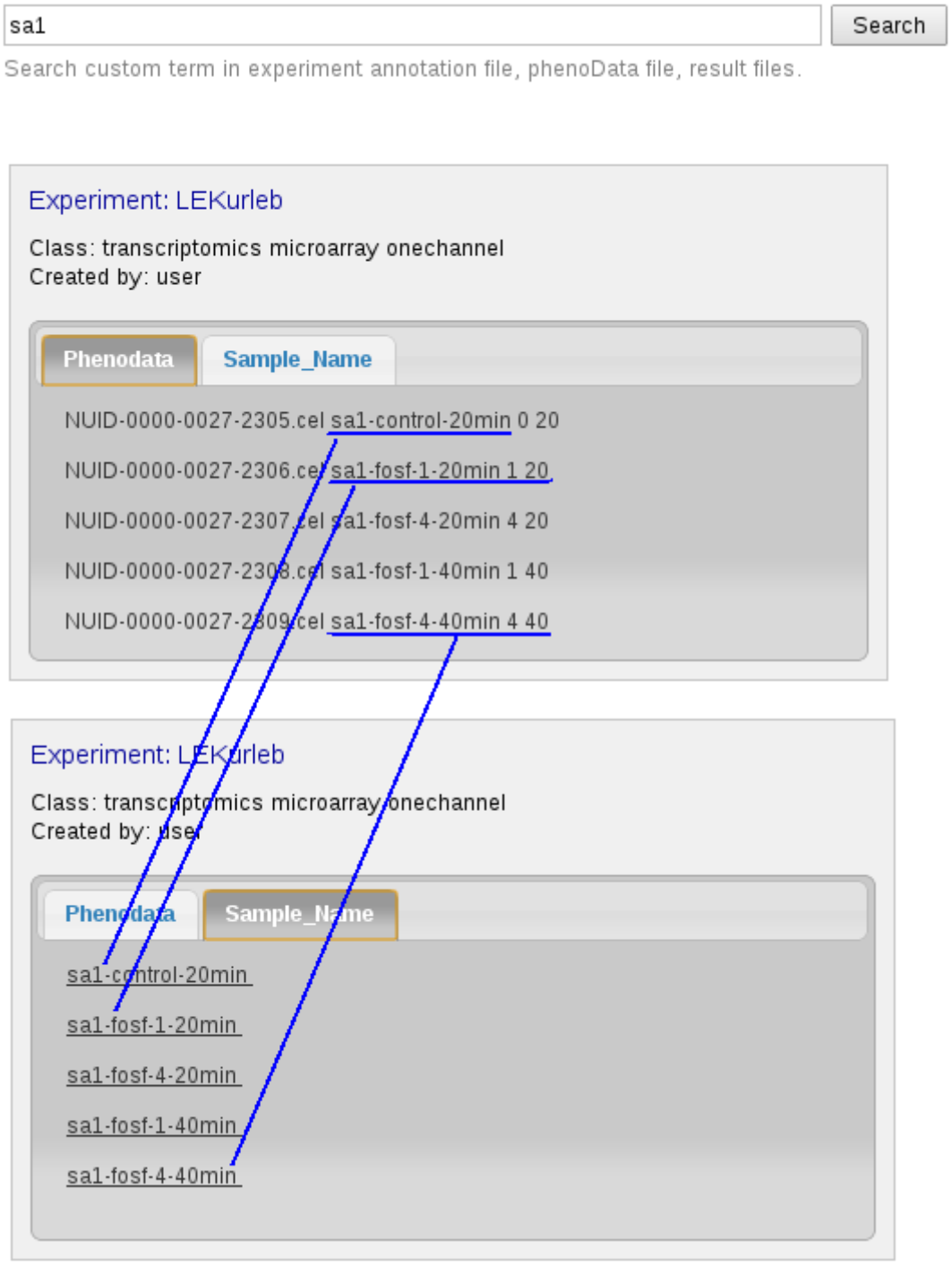


Slika 4.8: Prikaz rezultatov zavihka Gene_Expression.

