SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD

Prilagodljivi programski sustav za simboličku regresiju

Domagoj Stanković

Voditelj: *izv. prof. dr. sc. Domagoj Jakobović*

Zagreb, srpanj, 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 9. ožujka 2016.

Predmet: **Diplomski rad**

DIPLOMSKI ZADATAK br. 1227

Pristupnik: **Domagoj Stanković (0036467862)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Prilagodljivi programski sustav za simboličku regresiju**

Opis zadatka:

Opisati metodu simboličke regresije i primjenu u problemima strojnog učenja. Istražiti postojeće načine prikaza rješenja i tehnike poboljšavanja učinkovitosti i kvalitete modela strojnog učenja. Ostvariti programski sustav za simboličku regresiju uz pojednostavljeno zadavanje parametara problema. Omogućiti dinamičku prilagodljivost sustava uz pomoć odgovarajućeg korisničkog sučelja. Ispitati učinkovitost ostvarenog sustava na dostupnim problemima simboličke regresije, s obzirom na različite tehnike poboljšanja. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 18. ožujka 2016.
Rok za predaju rada: 1. srpnja 2016.

Mentor:

Izv. prof. dr. sc. Domagoj Jakobović

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

Zahvaljujem svom mentoru, prof. dr. sc. Domagoju Jakoboviću, na izuzetnom strpljenju, razumijevanju i podršci.

Također velika hvala mojoj obitelji na bezuvjetnom odricanju, potpori, utjesi i motivaciji. Hvala vam što ste uvijek bili uz mene kada sam vas trebao.

Sadržaj

1. Uvod

1.1 Evolucijsko računanje

Unatoč tome što su računala danas nevjerojatno brza, u praksi često nailazimo na probleme koje nije moguće riješiti tehnikom grube sile (engl. *brute-force*) tj. pretraživanjem cjelokupnog prostora rješenja. Ti problemi poznati su kao NP-potpuni i NP-teški problemi. Takvi su problemi netraktabilni tj. za takve probleme ne postoje algoritmi čija je složenost zadovoljavajuća. Primjeri takvih problema su problem trgovačkog putnika, problem izrade rasporeda predavanja i raspoređivanja studenata u grupe, problem bojanja grafova, problem raspoređivanja medicinskih sestara u smjene itd.

Kako često nije potrebno pronaći optimalno već dovoljno dobro rješenje, postoje brojni algoritmi koji nam u tome pomažu, a imaju nisku računsku složenost. Takvi algoritmi nazivaju se heuristički algoritmi, ili jednostavnije, heuristike. U današnje doba posebno su nam zanimljive metaheuristike. Metaheuristika je skup algoritamskih koncepata koji koristimo za definiranje heurističkih metoda primjenjivih na širok skup problema. Možemo reći da je metaheuristika heuristika opće namjene čiji je zadatak usmjeravanje problemski specifičnih heuristika prema području prostora rješenja u kojem se nalaze dobra rješenja [1]. Primjeri su metaheuristika algoritam simuliranog kaljenja, genetski algoritmi, genetsko programiranje, algoritam roja čestica, algoritam diferencijske evolucije itd.

Uz priču o algoritmima pretraživanja važno je spomenuti teorem *no-free-lunch* koji kaže da su svi algoritmi pretraživanja prosječno jednako dobri. Za jednu vrstu problema određeni algoritam ponašat će se bolje u odnosu na neki drugi algoritam dok će za drugi skup problema vrijediti obrnuto. Iz toga slijedi zaključak da je jako važno dobro odabrati algoritam kojim će se rješavati određeni problem jer bi u suprotnom i običan algoritam slijepe pretrage mogao davati bolje rezultate.

1.2 ECF

ECF je postojeći programski okvir za evolucijsko računanje napisan u programskom jeziku C++. Razvijan je kroz niz godina te sadrži brojne funkcionalnosti potrebne za rješavanje mnogih optimizacijskih problema algoritmima evolucijskog računanja. ECF se dosad pokretao isključivo preko konzole uz ručno pisanje konfiguracijskih datoteka što je bilo poprilično naporno uz veliku mogućnost pogreške. Uz to, rezultati dobiveni radom algoritama bili su u tekstualnom obliku te nisu bili vizualizirani. Iz tih razloga javila se potreba za izradom korisničkog sučelja kojim će se na jednostavan način upravljati ECF-om. Zamisao je da sučelje upravlja programskim okvirom neovisno o problemu, algoritmu, genotipu i ostalim parametrima.

Optimizacijski problem definiran je kroz izvršnu datoteku dobivenu prevođenjem programa koji koristi biblioteku ECF. Iz te je datoteke moguće, uz pokretanje s određenim parametrima, dobiti datoteku s popisom svih algoritama,

genotipa i ostalih parametara koje nudi ECF. Na taj način grafičko sučelje zna kako prikazati i ponuditi korisniku sve opcije koje nudi ECF.

Algoritam [2] je konačan slijed dobro definiranih naredbi za ostvarenje zadatka, koji će za dano početno stanje završiti u definiranom konačnom stanju. Algoritam ustvari određuje način na koji će se zadani problem rješavati. Primjeri algoritama evolucijskog računanja su: algoritam diferencijske evolucije, algoritam kolonije mrava i genetski algoritam.

Genotip označava jedno rješenje optimizacijskog problema i posjeduje mjeru dobrote, tj. numeričku vrijednost koja označava u kolikoj mjeri rješenje zadovoljava određeni problem. Cilj je evolucijskog računanja pronaći rješenje sa što većom mogućom dobrotom, ali ne postoji garancija da će algoritam pronaći optimalno rješenje. Postoje različite vrste genotipa kao što su genotip u obliku stabla, genotip u obliku niza bitova i genotip u obliku polja decimalnih brojeva.

Parametri definiraju dodatne podatke o algoritmu odnosno problemu. Algoritmi i problemi predstavljaju općenite principe, dok parametrima definiramo konkretni algoritam odnosno problem. Tako je *pokus* definiran kao skup algoritma i problema s pripadajućim parametrima. Primjeri parametara su: veličina populacije, najveći dopušteni broj iteracija i selekcijski pritisak.

Algoritmi, genotipi i ostali parametri koji će se koristiti u pokusu definirani su kroz konfiguracijsku datoteku u XML obliku. Ta se datoteka predaje ECF-u koji ju čita te oblikuje sve potrebno za izvođenje evolucijskog računanja. Konfiguracijska se datoteka sastoji od skupa algoritama, skupa genotipa te skupa dodatnih parametara. Za svaki algoritam i genotip naveden je niz pripadnih parametara.

Rezultat rada ECF-a odnosno rezultat pokretanja pokusa tekstualna je datoteka u kojoj su zapisane neke osnovne značajke stanja dobrote populacije po generacijama.

Korisničko sučelje za rukovanje ECF-om započeto je u okviru kolegija Preddiplomski projekt koji sam radio u suradnji s kolegama Vlahom Polutom i Svenom Vidakom. Kolege su bile zadužene za dio koji je obuhvaćao parsiranje potrebnih dokumenata koji nastaju tokom rada ECF-a te za komunikaciju korisničkog sučelja s procesom ECF-a. Nad tim je komponentama potom izgrađeno grafičko sučelje.

Ovaj diplomski rad sastoji se od 3 dijela: dijela za olakšavanje pokretanja generičkih pokusa koji koriste ECF radni okvir, dijela za unapređivanje postupka simboličke regresije i dijela koji olakšava pokretanje pokusa simboličke regresije.

2. ECF Lab

ECF Lab je *desktop* aplikacija nastala kao rezultat prethodno iznesenih težnji. Aplikacija je napisana u programskom jeziku Java korištenjem alata Swing tako da je neovisna o platformi i operacijskom sustavu. Korištena je mogućnost *Look and Feel* kako bi aplikacija poprimila izgled autohtone aplikacije (engl. *native*).

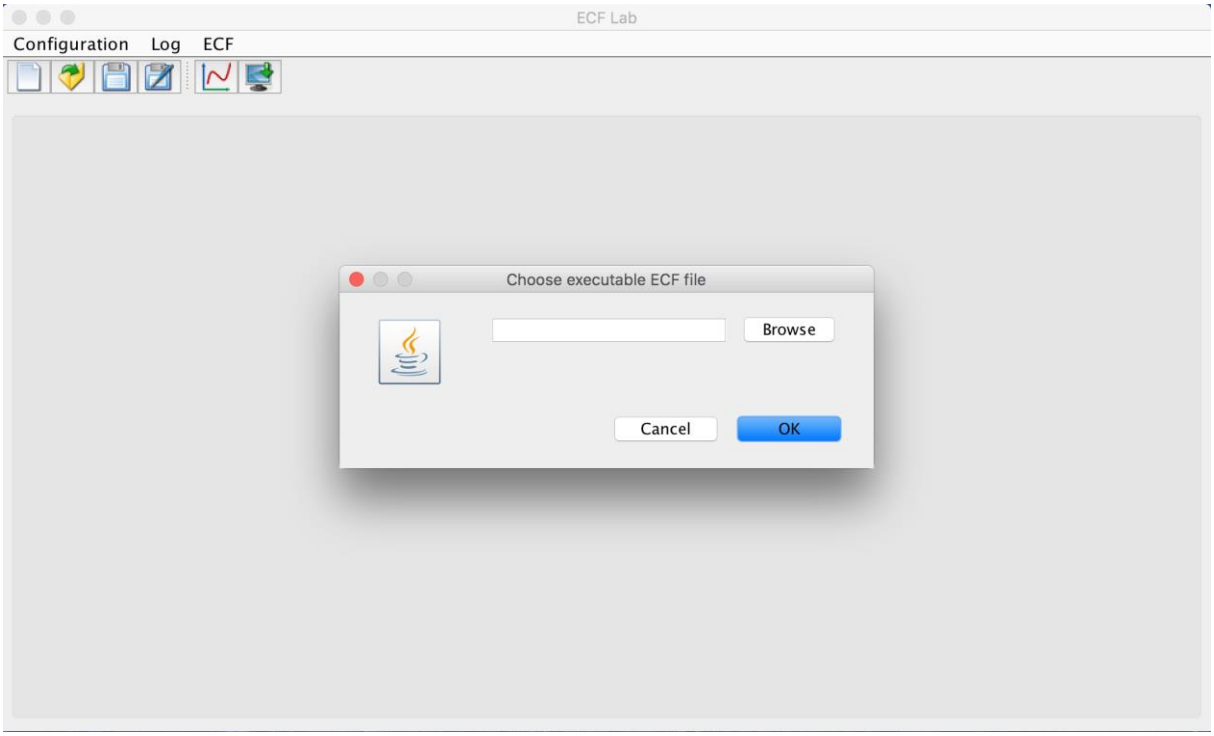
Aplikacija se sastoji od 3 glavna dijela: dijela za komunikaciju s procesom ECF-a, dijela za parsiranje i generiranje svih potrebnih datoteka te grafičkog korisničkog sučelja. Kako je ECF napisan u programskom jeziku C++, a grafičko sučelje u programskom jeziku Java, komunikacija među tim komponentama ostvaruje se tako da grafičko sučelje pokreće proces ECF-a preko konzole. Proces ECF-a pokretat će se iz 2 razloga, za dobivanje popisa svih dostupnih opcija te za izvođenje pokusa. Za izvođenje se pri stvaranju procesa odredi put do konfiguracijske datoteke te mjesto gdje će se spremiti rezultat izvođenja. Nakon gašenja procesa jednostavno se pročitaju podaci koji se nalaze na dogovorenom mjestu.

2.1 Korištenje aplikacije ECF Lab

2.1.1 Početni zaslon

Pri pokretanju aplikacije omogućen je odabir izvršne datoteke nad kojom će se izvršavati željene operacije. Nakon odabira, *ECF Lab* će zatražiti od izvršne datoteke ispis svih algoritama, genotipa i ostalih parametara kako bi ih znao prikazati korisniku.

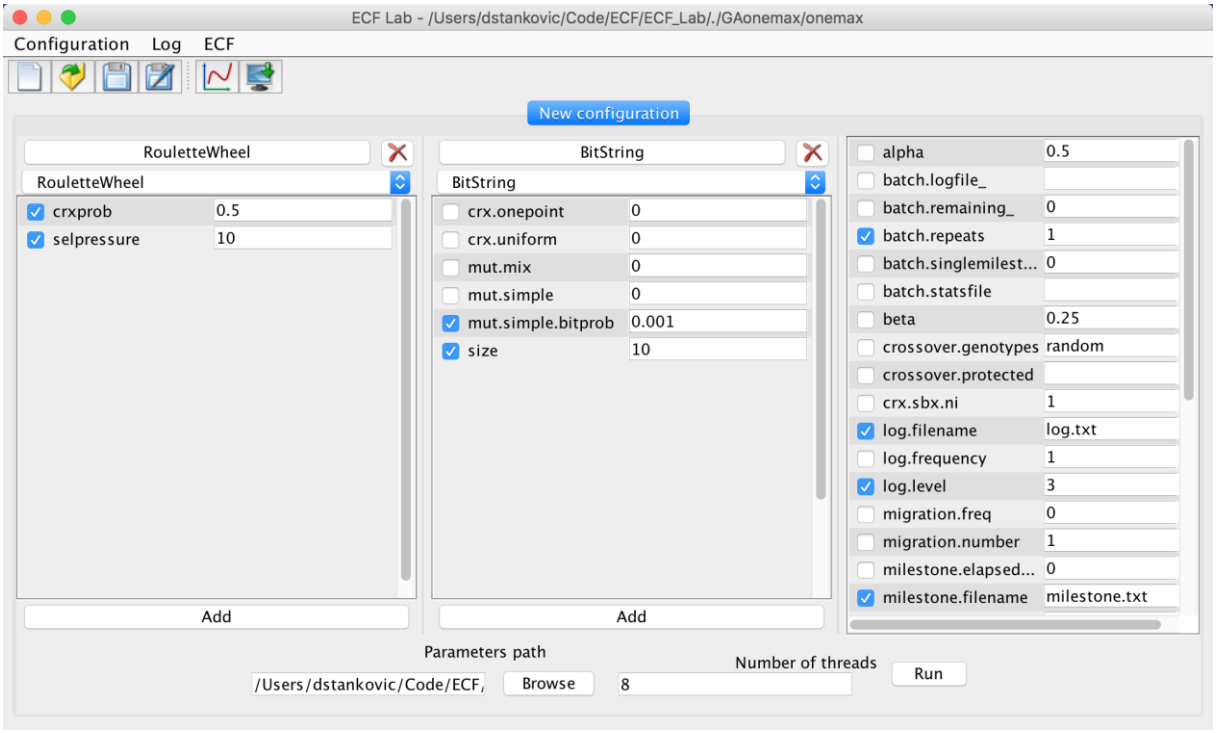
Nakon toga korisniku se nude mogućnosti vezane za konfiguraciju, rezultate i izvršnu datoteku te dodatne informacije. Bilo koju od mogućnosti moguće je odabrati preko padajućeg izbornika, alatne trake ili tipkovničke prečice. Prelaskom pokazivača miša preko stavke u određenoj traci prikazuje se i opis određene akcije. Stvaranje i otvaranje konfiguracijske datoteke otvaraju novu karticu.