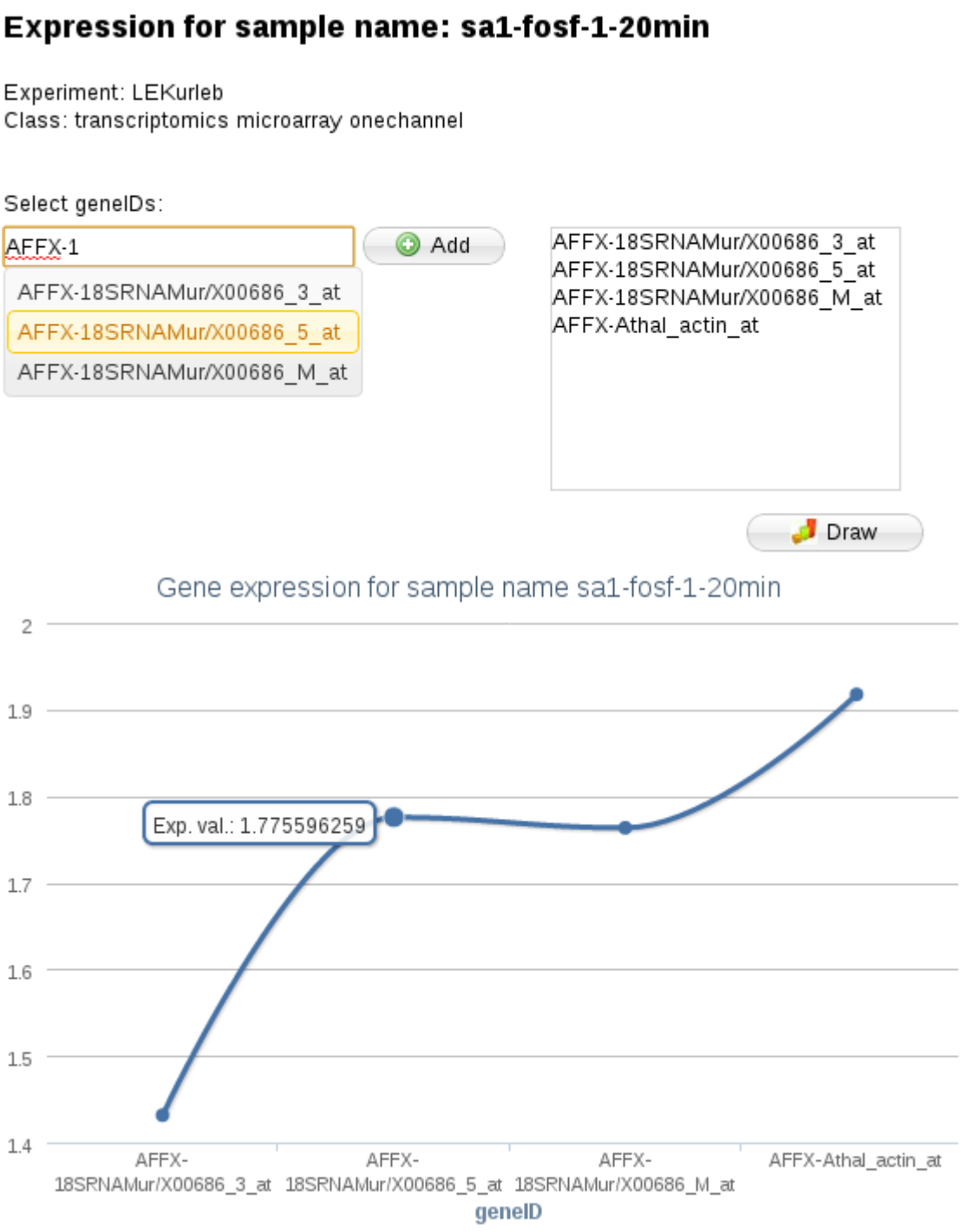


Slika 4.9: Izraženost gena STMIN19 po okužbi vzorcev slovenske sorte krompirja igor z virusom s časovnimi točkami 30 minut, 12 ur in 48 ur.

Izris izraženosti vzorca omogoča iskani niz, ki se nahaja v vrsticah datoteke *Phenodata* (3.1.1). Datoteka *Phenodata* poleg prvega, drugega in poljubnih dodatnih stolpcev v drugem stolpcu `Sample name` prikazuje imena vzorcev, uporabljenih v poskusu. Če se iskani niz nahaja v vrstici datoteke *Phenodata*, se poleg prikaza celotne vsebine vrstice v zavihku `Phenodata` v zavihku `Sample name` prikaže tudi ime vzorca, ki pripada najdeni vrstici, kar prikazuje slika 4.10. Ime vzorca določa vrstico datoteke. Ko vzorcu pripada več datotek *Rawdata*, ime vzorca določa več vrstic datoteke *Phenodata*. Zavihek `Sample name` zato vsebuje edinstvena imena vzorcev. Preostane nam še izbira genov, katerih izraženost nas zanima, in tako izpolnjujemo vse pogoje za izris. Slika 4.11 prikazuje izraženost vzorca `sa1-fosf-1-20min` za štiri različne gene. Datoteka *Normalized* vedno vsebuje edinstvena imena vzorcev, ki so definirana v datoteki *Phenodata*. Ni potrebno skrbeti, da se najdeno ime vzorca ne nahaja v datoteki *Normalized*.



Slika 4.10: Prikaz rezultatov zavihka Sample_Name.



Slika 4.11: Izraženost vzorca sa1-fosf-1-20min pri različnih genih.

4.4 Možne razširitve

NIB poleg poskusov z DNA-mikromrežami opravlja veliko poskusov qPCR (quantitative polymerase chain reaction), v zadnjem času pa tudi poskuse RNASeq (quantitative RNA sequencing), proteinsko kvantitativno profiliranje in metabolno kvantitativno profiliranje. Veliko analiz qPCR se opravi tudi na področju gensko spremenjenih organizmov (GSO), kjer se preverja dovoljena količina GSO v hrani in krmi (`http://www.nib.si/oddelki/oddelek-za-genetsko-toksikologijo-in-biologijo-raka`). V želji po enotnem postopku računanja kontrolnih vrednosti rezultatov poskusa smo razvili spletno aplikacijo qPCR Calculator. Aplikacija kot rezultat ponuja več numeričnih vrednosti, namenjenih kontroli vmesnih vrednosti. Vrednost, ki jo interpretiramo kot končni rezultat, predstavlja polje `Final result`. `Final result` v kombinaciji z vzorci in imenom gena uporabimo kot datoteko *Normalized* (3.1.4). Iz datoteke *Normalized* izpeljemo datoteke *Phenodata*, *Annotation*, *Metaresult* in *Result*. Poskus qPCR prav tako kot poskus z DNA-mikromrežo izvedemo s pomočjo namenskega stroja. Stroj qPCR nam v kombinaciji z njegovim namenskim računalniškim programom vrne neobdelane rezultate poskusa v obliki datoteke, ki jo lahko uporabimo kot datoteko *Rawdata*. Tako poskus qPCR opišemo z enako množico datotek kot poskus z DNA-mikromrežo. Pri vseh ostalih tipih poskusov pa je oblika datotek, z izjemo datotek *Rawdata*, enaka kot pri DNA-mikromrežah. V prihodnosti imamo tako možnost razvoja podpore novim poskusom.

Izraženost gena lahko razširimo z izraženostjo enakih genov med različnimi poskusi. Tako primerjamo izraženost dveh enakih genov pri poskusih z DNA-mikromrežo in qPCR. Za izraženost enako poimenovanih vzorcev med različnimi poskusi pa ne moremo privzeti ali enako ime vzorca predstavlja enak rastlinski material. Dva raziskovalca na primer v svojih poskusih tako poimenujeta vzorec krompirja Nadine-30min enako, kar predstavlja časovno točko 30 minut, pri čemer vzorca nista pridobljena iz enakega rastlinskega materiala. Enoznačno poimenovanje vzorcev bi bilo za razširitev nabora možnih analiz potrebno uvesti, vendar nujno v dogovoru z razisko-

valci, ki podatke proizvajajo.