Slika 2.1 Početni zaslon

2.1.2 Stvaranje nove konfiguracije

Kako bi izvođenje pokusa moglo početi, potrebno je navesti algoritme (*engl. algorithms*) i njihove parametre, genotipe (*engl. genotypes*) i njihove parametre te dodatne opcije (*engl. registry*). To se ostvaruje tako da se stvori nova konfiguracijska datoteka odabirom željenih algoritama i genotipa te unosom njihovih parametara. Parametri se unose označavanjem odgovarajuće kućice te upisom vrijednosti parametra. Obvezni su parametri već označeni i ne mogu biti odznačeni. Kako je omogućen unos više algoritama i genotipa, potrebno je pritisnuti gumb *Add* kako bi algoritam odnosno genotip bio upisan u konfiguracijsku datoteku. Nakon tog koraka moguće je unijeti idući algoritam odnosno genotip. U slučaju da želimo promijeniti neki od navedenih algoritama ili genotipa, potrebno je kliknuti na taj unos nakon čega će se otvoriti novi prozor u kojem će biti moguće mijenjati pojedine parametre. Za slučaj da želimo ukloniti određeni algoritam ili genotip, potrebno je pritisnuti gumb s crvenim znakom „X“ pokraj unosa. Konfiguracijska datoteka bit će spremljena tek kada se ili pokrene rad ECF-a ili odabere opcija za spremanje konfiguracije.



Slika 2.2 Stvaranje nove konfiguracijske datoteke

2.1.3 Otvaranje postojeće konfiguracijske datoteke

Ako je dostupna već postojeća konfiguracijska datoteka, ta se datoteka može učitati pomoću aplikacije kako bi se izmijenila ili iskoristila za pokretanje novog pokusa.

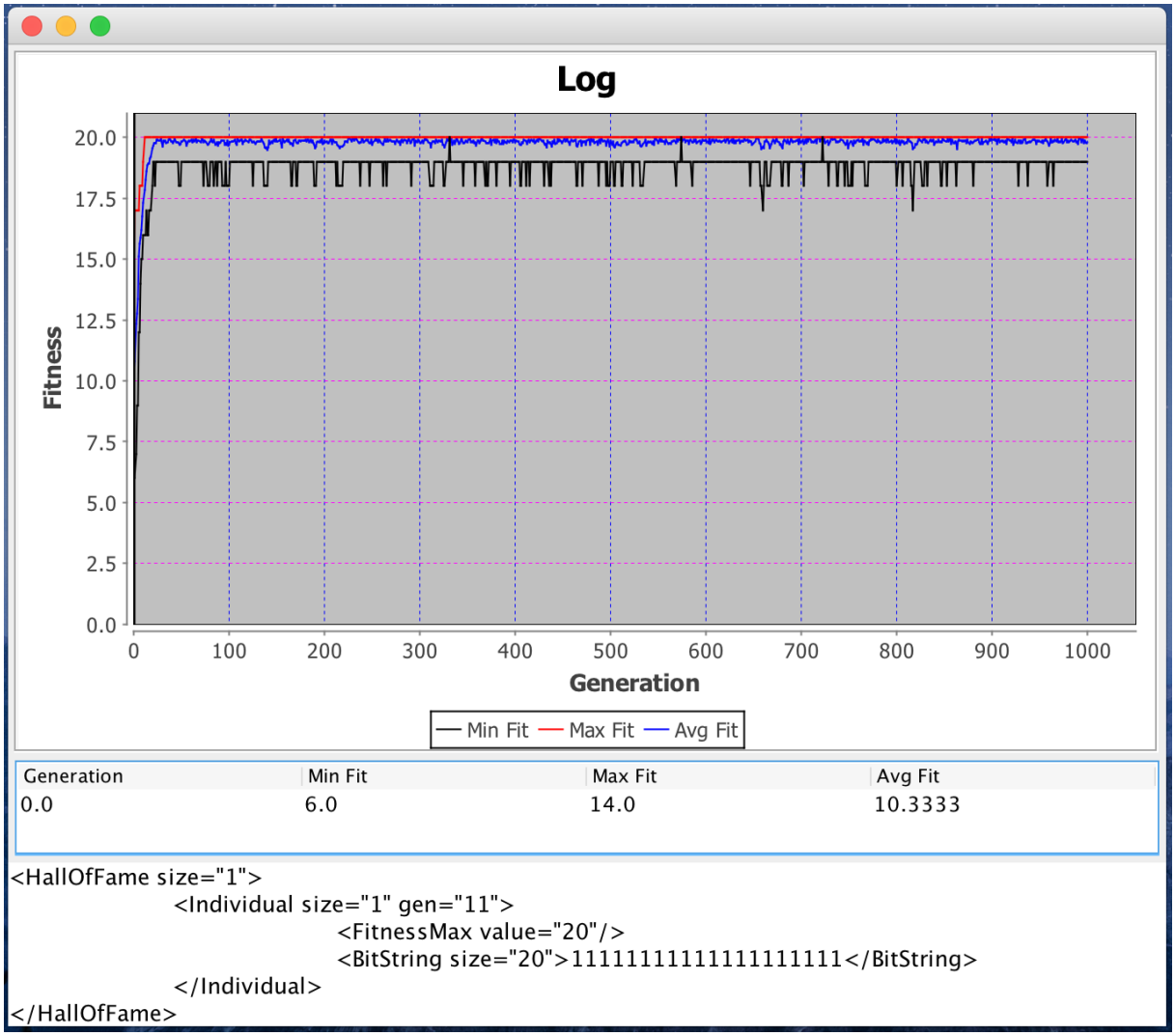
2.1.4 Spremanje trenutne konfiguracije

Trenutno odabrana konfiguracija može se spremiti u konfiguracijsku datoteku. Ako se odabere *Configuration -> Save,* konfiguracija će se spremiti pod imenom navedenim ispod u rubrici *Parameters path.* Taj put može se promijeniti pritiskom na gumb *Browse.* Nakon spremanja konfiguracije, naziv označene kartice pretvorit će se u put pod kojim je ta konfiguracija spremljena. Ako se odabere *Configuration -> Save As,* iskočit će izbornik za odabir mjesta za spremanje konfiguracijske datoteke. Spremanje na taj način neće utjecati na naziv kartice.

2.1.5 Otvaranje rezultata

U okviru aplikacije *ECF Lab* moguće je otvoriti datoteke u kojima su zapisani rezultati izvođenja ECF-a. To se ostvaruje odabirom *Log -> Open* nakon čega se odabire put do željene datoteke. Nakon što se odabrana datoteka isparsira, pojavljuje

se novi prozor koji prikazuje graf dobrote populacije kroz generacije. Vodoravna os predstavlja broj generacije, a okomita dobrotu. Uz graf se nalazi i legenda koja govori koja linija prikazuje maksimalnu, koja minimalnu, a koja srednju dobrotu. Lijevim klikom na graf pojavljuje se okomita crna linija koja označava generaciju i siječe linije koje predstavljaju dobrotu. Kako je na taj način teško očitati koje su vrijednosti dobrote za odabranu generaciju, ispod grafa nalazi se i tablica u kojoj pišu odgovarajuće vrijednosti za odabranu generaciju. Ispod tablice nalazi se i prikaz najboljeg rješenja (engl. *Hall of fame*). Desnim klikom na graf moguće je odabrati još neke dodatne opcije kao što su zumiranje, mijenjanje svojstava grafa, kopiranje slike grafa te spremanje ili ispis iste. Zumiranje je također moguće ostvariti označavanjem dijela grafa koji se želi zumirati.



Slika 2.3 Vizualizacija rezultata pokusa

2.1.6 Mijenjanje izvršne datoteke

Tijekom rada aplikacije *ECF Lab* moguće je i promijeniti izvršnu datoteku nad kojom će se izvoditi pokusi. To se obavlja odabirom opcije *ECF -> Change ECF* nakon čega se otvara prozor u kojem je potrebno odabrati novu izvršnu datoteku. Nakon toga će *ECF Lab* pozvati ispis svih algoritama, genotipa i ostalih opcija nad novom izvršnom datotekom. Ti će podaci biti vidljivi tek pri otvaranju nove kartice. Na vrhu prozora, iznad izborničke trake, uz ime aplikacije naveden je i put do izvršne datoteke koja se trenutno koristi.

2.1.7 *Web* stranica ECF projekta

Više informacija o ECF-u možete potražiti na službenoj *web* stranici projekta [3] ili iz *ECF Lab*-a jednostavnim pritiskom na opciju *ECF home page* iz padajućeg izbornika *ECF.* Odabirom te opcije, otvorit će se *web* preglednik s odgovarajućom stranicom.

2.1.8 Pokretanje pokusa

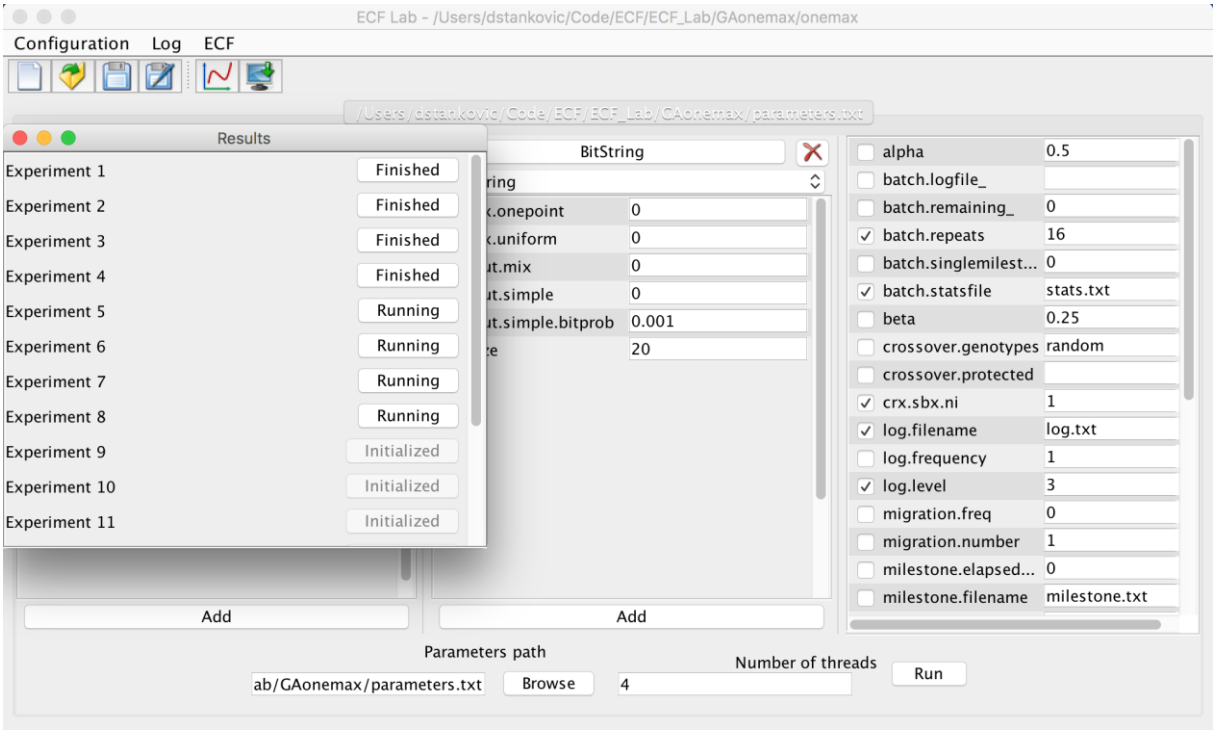
Nakon što se odaberu svi željeni algoritmi i genotipi te se definiraju svi potrebni parametri, moguće je pokrenuti izvođenje pokusa. Lijevo od gumba *Run* nalaze se 2 rubrike: rubrika za odabir mjesta gdje će se spremiti konfiguracijska datoteka i rubrika za odabir broja dretvi koje će izvoditi zadani pokus.

U dodatnim parametrima postoji opcija pod imenom „batch.repeats“ koja govori koliko puta zaredom će se pokus izvršiti. Kada je „batch.repeats“ označen i vrijednost mu je veća od 1, ECF će automatski dodavati broj ponavljanja u ime datoteke ispred ekstenzije, npr. log.txt postat će log_XX.txt gdje XX označava broj ponavljanja. *ECF Lab* će dočekati sve datoteke nastale na taj način te omogućiti prikazivanje istih. U slučaju da je označeno više od jedne dretve, *ECF Lab* će promijeniti konfiguraciju kako bi se omogućilo paralelno izvođenje pokusa. „batch.repeats“ će se postaviti na 1 te će se stvoriti onoliko poslova koliko je ponavljanja bilo označeno. Poslovi će se staviti u red izvođenja koje će izvoditi bazen dretvi (engl. *thread pool*) sačinjen od onoliko dretvi koliko je bilo označeno u odgovarajućoj rubrici. U svim će se ostalim slučajevima broj dretvi automatski postaviti na 1 jer nema potrebe za paralelizacijom.

U dodatnim parametrima također se nalazi opcija „batch.statsfile“ koja određuje datoteku u koju će se spremati statistika o pokrenutim pokusima kao što su minimalna i maksimalna mjera dobrote, utrošeno vrijeme, broj evaluacija itd. ECF izrađuje tu statistiku i zapisuje je u zadanu datoteku kada je „batch.repeats“ veći od 1. Pošto ECF Lab za potrebe paralelnog izvođenja razdvaja konfiguracijsku datoteku, bilo je potrebno razdvojiti i *stats* datoteke za pojedine pokuse te ih sve nazad spojiti po završetku izvođenja svih pokusa. ECF Lab to automatski radi te korisnik o tome ne treba voditi računa.

Pokus se pokreće pritiskom na gumb „*Run“.* Nakon klika, ECF Lab će pokupiti sve definirane opcije iz trenutno aktivne kartice, zapisati ih u konfiguracijsku datoteku na zadano mjesto i pozvati komponentu koja vodi brigu o pokretanju pokusa. Ta će komponenta obavještavati grafičko sučelje o trenutnom stanju pokrenutih pokusa. Otvorit će se novi prozor u kojem će biti vidljivi svi pokusi pokrenuti za tu karticu. Uz svaki pokus stoji gumb na kojem piše trenutno stanje tog pokusa. Pokus može biti u 5 stanja: inicijaliziran (*engl. Initialized*), započet (*engl. Started*), u izvođenju (*engl. Running*), završen (*engl. Finished*) i neuspio (*engl. Failed*). Pokus je inicijaliziran kada se stvori zadatak koji će izvršiti zadani pokus, a započet tek kada se počne izvršavati. ECF Lab omogućava *online* čitanje rezultata pokusa tj. svake će se sekunde pročitati (nepotpuna) datoteka s rezultatima što omogućava korisniku pregled trenutnog napretka pojedinog pokusa. Sve dok ne završi, pokus je u stanju *Running*, a kad završi dolazi u stanje *Finished*. Ako se dogodi neka greška, pokus će doći u stanje *Failed*.

Dok je pokus u stanju *Running* ili *Finished*, moguće je kliknuti na gumb uz taj pokus i vidjeti vizualiziranu datoteku s rezultatima. Prozor s popisom svih rezultata za pojedinu karticu također se može otvoriti i odabirom opcije *Log -> Results*.

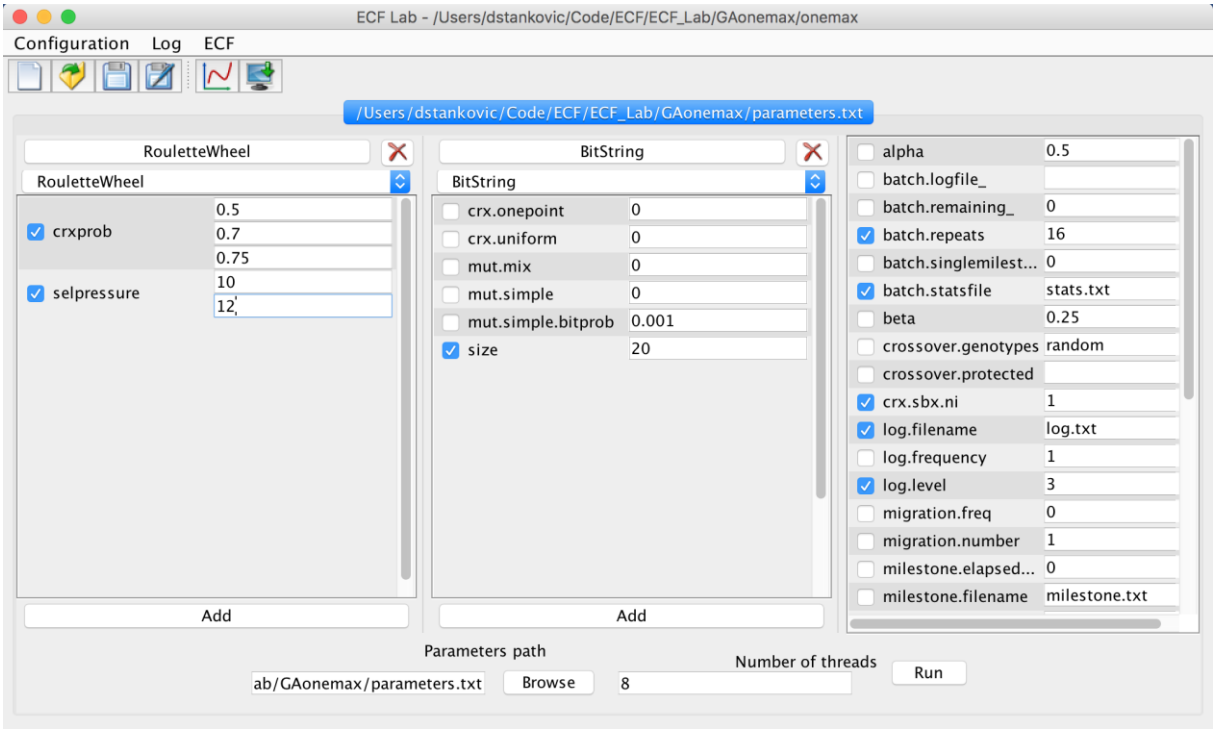


Slika 2.4 Pokretanje pokusa

2.1.9 Pokretanje skupa pokusa

Pošto često želimo pokrenuti više pokusa uz male izmjene vrijednosti nekih parametara morali bismo pojedinačno pokretati pokuse svaki sa svojim vrijednostima parametara te shodno tome mijenjati imena konfiguracijskih datoteka. ECF Lab nam

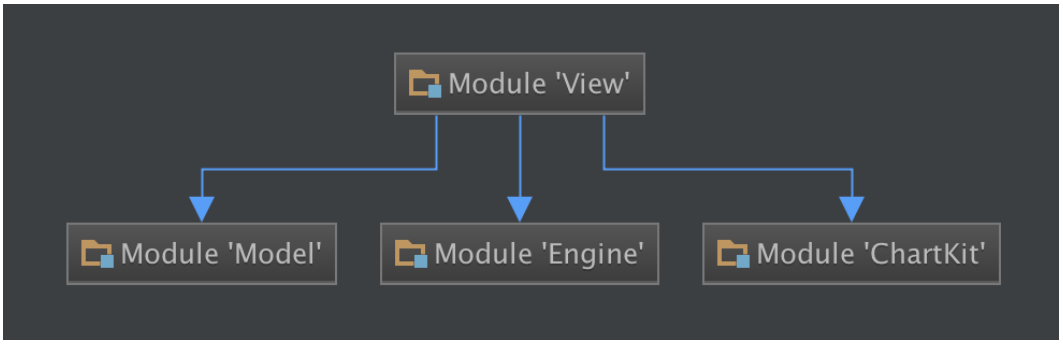
pomaže u tome i dopušta definiranje skupa vrijednosti za pojedine parametre. Potrebno je desnim klikom odabrati ime parametra te zatim pritisnuti *Add*. Uz postojeće polje otvorit će se novo polje u koje je moguće upisati dodatnu vrijednost. Moguće je definirati skup vrijednosti za više parametara te će se onda prilikom pokretanja napraviti kartezijev produkt svih parametara. Datoteke će automatski biti imenovane prema definiranim parametrima po predlošku log_param1-value1_param2-value2_paramN-valueN.txt. U slučaju da želimo obrisati suvišni unos, desnim klikom odaberemo željeno polje te kliknemo na *Remove*.



Slika 2.5 Definiranje skupa pokusa

2.2 Struktura aplikacije *ECF Lab*

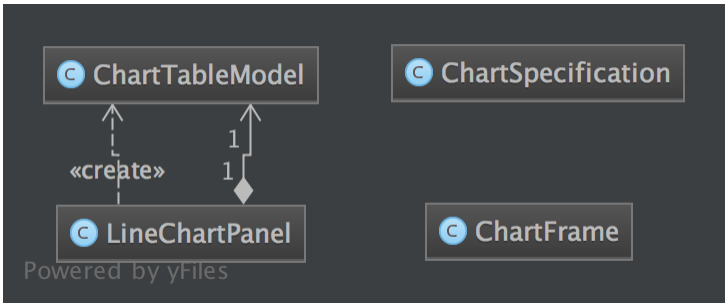
Aplikacija *ECF Lab* je projekt koji se sastoji od 4 modula: *ChartKit*, *Engine*, *Model* i *View*. *ChartKit* se brine o iscrtavanju grafova za prikaz rezultata pokusa, dok se *Engine* brine o samom pokretanju pokusa te čitanju i pisanju konfiguracijskih datoteka i datoteka s rezultatima. *Model* sadrži pomoćne razrede kao i razrede za bilježenje grešaka i učitavanje postavki. *View* sadrži razrede i logiku vezane za grafičko sučelje. Modul *View* glavni je modul koji koristi mogućnosti ostala 3 modula. Moduli su izdvojeni s ciljem ponovnog iskorištavanja koda i jasnije podjele odgovornosti. Modul *Engine* iskorišten je i u aplikaciji *SymReg Lab* koja također pokreće pokuse. Za automatizaciju razvojnog ciklusa korišten je alat *Gradle*. Svaki modul zajedno s vršnim projektom sadrži *gradle* skriptu u kojoj su definirane zavisnosti o drugim modulima i bibliotekama, opis izrade *jar* arhive i sl. Za izvršavanje *gradle* zadataka nije potrebno imati *gradle* instaliran jer postoji *Gradle Wrapper* koji nudi dvije skripte (jedna za operacijski sustav *Windows*, a druga za *Unix* porodicu) koje skidaju *gradle jar* s Interneta te ga pokreću uz željene opcije.



Slika 2.6 Struktura aplikacije ECF Lab

2.2.1 Modul *ChartKit*

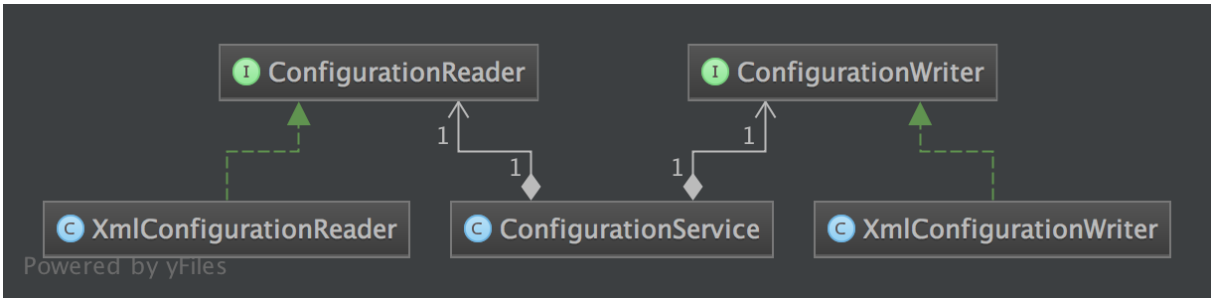
Modul *ChartKit* odgovoran je za vizualizaciju rezultata izvođenja pokusa. Kao pomoć koristi biblioteku *JFreeChart*[4] koja nudi gotova rješenja za iscrtavanje grafova. *ChartSpecification* sadrži podatke koje treba iscrtati, a *ChartTableModel* implementira model tablice koja će prikazivati točne statistike funkcije dobrote u određenoj generaciji nakon označavanja generacije na grafu. *LineChartPanel* sadrži logiku oko iscrtavanja grafa i osvježavanja tablice nakon klika po grafu. *ChartFrame* je samo prozor koji se pojavljuje kada treba prikazati rezultate, a sadrži *LineChartPanel* i prikaz najboljeg rješenja u tekstualnom obliku.



Slika 2.7 Modul ChartKit

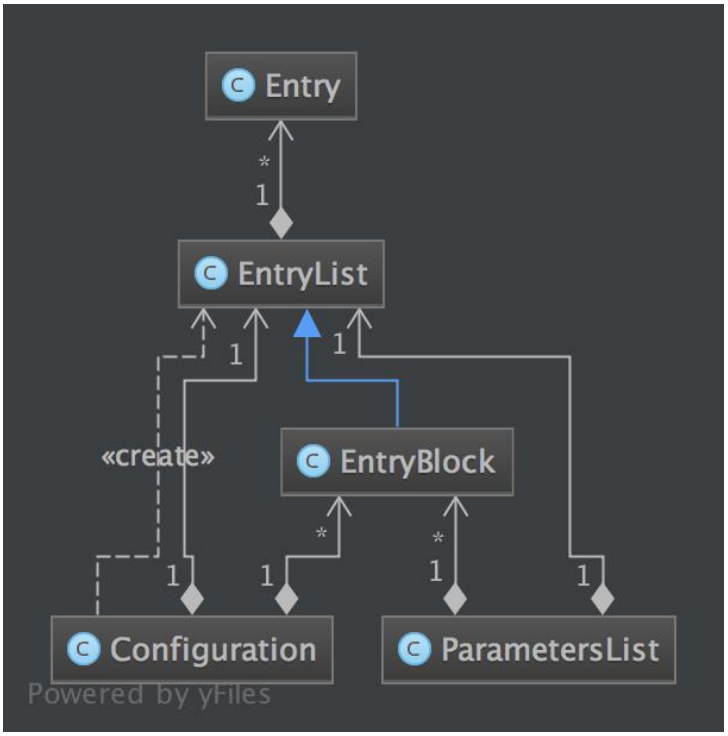
2.2.2 Modul *Engine*

Jedna od odgovornosti modula *Engine* je parsiranje konfiguracijskih datoteka i njihovo pisanje te čitanje skupa parametara koje nudi zadana izvršna ECF datoteka. Sučelja *ConfigurationReader* i *ConfigurationWriter* definiraju metode za čitanje i pisanje konfiguracijskih datoteka. Konfiguracijske su datoteke u XML formatu te razredi *XmlConfigurationReader* i *XmlConfigurationWriter* sadrže konkretne implementacije navedenih sučelja. *ConfigurationService* je jedinstveni objekt (*engl. Singleton*) koji odlučuje koje će se konkretno implementacije koristiti za slučaj da u budućnosti odlučimo promijeniti format konfiguracijske datoteke. Navedeni se razredi nalaze u paketu *conf*.