Poglavje 5

Sklepne ugotovitve

V okviru diplomskega dela smo razvili spletno aplikacijo Fitobase, ki omogoča opis poskusov z DNA-mikromrežo in iskanje po datotekah poskusov. Pri načrtovanju smo upoštevali možnost velikega števila poskusov, ki jih aplikacija shranjuje in s tem povezano zagotavljanje učinkovitega iskanja. Hkrati je pomembna preprosta in kvalitetna predstavitev rezultatov iskanja, saj je le-ta ključnega pomena za uporabnika. Aplikacija za svoje delovanje potrebujejo množico orodij (2), katera so skupaj z aplikacijo testirana na operacijskem sistemu Linux. S podporo orodij na različnih platformah, lahko aplikacija deluje tudi na operacijskem sistemu Windows. Aplikacija je trenutno nameščena na namenskem spletnem strežniku Nacionalnega inštituta za biologijo, ki ga poganja distribucija Linux Fedora (`http://fedoraproject.org`). Trenutno shranjujemo tri poskuse z DNA-mikromrežo. Aplikacija je v fazi testiranja. Testira jo nekaj raziskovalcev iz oddelka za biotehnologijo in sistemsko biologijo (`http://www.nib.si/oddelki/oddelek-za-biotehnologijo-in-sistemsko-biologijo`).

Po vnosu poskusa ne potrebujemo administratorja ali nadzornika, ki bi preverjal pravilnost poskusa in zagnal proces validacije datotek poskusa. Proces validacije datotek zažene orodje cron s poljubno časovno periodo (A). Ob neupoštevanju oblik datotek, proces validacije zabeleži napake datotek poskusa (B). Ob velikih količinah poskusov, le-ti zavzamejo veliko prostora na

disku. V prihodnosti bi bilo smiselno razviti sistem obveščanja zasedenosti diska s strani aplikacije Fitobase. Aplikacijo smo tako načrtovali z minimalnim delom vzdrževanja.

V zadnjem času se vse bolj uporablja tehnologija qPCR, ki postaja standard za detekcijo in kvantifikacijo DNA in RNA. S pomočjo razvite aplikacije qPCR Calculator (4.4) prenesemo končne rezultate v aplikacijo Fitobase ter tako dodamo podporo novemu poskusu qPCR.

Literatura

[1] J. Demšar, "Python za programerje", Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, Ljubljana, 2009.

[2] D. Hrvaćanin, "Primerjava arhitektur Model-Pogled-Kontroler", Diplomsko delo, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, Ljubljana, 2012.

[3] D. Hunter, J. Rafter, J. Fawcett, Eric van der Vlist, D. Ayers, J. Duckett, A. Watt, L. McKinnon, "Beginning XML 4th Edition", Wrox, 2007.

[4] M. Petek, "Interakcije med krompirjem (Solanum tuberosum L.), krompirjevim virusom Y (PVY) in koloradskim hroščem (Leptinotarsa decemlineata Say) na molekularnem nivoju", Doktorska disertacija, Biotehniška fakulteta, Univerza v Ljubljani, Ljubljana, 2012.

[5] M.Petek, "Uporaba metabolnih mrež pri globalni analizi transkriptoma bakterije Staphylococcus aureus", Diplomsko delo, Fakulteta za farmacijo, Univerza v Ljubljani, 2007 .

[6] (2013) "General information". Dostopno na: `http://php.net/manual/en/faq.general.php`.

[7] (2013) "Loxun". Dostopno na: `https://pypi.python.org/pypi/loxun/`.

[8] (2013) "Understanding Model-View-Controller - CakePHP Cookbook v2.x documentation". Dostopno na: `http://book.cakephp.org/2.`

0/en/cakephp-overview/understanding-model-view-controller.html.