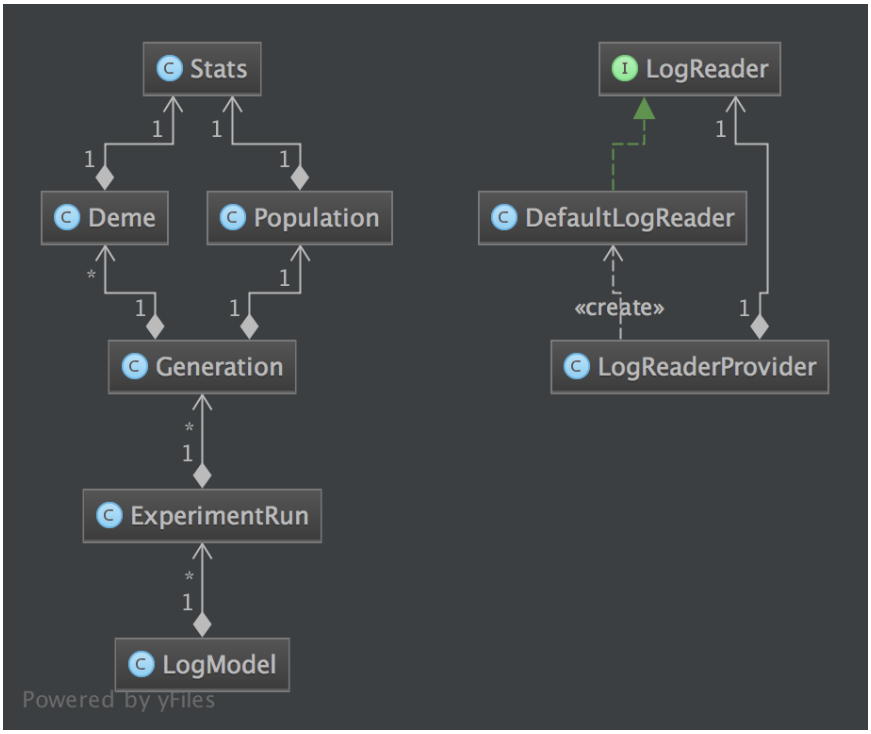
Slika 2.8 Engine - conf paket

U paketu *param* nalaze se razredi koji nastaju kao rezultat operacija iz paketa *conf*. Razred *Configuration* predstavlja opisnik konfiguracijske datoteke te se sastoji od liste algoritama, liste listi genotipa i liste dodatnih mogućnosti. *Entry* predstavlja pojedini parametar s njegovom vrijednosti. *EntryList* je lista *Entry*-ja, a *EntryBlock* je *EntryList* uz dodatan naziv. *EntryList* koristi se za listu dodatnih mogućnosti dok se *EntryBlock*-ovi koriste za zapis algoritama i genotipa. Razred *ParametersList* nastaje kao rezultat čitanja skupa parametara koje nudi izvršna ECF datoteka.



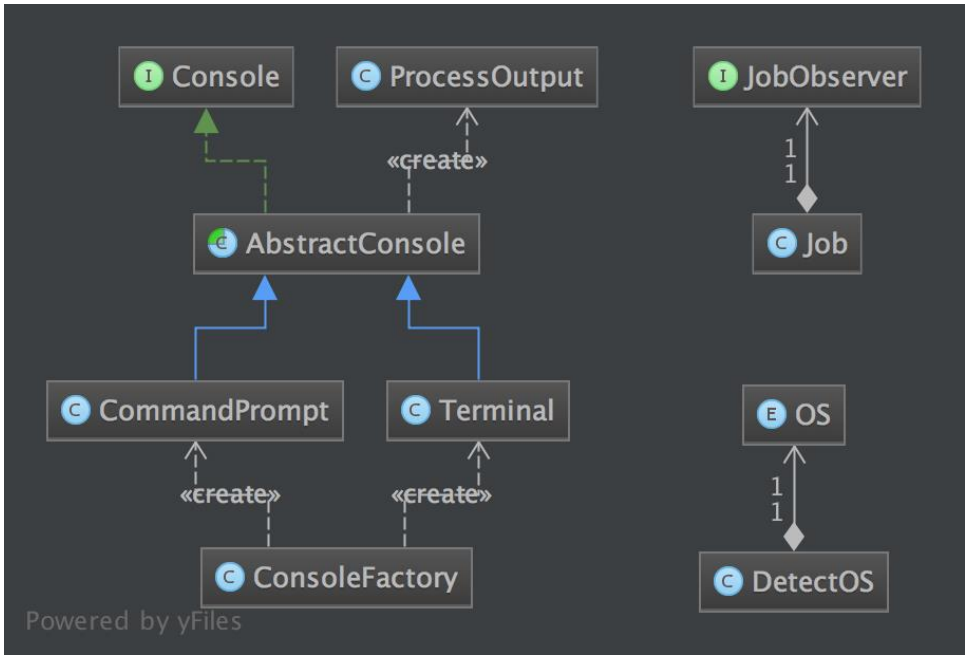
Slika 2.9 Engine - paket param

Modul *Engine* također se brine i za čitanje datoteka s rezultatima (*engl. Log files*) koje izvršna datoteka ECF-a izbaci tijekom izvođenja pokusa. Razredi zaduženi za to nalaze se u paketu *log*. *LogReader* je sučelje koje definira metode potrebne za čitanje datoteke s rezultatima dok je *DefaultLogReader* konkretna implementacija parsera za datoteke u trenutno zadanom formatu. *LogReaderProvider* je *singleton* koji dohvaća zadani parser. Rezultat parsiranja je razred *LogModel* koji sadrži ili listu *ExperimentRun*-ova ili opis pogreške u slučaju da je do pogreške došlo tijekom izvođenja pokusa. Razred *ExperimentRun* opisnik je pojedinog pokusa (svaka datoteka s rezultatima može sadržavati više pokretanja pokusa). Od bitnijih stvari, razred *ExperimentRun* sadrži listu generacija i opis najboljeg rješenja (*engl. Hall of Fame*). Svaka generacija (*engl. Generation*) sadrži identifikator, proteklo vrijeme, listu podpopulacija (*engl. Deme*), opisnik populacije (*engl. Population*) te opis najboljeg rješenja te generacije. I populacija i podpopulacija sadrže broj evaluacija i opisnik statistike (*engl. Stats*). Opisnik statistike funkcije dobrote sadrži vrijednost minimalne, maksimalne i prosječne dobrote te standardnu devijaciju.



Slika 2.10 Engine - paket log

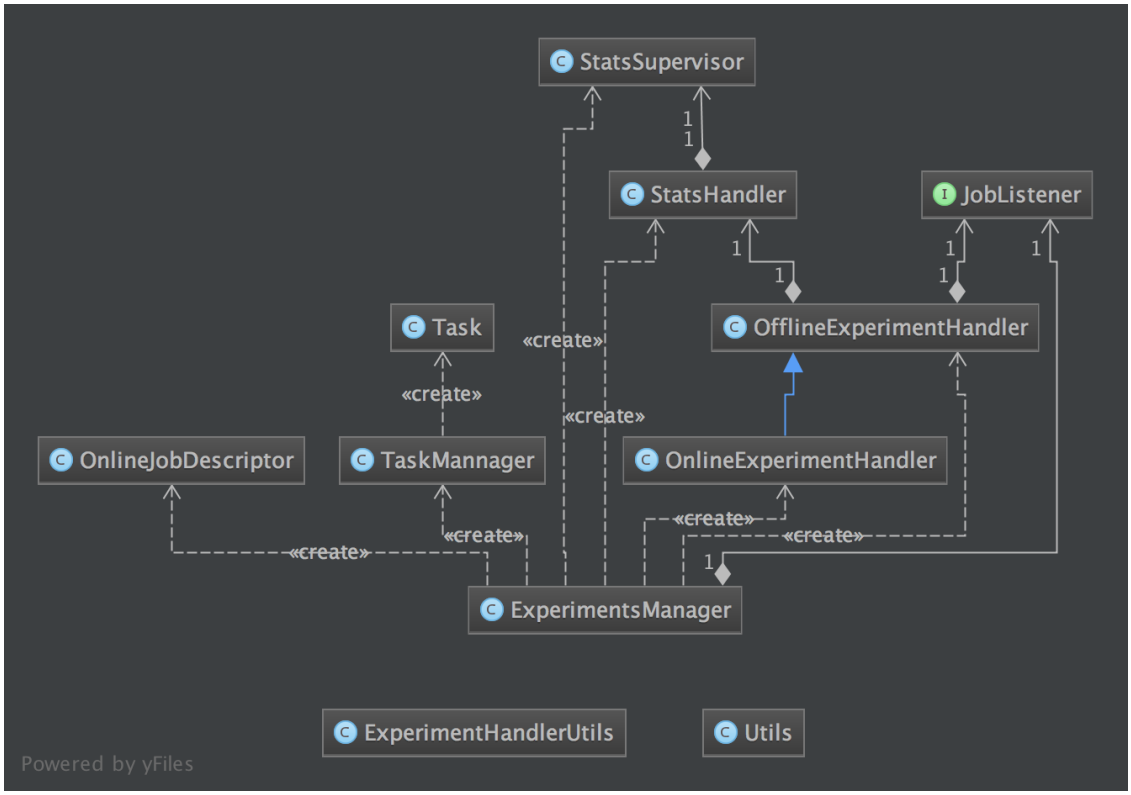
Najbitnija uloga modula *Engine* je pokretanje pokusa, a razredi koji se time bave nalaze se u paketima *console* i *task*. Paket *console* bavi se niskom razinom pokretanja pokusa tj. pokretanjem novog procesa uz odgovarajuće argumente komandne linije. Sučelje *Console* definira skup metoda potrebih za pokretanje novog pokusa. Apstraktni razred *AbstractConsole* implementira neke zajedničke metode dok razredi *CommandPrompt* i *Terminal* implementiraju specifičnosti za pojedini operacijski sustav. Razred *DetectOS* bavi se detekcijom konkretnog operacijskog sustava, a *ConsoleFactory* instancira odgovarajuću implementaciju u ovisnosti o operacijskom sustavu. *Job* je opisnik posla kojeg treba izvršiti, a sve važne događaje u životnom ciklusu pokusa (početak, kraj, neuspjeh) dojavljuje zainteresiranim objektima preko sučelja *JobObserver*.



Slika 2.11 Paket console

Paket *task* sadrži razrede zadužene za višu razinu upravljanja pokusima te se za svoje izvođenje oslanjaju na razrede iz paketa *console*. *ExperimentsManager* glavni je razred preko kojeg se pokreću novi pokusi. On se brine o svim slučajevima koji se mogu pojaviti pri pokretanju pokusa kao što su npr. paralelno pokretanje pokusa u više dretvi, stvaranje *online* odnosno *offline* pokusa, spajanje *stats* datoteka u jednu nakon razdvajanja. *OfflineExperimentHandler* i *OnlineExperimentHandler* bave se specifičnostima vezanim za *offline* odnosno *online* način izvođenja. *StatsSupervisor* i *StatsHandler* bave se spajanjem i razdvajanjem *stats* datoteka. *TaskMannager* bavi se stvaranjem bazena dretvi i paralelnim pokretanjem pokusa.

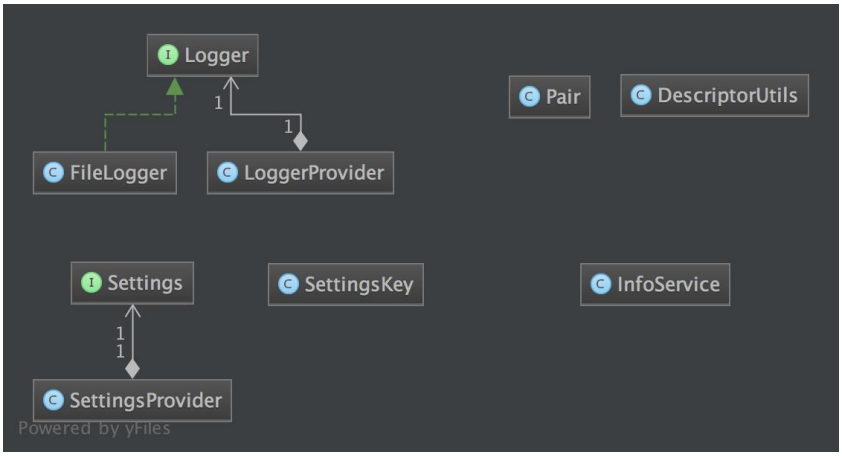
Ovaj modul nudi poprilično složene operacije tako da se pokazalo zgodnim pisati testove (*engl. Unit tests*). Testovi se nalaze u *test* direktoriju te povećavaju stabilnost aplikacije jer je pomoću njih moguće lako testirati rubne slučajeve. Prilikom svake promjene implementacije pokrenu se testovi čime se provjerava je li promjena uzrokovala kršenje na nekoliko uobičajenih slučajeva. To se posebno pokazalo korisnim za parser datoteka s rezultatima u koju su se naknadno dodavali podaci, a trebalo je osigurati da radi za sve prethodne slučajeve.



Slika 2.12 Paket task

2.2.3 Modul *Model*

Modul *Model* predstavlja pomoćni modul. U njemu se nalaze razredi za bilježenje grešaka (*Logger*, *FileLogger* i *LoggerProvider*), razredi za učitavanje postavki (*Settings*, *SettingsKey* i *SettingsProvider*) te pomoćni razredi (*Pair*, *DescriptorUtils* i *InfoService*).



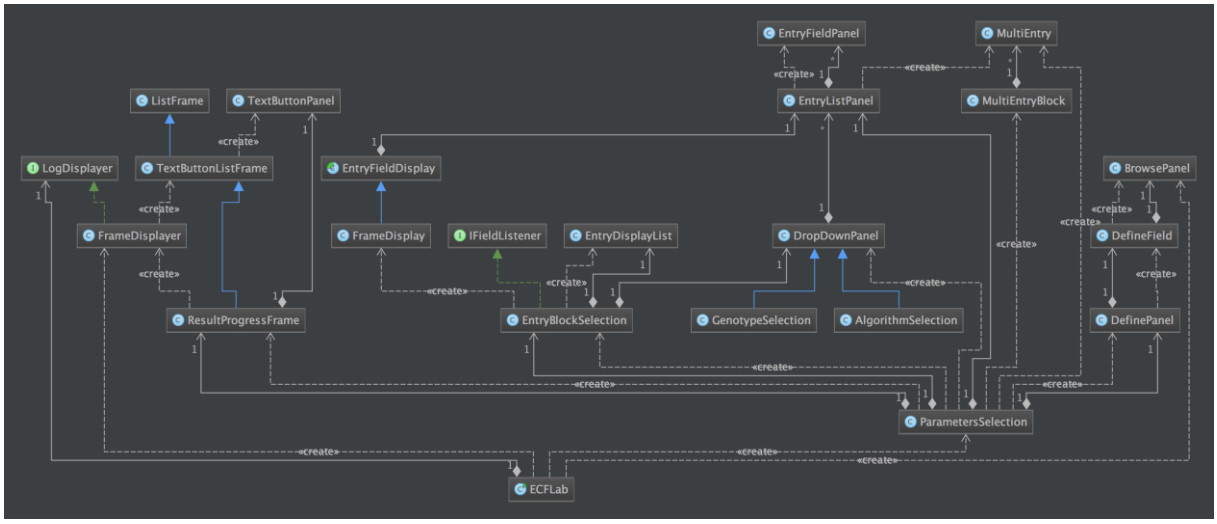
Slika 2.13 Modul Model

2.2.4 Modul *View*

Modul *View* brine se o grafičkom sučelju. Razred *ECFLab* glavni je prozor i ulazna točka aplikacije. U njemu se postavlja padajući izbornik i alatna traka sa svim pripadajućim akcijama. Stvaranjem nove konfiguracije ili otvaranjem postojeće otvorit će se nova kartica koja je predstavljena razredom *ParametersSelection*. Svaka kartica sastoji se od 3 rubrike za definiranje algoritama, genotipa i dodatnih parametara te rubrike za definiranje putanje konfiguracijske datoteke, broja korištenih dretvi i gumba za pokretanje pokusa. Ta posljednja rubrika predstavljena je razredom *DefinePanel* koji sadrži dva *DefineField*-a i gumb za pokretanje. *DefineField* sastoji se od teksta i polja za unos vrijednosti, a opcionalno se može dodati i gumb za pretraživanje diska s ciljem olakšavanja određivanja putanje datoteke.

EntryFieldPanel predstavlja jedan parametar tj. sadrži kućicu za označavanje uključenosti parametra, naziv parametra te polje odnosno polja za unos vrijednosti. Moguće je definirati više vrijednosti za jedan parametar. Prelaskom preko naziva parametra prikazat će se opis tog parametra. Unošenjem vrijednosti u polje kućica će se automatski označiti. *EntryListPanel* sadrži listu *EntryFieldPanel*-ova te im naizmjenično boja pozadinu u bijelu odnosno sivu boju. *DropDownPanel* omotava *EntryListPanel* te dodaje padajući izbornik pomoću kojeg je moguće odabrati odgovarajući algoritam odnosno genotip. *EntryBlockSelection* sadrži već dodane algoritme/genotipe, jedan *DropDownPanel* za definiranje novog algoritma/genotipa i gumb za njihovo dodavanje. *EntryDisplayList* predstavlja algoritme/genotipe koji ulaze u konfiguracijsku datoteku, a sastoji se od niza objekata tipa *EntryFieldDisplay*. Svaki taj objekt sadrži gumb s imenom algoritma/genotipa i gumb za brisanje odnosno isključivanje iz konfiguracijske datoteke. To je apstraktni razred jer nije definirana akcija za klik na gumb s imenom. Razred *FrameDisplay* nasljeđuje taj razred i implementira tu akciju otvaranjem novog prozora u kojem će se prikazati parametri.

Za svaku karticu postoji jedan prozor s rezultatima koji je predstavljen razredom *ResultProgressFrame*. Taj je razred izveden iz razreda *TextButtonListFrame* koji sadrži niz unosa s imenom i gumbom. Taj se razred koristi za izravno prikazivanje više paralelnih pokusa od kojih svaki ima jedno pokretanje, ali i za prikazivanje jednog pokusa koji ima više pokretanja. On je naslijeđen iz razreda *ListFrame* koji predstavlja prozor s nizom komponenti. Razred *TextButtonPanel* predstavlja generički razred koji se sastoji od teksta i gumba. Koristi se u *TextButtonListFrame*-u, a konkretna akcija zadaje mu se u razredu *FrameDisplayer*. Taj razred implementira sučelje *LogDisplayer* što znači da može prikazati rezultate pokusa. Instanca tog razreda primit će *LogModel* i zatim to pretvoriti u podatke potrebne *ChartKit*-u za vizualizaciju rezultata.



Slika 2.14 Modul View

2.3 Instalacija aplikacije ECF Lab

Za pokretanje aplikacije potrebno je instalirati Javin virtualni stroj koji se može skinuti sa službene Javine stranice [5]. Aplikacija će biti isporučena u obliku *jar* arhive koja se može pokrenuti dvoklikom. Izvorni kod aplikacije javno je dostupan na GitHub repozitoriju ECF Lab projekta [6]. Aplikaciju je moguće izgraditi iz izvornog koda korištenjem *gradle* alata. U korijenskom direktoriju projekta potrebno je pokrenuti odgovarajuću skriptu s opcijom *build*. Npr. u *unix terminalu* potrebno je pokrenuti naredbu *./gradlew build*, a na *Windowsima gradlew.bat build*. U direktoriju *View/build/libs* nalazi se jar arhiva ECFLab-1.2.jar koju je moguće pokrenuti dvoklikom.

3. Simbolička regresija

3.1 Uvod

Simbolička regresija postupak je pronalaženja matematičkog izraza iz empirijskih podataka. U tablici 3.1 nalazi se primjer empirijskih podataka koji su dobiveni mjerenjima pobude nekog sustava, uzorkovanjem neke funkcije i sl. Zadatak je iz dobivenog skupa za učenje naći matematički izraz koji ga dovoljno dobro opisuje.

x	-4	-3	-2	-1	0	1	2	3	4
t	0.844	-0.856	0.725	-1.695	-3.143	-1.457	0.255	-0.162	-0.059

Tablica 3.1 Primjer empirijskih podataka

Osim traženja parametara modela traži se i sam model koji nije pretpostavljen kao kod npr. linearne regresije. Prostor pretraživanja je prevelik pa se koristi genetsko programiranje za izgradnju stabla. Stablo predstavlja matematički izraz, a sastoji se od funkcija i terminala. Funkcije su npr. zbrajanje, dijeljenje, sinus, logaritam itd., a terminali mogu biti konstante ili varijable. Mogu se zadati konkretne vrijednosti konstanti ili interval iz kojeg će sustav slučajno odabrati jednu vrijednost. Stablo se evaluira za svaki uzorak iz skupa uzoraka te se računa ukupna funkcija pogreške. Postoje razne funkcije pogreške, a najčešće se koriste srednja kvadratna pogreška (*engl. Mean squared error, MSE*),

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t - f(x))^2$$

srednja apsolutna pogreška (*engl. Mean absolute error, MAE*)

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|t - f(x)|$$

i srednja apsolutna postotna pogreška (*engl. Mean absolute percentage error, MAPE*)

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{t - f(x)}{t}\right|$$

gdje je t izmjerena vrijednost iz skupa za učenje, a $f(x)$ vrijednost dobivena evaluacijom modela. *SRM* projekt podržava sve tri navedene funkcije pogreške te se one uključuju preko konfiguracijske datoteke.

3.2 Linearno skaliranje

Promotrimo slučaj kada imamo dvije funkcije cilja: $t_1 = x^2$ i $t_2 = x^2 + 100$. Korištenjem standardne simboličke regresije, pokazat će se velika razlika u efikasnosti traženja rješenja. Prvi će izraz biti lako pronađen (često čak i početnoj populaciji), dok drugi izraz često uopće niti neće biti pronađen. Razlika je u tome što sustav potroši previše vremena na pogađanje točne konstante umjesto na pogađanje oblika rješenja. Kao odgovor na taj problem dolazi tehnika linearnog skaliranja.

Linearno skaliranje je tehnika koja se koristi za unapređivanje simboličke regresije, a omogućava efikasno određivanje koeficijenata skaliranja i pomaka određene funkcije. Ako imamo zadanu funkciju $y = gp(x)$ koja je predstavljena stablom dobivenim genetskim programiranjem za ulazne podatke x, linearno skaliranje pomoći će nam pronaći parametre a i b tako da najbolje odgovaraju podacima za učenje ($t = a + by$). Ti se parametri izračunavaju linearnom regresijom uz pomoć sljedećih formula:

$$b = \frac{\sum[(t - \bar{t})(y - \bar{y})]}{\sum[(y - \bar{y})^2]}$$
$$a = \bar{t} - b\bar{y}$$

gdje t predstavlja točan izlaz funkcije određenog primjerka iz skupa za učenje, y vrijednost koju je model izračunao za dane ulazne podatke, a $\bar{t}$ i $\bar{y}$ aritmetičke sredine točnih odnosno izračunatih podataka. Suma se izračunava za sve primjerke iz skupa za učenje. Nakon izračuna tih dvaju parametara može se izračunati mjera pogreške koristeći skaliranu formulu $a + by$, npr. srednja kvadratna pogreška (*engl. Mean squared error, MSE*):

$$MSE(t, a + by) = \frac{1}{N}\sum_{i}^{N}(a + by - t)^2$$

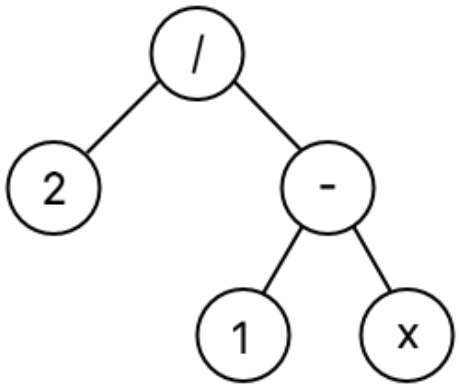
Ako je a različit od 0 i ako je b različit od 1, prethodno naveden postupak garantirano reducira srednju kvadratnu pogrešku za bilo koju formulu $y = gp(x)$.

Složenost izračunavanja tih dvaju parametara je linearna u odnosu na veličinu skupa za učenje što znači da je dodatni trošak (*engl. overhead*) zanemariv.

3.3 Intervalna aritmetika

Intervalna aritmetika generalna je metoda za izračunavanje granica aritmetičke operacije uz zadane granice ulaznih argumenata. Pomoću intervalne aritmetike možemo provjeriti je li dobiveni matematički izraz definiran za sve vrijednosti ulaznog prostora. Izraz neće biti valjan ako se npr. funkciju dijeljenja pokuša upotrijebiti s ulazom u granicama $[-2, 3]$. Intervalna aritmetika koristi se kao pretkorak u postupku prihvaćanja odnosno odbacivanja jedinke. Ako jedinka ne prođe provjeru granica ona se automatski odbacuje te do evaluacije niti ne dolazi. Računanje granica obavlja se rekurzivno, svaka funkcija će na temelju svojih podstabala znati odrediti zadovoljava li ulazni prostor granice domene. Budući da ne možemo znati koji raspon vrijednosti mogu poprimiti ulazne varijable, uzimaju se najmanja i najveća vrijednost za svaku varijablu iz skupa za učenje.

Pokažimo na primjeru kako to funkcionira. Na slici 3.1 nalazi se primjer stabla koje predstavlja matematički izraz $f(x) = \frac{2}{1-x}$. Ako je x definiran na intervalu $[-10, 10]$, desno podstablo odnosno operator oduzimanja izračunat će granice $[-9, 11]$. Operator dijeljenja prepoznat će da se radi o nedozvoljenoj situaciji jer u nazivniku može doći nula. Iz toga razloga ovo se stablo odbacuje odnosno njegova funkcija kazne postavlja se na veliku vrijednost. Da je x bio definiran na intervalu $[-10, 0]$, sve bi bilo u redu jer bi minus operator izračunao granice $[1, 11]$, a to bi bile valjane granice nazivnika.