Dodatek A

Vzajemno izključevanje procesa validacije datotek poskusa

Ko poskus preide v stanje `procesiranje`, čaka na validacijo datotek. Proces validacije se sproži s časovno periodo 30 minut, z uporabo Linux orodja cron (`http://en.opensuse.org/SDB:Cron`). Konfiguracijska datoteka orodja cron za zagon skripte bash `check_exp.sh` s časovno periodo 30 minut je prikazana spodaj.

```
# min  hour  day of month  month  day of week  command
*/30   *     *             *      *            /path/check_exp.sh
```

V primeru, da proces validacije traja več kot 30 minut, se ustvari še ena instanca skripte bash. V skripti bash moramo poskrbeti, da se vedno izvaja le ena instanca skripte, saj drugače tvegamo možnost večkratne validacije enega poskusa in ponoven vnos izraženosti genov v podatkovno bazo MySQL ter vsebino datotek *Phenodata*, *Annotation* in *Result* v indeks odprtokodnega iskalnika teksta Sphinx. Enkratno validacijo enega poskusa in nepodvojeno vsebino datotek poskusa dosežemo z vzajemnim izključevanjem procesa validacije, ki jo omogoča uporaba ukaza `flock` (Linux man pages). Vsebina skripte bash je prikazana spodaj.

```
#!/bin/bash

set -e

# -e  errexit
# Exit immediately if a simple command exits with
# a non-zero status, unless the command that fails
# is part of an until or while loop, part of an if
# statement, part of a && or || list, or if the
# command's return status is being inverted using !.


(
  flock -e -n 200

  # if there is a least one experiment
  # validation ok, we proceed to next line (sphinx index)
  #
  python validator.py

  # build sphinx index with new data
  #
  /usr/local/bin/indexer --all --rotate

) 200>script.lock

rm -f script.lock
```

Dodatek B

Validacija datoteke Phenodata

```
def _phenodata(self, path):

  # Example phenoData format:
  #
  # Filename            Sample name     sampleNo  group dpi
  # des_moc_b-03-1.txt  des_moc_b-03-1  4         des_b03m  3
  # des_moc_b-03-2.txt  des_moc_b-03-2  5         des_b03m  3
  #
  # sample names are used in normalizedData,
  # filenames are used in rawData
  #
  self.sample_names = set()
  self.filenames    = set()

  try:
    f = open(path + os.listdir(path)[0], 'rbU')
  except:
    return {0: 'Can not open phenoData.'}

  error_data  = {}
  line_number = 1

  # header line, Filename \t Sample name + any custom data
  line = f.readline()
```

```
try:
  line = line.decode('utf-8', 'strict')
except UnicodeDecodeError:
  f.close()
  return {1: 'Invalid utf-8 string.'}

if re.search('^Filename\tSample name(\t.*)?$', line) is None:
  f.close()
  return {1: 'Header line: Filename \t Sample name not found.'}

line_number = 1

try:
  for line in f:
    line_number += 1

    # valid utf-8
    try:
      line = line.decode('utf-8', 'strict')
    except UnicodeDecodeError:
      error_data[line_number] = 'Invalid utf-8 string.'
      return error_data

    # if error count is greater than 20 return error message
    if len(error_data) > 20:
      return error_data

    line = line.strip(' \n\r\t')
    data = line.split('\t')

    # line must be splited with at least 1 tab
    if len(data) <= 1:
      error_data[line_number] = 'Line must be splitted
                                 with at least 1 tab.'
      continue

    filename    = data[0]
    sample_name = data[1]
```

```
    # check if filename is valid
    if re.search('^([a-zA-Z0-9\-\_]+)((\.[a-zA-Z0-9]+)?)$', filename)
    is None:
      error_data[line_number] = 'Filename: ' + filename[:512] +
                                ' is invalid. Filename can consist
                                 only of letters a-z, A-Z, 0-9, - or _.
                                 No spaces are allowed.';
      continue

    # sample name must be less than 64 character in length
    if len(sample_name) > 64:
      error_data[line_number] = 'Sample name: ' + sample_name[:512] +
                                ' Sample name must be
                                less than 65 characters.'
      continue

    # check if sample name is valid
    if re.search('^[a-zA-Z0-9\-\_]+$', sample_name) is None:
      error_data[line_number] = 'Sample name: ' + sample_name[:512] +
                                ' Sample name can consist
                                 only of letters a-z, A-Z, 0-9, - or _.
                                 No spaces are allowed.';
      continue

    # add sample name
    self.sample_names.add(sample_name)

    # add filename
    self.filenames.add(filename)

finally:
  f.close()

if len(error_data) == 0:
  return True

return error_data
```