Slika 3.1 Primjer stabla

3.4 Rezultati pokusa

3.4.1 Problemi

Testiranju će biti podvrgnuto 7 instanci problema, od toga će 6 skupova za učenje biti dobiveno uzorkovanjem različitih funkcija, a jedan skup za učenje javno je dostupan kao *Auto MPG* skup dostupan na *UCI* repozitoriju[10]. Uzorkovane funkcije navedene su u tablici 3.2.

oznaka	izraz	interval domene	funkcija kazne
symb1	$\log(x+1)+\log(x^2+1)$	$x \in [0, 20]$	*mse*
symb2	$\sin(x)+\sin(y^2)$	$x, y \in [-10, 10]$	*mse*
symb3	$2 \cdot \sin(x) \cdot \cos(y)$	$x, y \in [-10, 10]$	*mse*
symb4	$x \cdot y + \sin\big((x+1) \cdot (y-1)\big)$	$x, y \in [-10, 10]$	*mse*
symb5	$\frac{8}{2+x^2+y^2}$	$x, y \in [-10, 10]$	*mse*
symb6	$\frac{x^3}{5}+\frac{y^3}{2}-x-y$	$x, y \in [-10, 10]$	*mse*

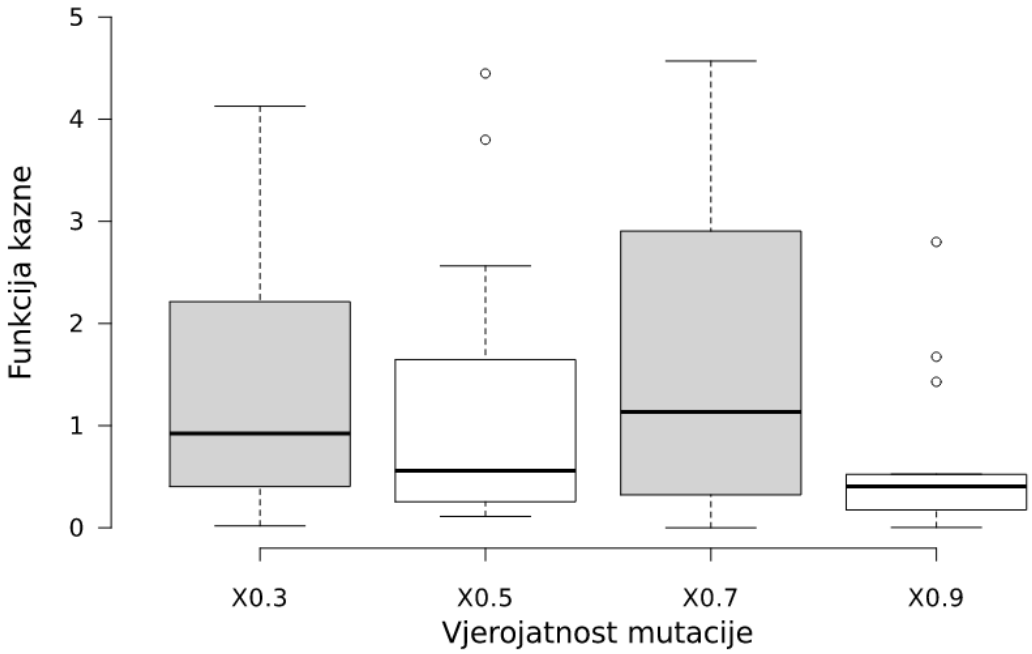
Tablica 3.2 Popis problema simboličke regresije

3.4.2 Zadani parametri

U pokusima simboličke regresije na ovim problemima koristi se troturnirska selekcija. Stablo može postići maksimalnu dubinu 7, a dostupne funkcije su zbrajanje, oduzimanje, množenje, dijeljenje, sinus, kosinus, drugi korijen i logaritam po bazi 10. Od terminalnih vrijednosti pojavljuju se varijable i slučajne konstante u intervalu $[-1, 1]$. Veličina populacije je 500 jedinki, a maksimalni broj evaluacija je 500,000. Zadana vrijednost vjerojatnosti mutacije je 0.3, a taj će se broj ažurirati nakon obavljenih pokusa za određivanje najbolje vrijednosti.

3.4.3 Određivanje najbolje vrijednosti parametra vjerojatnosti mutacije

Nad skupom za učenje s oznakom symb6 provedeno je po 15 pokusa s različitim vrijednostima vjerojatnosti mutacije: 0.3, 0.5, 0.7 i 0.9. Na slici 3.2 prikazani su rezultati pokusa. Iz navedenog se vidi da je najbolja vrijednost vjerojatnosti mutacije upravo vrijednost 0.9 te će se ta vrijednost koristiti u sljedećim pokusima.



Slika 3.2 Funkcija kazne za različite vrijednosti vjerojatnosti mutacije

3.4.4 Utjecaj intervalne aritmetike

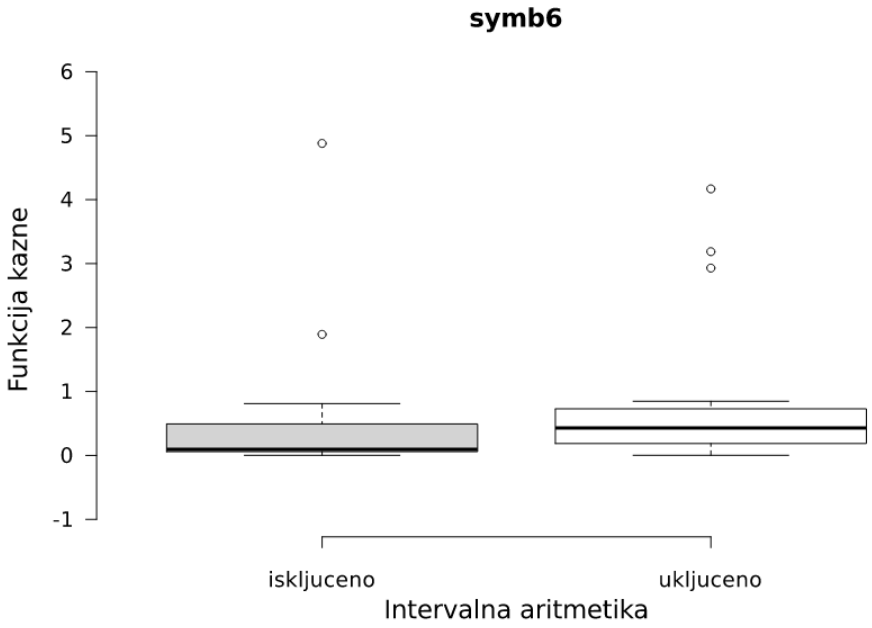
Nad svih 7 skupova za učenje provedeno je po 15 pokusa sa i bez uključene intervalne aritmetike. Za prvih 5 skupova, oba postupka postižu funkciju kazne gotovo ravnu nuli. Rezultati za skup symb6 i *Auto MPG* prikazani su na slikama 3.3 i 3.4. Najbolje dobiveno rješenje zapisano u prefiksnom obliku za skup symb6 izgleda ovako:

- * x1 + sqrt * + * x1 D_0.848782 * x1 D_0.848782 - - x1 D_-0.309998 log x2 - - cos * x1 D_0.527009 cos / x1 D_1.14542 * cos * x1 D_0.256514 + D_4.39236 / x1 x1 * * * x2 / x2 D_0.525036 D_-0.257057 x2,

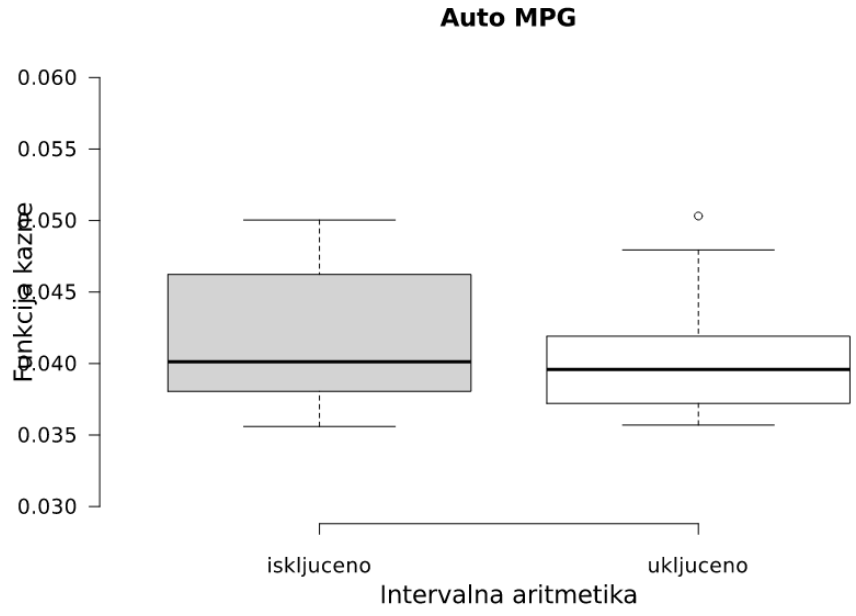
a za skup *Auto MPG* ovako:

/ - + + x5 sin * sqrt x5 x1 - log - - x2 D_-0.505867 + x3 x5 sin * log x5 sqrt x2 cos sqrt sqrt / + x5 x5 / x1 x4 cos sqrt cos sqrt sqrt * x2 x5

Konstane su prefiksirane s “D_”, a varijable počinju s “x” gdje pridruženi broj označava redni broj u skupu za učenje.



Slika 3.3 Rezultati za skup symb6



Slika 3.4 Rezultati za skup Auto MPG

3.4.5 Utjecaj linearnog skaliranja

Nad modificiranim skupom symb5 provedeno je ispitivanje utjecaja linearnog skaliranja. Skup je modificiran na način da je funkciji koja predstavlja taj skup dodana konstanta 100. Modificirana funkcija prikazana je sljedećom jednadžbom.

$$f(x,y) = \frac{8}{2 + x^2 + y^2} + 100$$

Izvedeno je po 15 pokusa s isključenim i uključenim linearnim skaliranjem te su oba postupka svela funkciju kazne gotovo na nulu.

3.5 SymReg Lab

SymReg Lab je aplikacija koja nam pruža grafičko sučelje za lakše pokretanje pokusa simboličke regresije. Kao i *ECF Lab*, napisana je u programskom jeziku Java korištenjem alata Swing tako da je neovisna o platformi i operacijskom sustavu. Također je korištena mogućnost *Look and Feel* kako bi aplikacija poprimila izgled autohtone aplikacije.

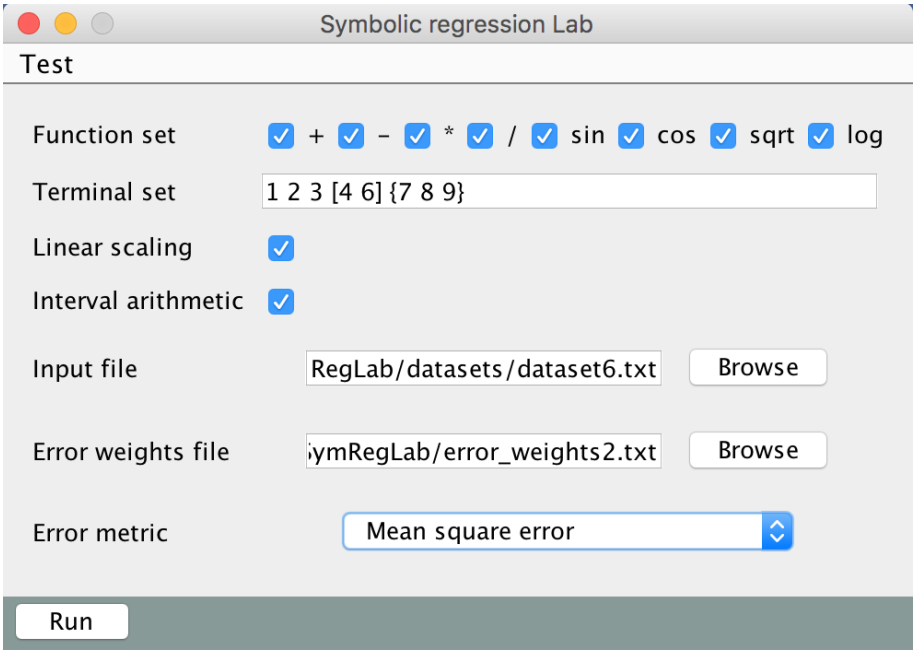
Aplikacija radi na način da sadrži već pripremljenu izvršnu ECF datoteku koja upogonjava simboličku regresiju. Koristi se *Engine* modul iz *ECF Lab* aplikacije za pokretanje pokusa. Postoje dvije ulazne točke aplikacije, paralelna (ParallelSymRegLab) i slijedna (SymRegLab), a razlikuju se jedino po načinu pokretanja pokusa. Paralelna će verzija pokretati pokuse u više dretvi te će ih iznova pokretati kako pokusi budu završavali, a slijedna će verzija pokrenuti samo jedan pokus i prestati s radom po završetku tog pokusa. Paralelna će verzija također izlučivati pareto frontu najboljih rješenja u odnosu na složenost modela te je periodično ispisivati na zaslon.

3.5.1 Korištenje aplikacije SymReg Lab

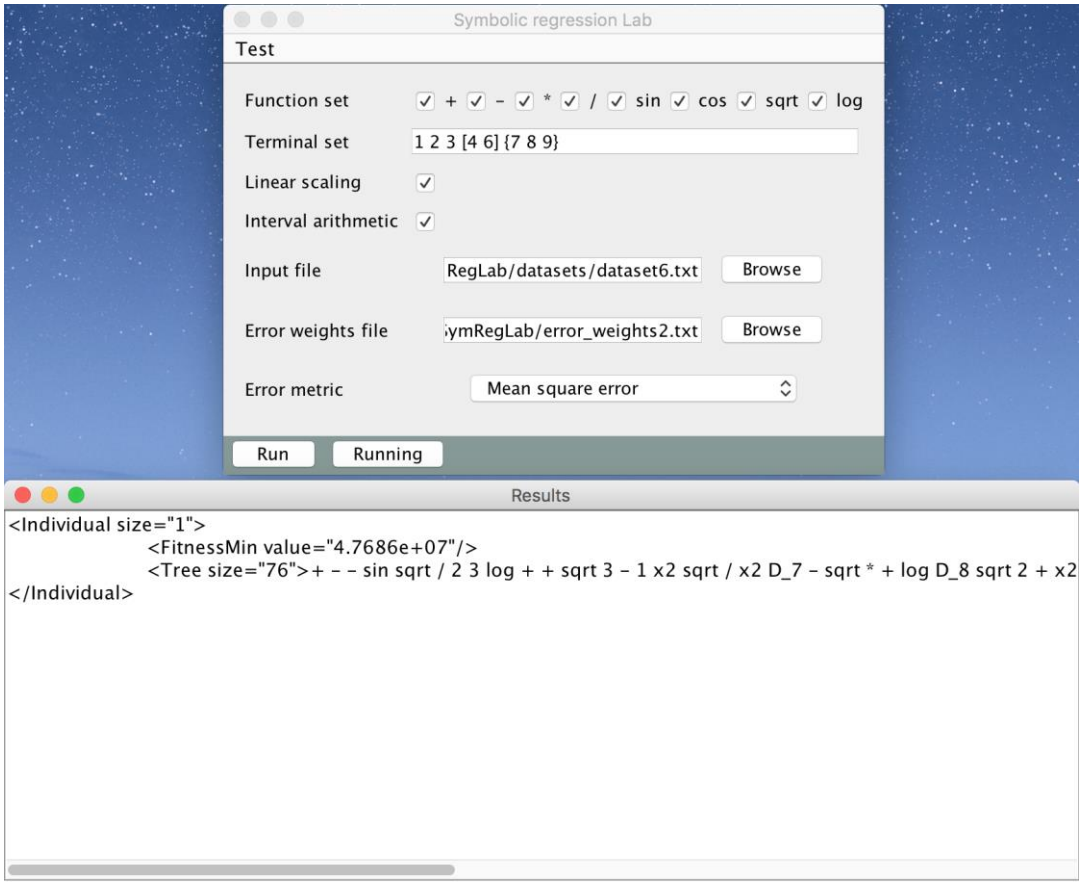
Aplikacija *SymReg Lab* sastoji se od jednog glavnog prozora na kojem se nalaze sve kontrole za pokretanje pokusa simboličke regresije. U glavnom prozoru, ispod izborničke trake nalazi se skup funkcija koje će biti uključene u stablo (*engl. Function set*). Nakon toga moguće je definirati i skup terminala (*engl. Terminal set*) što uključuje konstante i varijable. Varijable nije potrebno ručno definirati jer se automatski provjerava koliko je ulaznih varijabli u skupu podataka za učenje (*engl. Dataset*) te ih se dodaje u konfiguracijsku datoteku prilikom pokretanja. Ako želimo uključiti linearno skaliranje (*engl. Linear scaling*) ili intervalnu aritmetiku (*engl. Interval arithmetic*) potrebno je označiti kućice u sljedeća dva reda. Definiranje ulazne datoteke (*engl. Input file*) koja sadrži skup podataka za učenje te datoteke za težinske koeficijente (*engl. Error weights file*) obavlja se u sljedeća dva tekstualna polja uz pomoć pretraživača diska (*engl. Browse*). Ulazna datoteka treba biti u

tekstualnom formatu gdje svaki redak predstavlja jedan uzorak. U svakom retku trebaju se nalaziti ulazni podaci i jedan izlazni podatak odvojeni znakom tabulatora. *SRM* projekt trenutno podržava samo funkcije s jednim izlazom. Datoteka s težinskim koeficijentima sastoji se od onoliko redaka koliko ima uzoraka u skupu za učenje. U svakom je retku jedna vrijednost koja predstavlja koeficijent s kojim se množi funkcija greške pridružena tom uzorku. Moguće je definirati i funkciju greške (*engl. Error metric*) uz pomoć padajućeg izbornika.

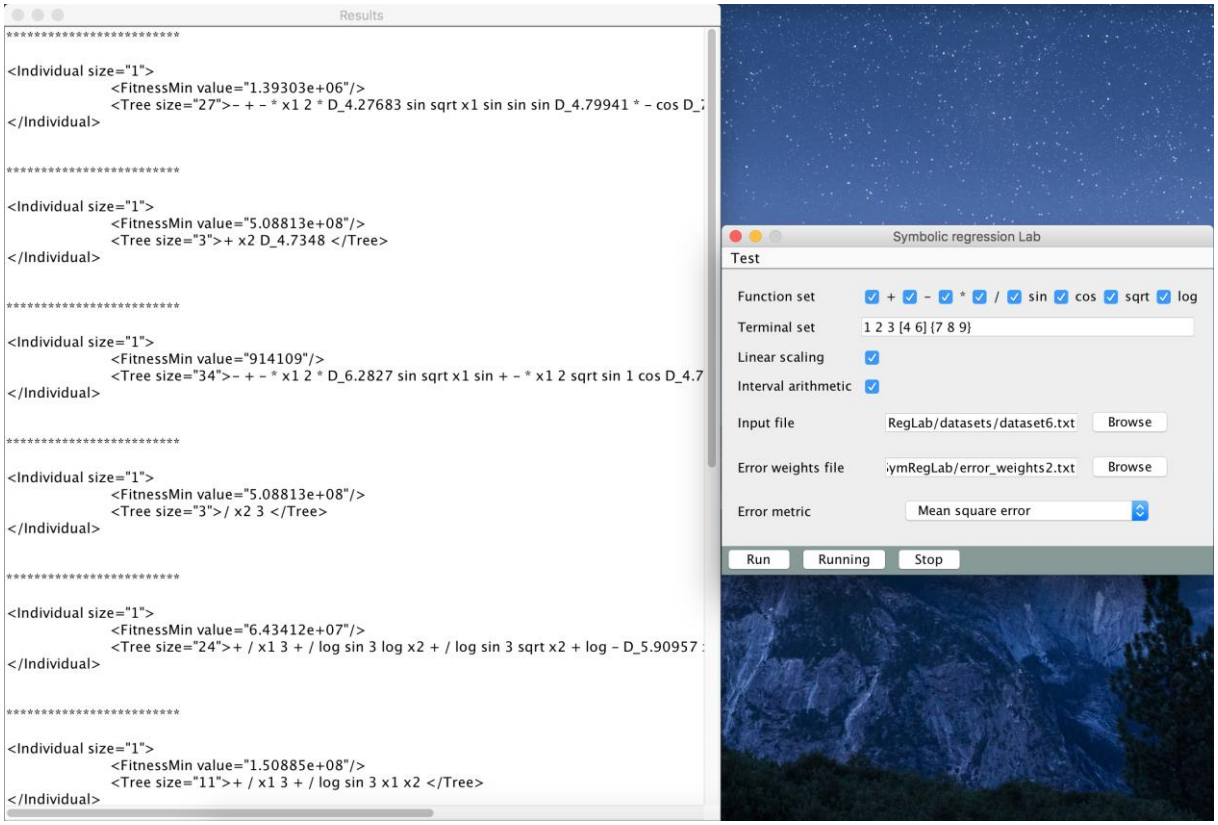
Na dnu je odvojena traka za pokretanje pokusa i otvaranje rezultata. U slijednoj verziji, klikom na gumb *Run* pokreće se jedan pokus te se rezultat tog izvođenja može vidjeti klikom na gumb *Finished* odnosno klikom na gumb *Running* ako je pokus još uvijek u tijeku. U paralelnoj verziji, pokretat će se niz pokusa koje je moguće zastaviti klikom na gumb *Stop*. Rezultati će se osvježavati svake sekunde te će se prikazivati pareto fronta najboljih rješenja u odnosu na složenost modela. Modeli se biraju po dva kriterija: grešci modela i veličini stabla. Često će nam biti zanimljivi modeli koji imaju manju složenost, a neznatno veću grešku na skupu za učenje.



Slika 3.5 SymReg Lab - početni zaslon



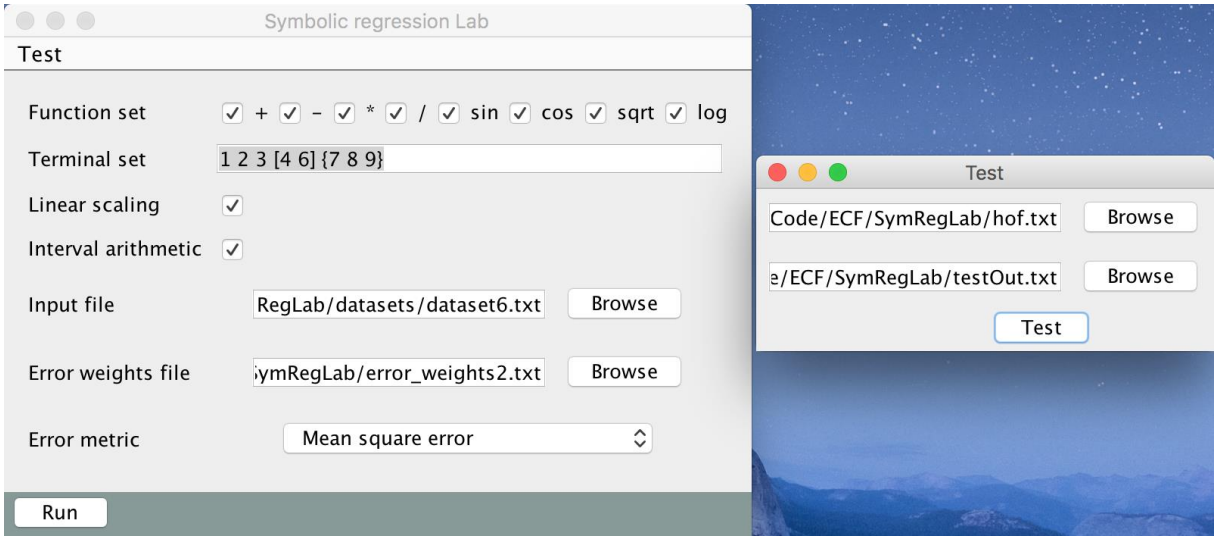
Slika 3.6 Slijedna verzija - rezultati izvođenja



Slika 3.7 Paralelna verzija - rezultati izvođenja

3.5.2 Ispitivanje dobivenih rješenja

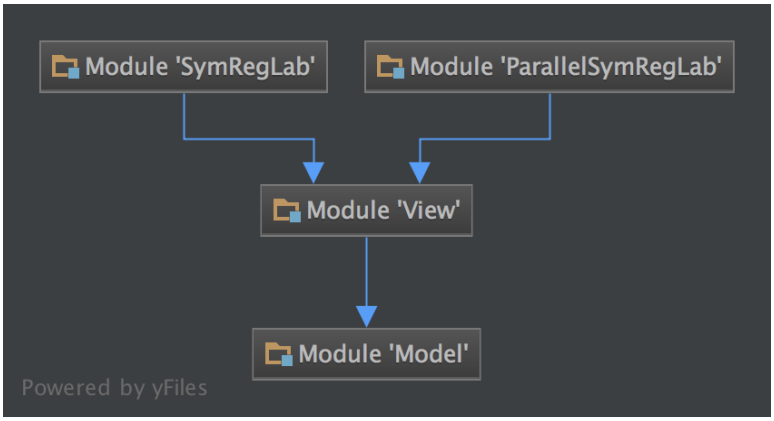
Na samom vrhu nalazi se izbornička traka s opcijom za testiranje jedinke. Klikom na tu opciju otvara se novi prozor u kojem je potrebno navesti putanju do XML datoteke u kojoj je spremljena jedinka koja se želi testirati. Također je potrebno navesti i putanju do datoteke u koju će biti spremljen rezultat testiranja. Klikom na gumb *Test*, iz glavnog prozora prikupit će se svi podaci o konfiguracijskoj datoteci i pokrenuti izvršna datoteka s opcijom za testiranje jedinke. Kao ulaznu datoteku u glavnom prozoru (*engl. Input file*) potrebno je navesti datoteku u kojoj se nalazi skup za ispitivanje, a koja je istog formata kao i datoteka sa skupom za učenje. Aplikacija će i izlaznu datoteku oblikovati u tom formatu gdje će se u zadnjem stupcu nalaziti rezultati koje ta jedinka daje za odgovarajuće ulazne podatke.



Slika 3.8 Testiranje jedinke

3.5.3 Izvedba aplikacije SymReg Lab

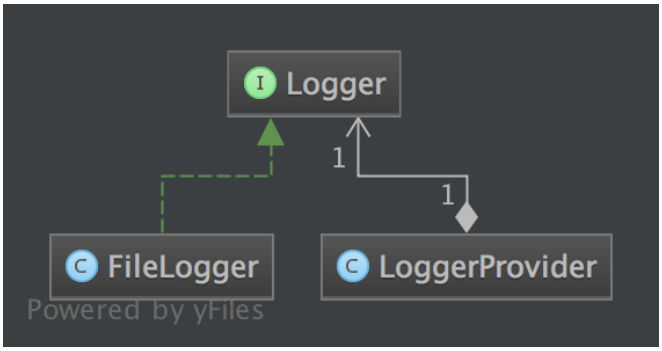
Projekt SymReg Lab sastoji se od 4 modula: *Model*, *View*, *SymRegLab* i *ParallelSymRegLab*. Posljednja su dva modula ustvari ulazne točke aplikacije, a oslanjaju se na zajednički modul *View* koji se oslanja na modul *Model*. *Model* se brine za pokretanje pokusa i za to koristi modul *Engine* iz aplikacije ECF Lab koji je dostupan kao jar arhiva. Uz to sadrži i pomoćne razrede i metode. *View* sadrži razrede i logiku vezane za grafičko sučelje. Modul *SymRegLab* ulazna je točka za slijednu verziju aplikacije, a modul *ParallelSymRegLab* za paralelnu verziju.



Slika 3.9 Struktura aplikacije SymReg Lab

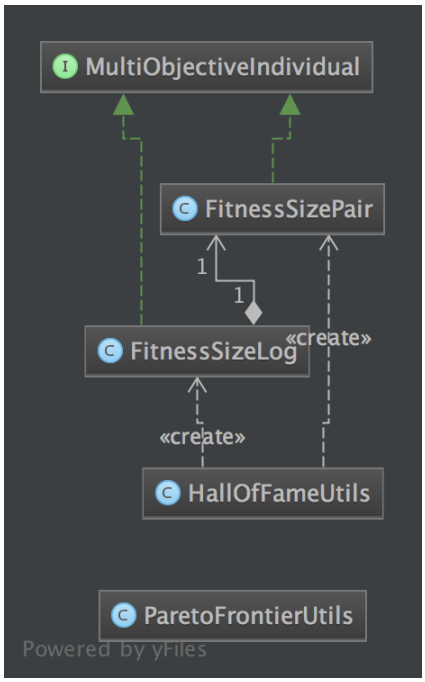
Modul *Model* podijeljen je u 4 paketa. Paket *logger* služi za zapisivanje pogrešaka do kojih dođe prilikom rada sustava. Sučelje *Logger* definira metode za zapisivanje pogrešaka. *FileLogger* konkretna je implementacija koja pogreške

zapisuje u datoteku. *LoggerProvider* je *singleton* koji odlučuje koja će se konkretno implementacija koristiti. U slučaju ove aplikacije to je i jedina dostupna implementacija, *FileLogger*.



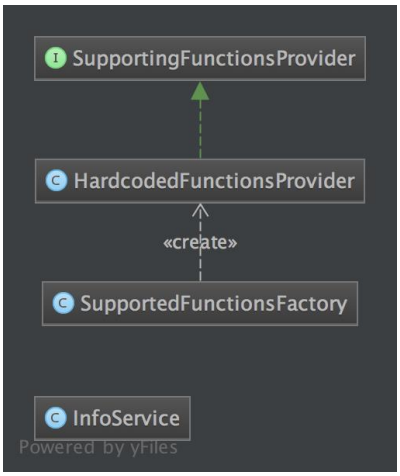
Slika 3.10 Paket logger

Paket *util* sadrži pomoćne razrede i metode. Razred *ParetoFrontierUtils* sadrži pomoćne metode za pronalaženje pareto fronte iz niza rezultata, a razred *HallOfFameUtils* pomoćne metode za izvlačenje raznih informacija iz opisnika rezultata odnosno iz opisa najboljeg rješenja. Sučelje *MultiObjectiveIndividual* predstavlja višekriterijsko rješenje, a u ovoj se aplikaciji koristi za izlučivanje pareto fronte jer imamo dva kriterija: dobrotu rješenja i veličinu stabla. Razredi *FitnessSizePair* i *FitnessSizeLog* služe kao pomoćni razredi pri izlučivanju pareto fronte.



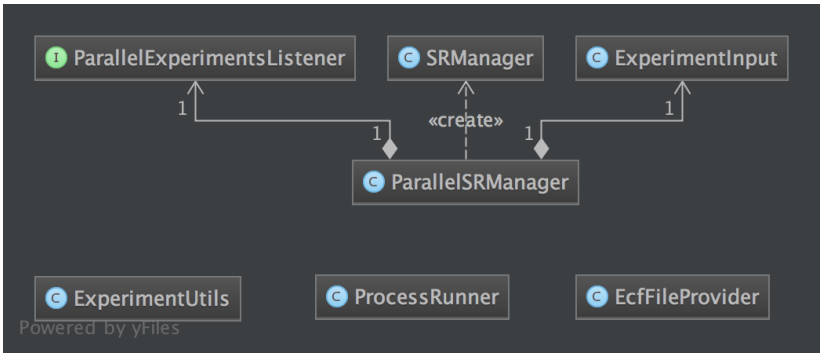
Slika 3.11 Paket util

Paket *info* sadrži razrede čiji objekti pružaju različite informacije. Sučelje *SupportingFunctionsProvider* definira metodu za objekte koji pružaju informaciju o dostupnim funkcijama. To je trenutno čvrsto definirano objektom tipa *HardcodedFunctionsProvider*, a dostupan je preko razreda tvornice *SupportedFunctionsFactory* koja ga instancira. Razred *InfoService* je *singleton* koji pamti zadnje odabranu putanju što nam olakšava odabiranje datoteka koje su spremljene blizu na disku.



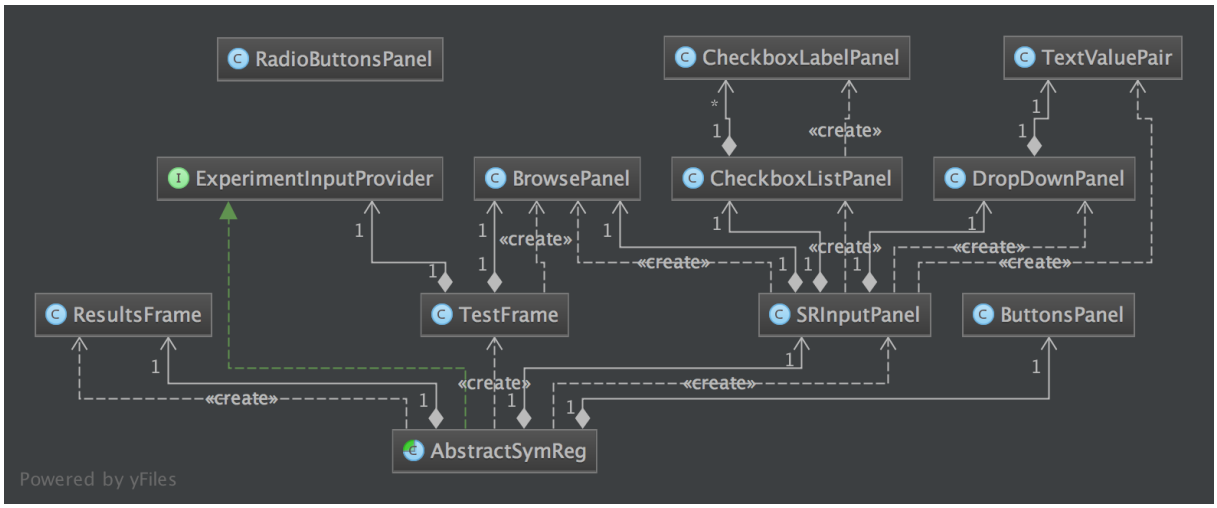
Slika 3.12 Paket info

Paket *exp* sadrži razrede koji se brinu o izvođenju pokusa. *SRManager* brine se o izvođenju jednog pokusa dok se *ParallelSRManager* brine o višestrukom paralelnom pokretanju pokusa koristeći objekte tipa *SRManager*. *ExperimentInput* je razred koji sadrži sve potrebne informacije za pokretanje pokusa simboličke regresije kao što su putanja do datoteke s podacima za učenje ili skup terminala. Sučelje *ParallelExperimentsListener* definira metode za objekt slušača na promjene u pareto fronti trenutnih rješenja. *ExperimentUtils* sadrži pomoćne metode, uglavnom za modifikacije konfiguracije pokusa. Razred *ProcessRunner* olakšava pokretanje jednostavih operacija nad izvršnom datotekom, a konkretno se koristi za pokretanje operacije testiranja jedinke. Razred *EcfFileProvider* sadrži informacije o putanji izvršne datoteke unutar projekta koja se koristi za izvršavanje pokusa.



Slika 3.13 Paket exp

Modul *View* zadužen je za grafičko sučelje te se u njemu nalaze svi razredi zaduženi za iscrtavanje na zaslon. *AbstractSymReg* apstraktni je razred koji definira glavni prozor aplikacije. Sastoji se od jednog *SRInputPanel*-a koji je zadužen za konfiguriranje pokusa i od jednog *ButtonsPanel*-a koji sadrži gumbe za upravljanje pokusima. Ta je klasa apstraktna jer paralelna i slijedna verzija imaju različitu logiku vezanu za traku s gumbima pa je svaka zasebno implementira. *DropDownPanel* predstavlja padajući izbornik, a koristi pomoćnu klasu *TextValuePair* jer je potrebno jednu vrijednost prikazati, a drugu iskoristiti prilikom izrađivanja konfiguracije. Npr. za funkciju pogreške potrebno je prikazati “Mean absolute error”, a u konfiguracijsku datoteku potrebno je upisati “mean_absolute_error”. *CheckboxListPanel* koristi se za odabir skupa funkcija, a sastoji se od niza *CheckboxLabelPanel*-a koji predstavljaju kućicu za označavanje i pripadajući tekst. *TestFrame* predstavlja prozor za testiranje jedinke, a preko sučelja *ExperimentInputProvider* dohvaća konfiguraciju iz glavnog prozora. *ResultsFrame* predstavlja prozor za prikaz rezultata pokusa.



Slika 3.14 Modul View

Moduli *SymReg* i *ParallelSymReg* sadrže samo po jednu klasu koje sadrže *main* metodu za pokretanje aplikacije te implemetaciju logike s gumbima za upravljanje pokusima.

3.5.4 Instalacija aplikacije SymReg Lab

Za pokretanje aplikacije potrebno je instalirati Javin virtualni stroj koji se može skinuti sa službene Javine stranice [5]. Aplikacija će biti isporučena u obliku dvije *jar* arhive, jedna za slijednu, a druga za paralelnu verziju koje se mogu pokrenuti dvoklikom. Izvorni kod aplikacije javno je dostupan na GitHub repozitoriju SymReg Lab projekta [7]. Aplikaciju je moguće izgraditi iz izvornog koda korištenjem *gradle* alata. U korijenskom direktoriju projekta potrebno je pokrenuti odgovarajuću skriptu s opcijom *build*. Npr. u *unix terminalu* potrebno je pokrenuti naredbu *./gradlew build*, a na *Windowsima gradlew.bat build*. Ovisno o željenoj verziji aplikacije, odgovarajuću jar arhivu potrebno je potražiti u direktoriju *SymRegLab/build/libs* odnosno u direktoriju *ParallelSymRegLab/build/libs*. U tim direktorijima nalaze se jar arhive SymReg_Lab-2.0.jar odnosno Parallel_SymReg_Lab-2.0.jar koje je moguće pokrenuti dvoklikom.

4. Zaključak

Metaheuristike nam pomažu kako bismo za teško izračunljive probleme dobili zadovoljavajuće dobra rješenja. Brojni problemi u praksi teško su rješivi pa se uspješno rješavaju primjenom odgovarajućih metaheuristika. Kako mnoge implementacije metaheuristika imaju dosta toga zajedničkog, pokazalo se zgodno izdvojiti neke funkcionalnosti u biblioteke koje će nam olakšati daljnju implementaciju novih problema. Jedna od takvih biblioteka je ECF.

ECF je dosad bio pokretan isključivo iz konzole uz ručno pisanje konfiguracijske datoteke, a rezultati dobiveni radom ECF-a ostajali su u tekstualnom obliku te nisu bili vizualizirani. *ECF Lab* aplikacija je koja rješava te probleme, tj. omogućava jednostavno pokretanje pokusa i vizualizaciju rezultata kroz grafičko korisničko sučelje.

Simbolička regresija postupak je pronalaženja matematičkog izraza iz empirijskih podataka. Efikasnost simboličke regresije moguće je znatno poboljšati tehnikama linearnog skaliranja i intervalne aritmetike uz neznatan dodatni trošak. Aplikacija *SymReg Lab* izrađena je s ciljem jednostavnijeg pokretanja pokusa simboličke regresije.

5. Literatura

[1] Čupić, M. Prirodom inspirirani optimizacijski algoritmi. Metaheuristike., 2013.

[2] Kiš Miroslav, Englesko-hrvatski i hrvatsko-engleski informatički rječnik, Zagreb, Naklada Ljevak, 2000., str. 36

[3] Službena stranica projekta ECF http://gp.zemris.fer.hr/ecf/

[4] Službena stranica projekta JFreeChart http://www.jfree.org/jfreechart/

[5] Službena javina stranica https://www.java.com/en/download/

[6] Izvorni kod aplikacije ECF Lab na *GitHub* repozitoriju https://github.com/dstank25/ECF_Lab

[7] Izvorni kod aplikacije SymReg Lab na GitHub repozitoriju https://github.com/dstank25/SymReg_Lab

[8] Maarten Keijzer, *Improving Symbolic Regression with Interval Arithmetic and Linear Scaling*, Computer Science Department, Free University Amsterdam

[9] Simbolička regresija, https://en.wikipedia.org/wiki/Symbolic_regression

[10] *Auto MPG* skup podataka za učenje, https://archive.ics.uci.edu/ml/datasets/Auto+MPG

6. Sažetak

Ovaj rad sastoji se od 3 dijela: dijela za olakšavanje pokretanja generičkih pokusa koji koriste ECF radni okvir, dijela za unapređivanje postupka simboličke regresije i dijela koji olakšava pokretanje pokusa simboličke regresije. Aplikacija *ECF Lab* predstavlja grafičko korisničko sučelje za jednostavnije upravljanje ECF-om, a aplikacija *SymReg Lab* služi za jednostavnije pokretanje pokusa simboličke regresije.

Ključne riječi: ECF, *Evolutionary Computation Framework*, grafičko korisničko sučelje, *ECF Lab*, evolucijsko računanje, metaheuristika, simbolička regresija, linearno skaliranje, intervalna aritmetika, *SymReg Lab*.

7. Summary

This thesis consists of 3 parts:

- application for easier handling of generic experiments that use ECF framework
- applying techniques for enhanced symbolic regression
- application that simplifies running of symbolic regression experiments

ECF Lab is a desktop application that provides graphical user interface for simple management of *ECF* experiments. *SymReg Lab* is a desktop application that makes running symbolic regression experiments easy.

Keywords: *ECF, Evolutionary Computation Framework*, graphical user interface, *ECF Lab*, evolutionary computation, metaheuristics, symbolic regression, linear scaling, interval arithmetic, *SymReg Lab*.