Optimizacija organizacije rasporeda nastave (Schedule)

Romana Erdelji, SW58/2017

Motivacija:

Rešavanje Nurse Scheduling Problem-a (NSP) adaptiran na problem organizacije rasporeda nastave u odnosu na zahteve fakulteta i želje (koji dani bi im više odgovarali u odnosu na druge) profesora i asistenata (hard and soft constraints).

Pregled problema:

Za rešavanje problema bi se koristio genetski algoritam. Algoritam bi bio implementiran u programskom jeziku Go i u programskom jeziku Rust, kako bi se poredile performanse i težina implementacije.

Takođe bi postojale crud operacije koje bi omogućavale unošenje potrebnih entiteta za adekvatan rad algoritma.

Scheduling problem:

*Genetski algoritam*

Predstavlja optimizacioni algoritam baziran na Darvinovoj teroriji evolucije. Upotrebom evolucionih mehanizama (selekcije, ukrštanje, mutacija) populacije se poboljšava iz generacije u generaciju i ponalazi se optimalno rešenje

*Pojmovi*

* **Jedinka (hromozom)** – potencijalno rešenje problema, implementirana kao struktura podataka/niza promenljivih
* **Gen** – deo jedinke koji se odnosi na jedan brojčani podatak, promenljiva
* **Populacija** – skup svih jedinki u jednom trenutku, iteracija
* **Fitness (Prilagođenost) –** vrednost kriterijuma optimalnosti
* **Elite potomci –** jedinke sa najboljim fitness-om
* **Soft constraints –** predstavljaju bonus prilikom računanja fitness-a
* **Hard constraints –** zahtevi koji moraju biti ispunjeni da bi se smatralo da je rezultat validan
* **End criteria (kriterijum za završetak) –** zadovoljavanje hard constraints

*Opis rada algoritma*

Simulacija algoritma počinje time što se generiše inicijalna populacija i izračuna fitness svake jedinke. U toku jedne iteracije se biraju roditelja iz predhodne generacije, rekombinuju se u dva potomka, izračuna njihov fitness i ubace u novu populaciju. Pre pokretanja nove iteracije takođe se izaberu elite potomci. Prestanak rada algoritma može da usledi nakon što je populacija iskonvergirala odnosno dostigao se generation limit ili smo zadovoljili kriterijum za završetak.

*Opis implementac ije*

Radi što boljeg poređenja implementacije i performansi korišteni su isti podaci i strukture podataka u Go i Rust-u.

Entiteti

Admin upravlja celom aplikacijom (sve crud operacije i pokretranje generisanja rasporeda – rad algoritma), dok profesor/asistent (dalje zaposleni) može samo da vidi svoj raspored ukoliko postoji, predavanja i mogu da menjaju svoje želje. Želje predstavljaju namapirane radne dane i vrednost od 1 do 5 koliko zaposlenom odgovara da drži predavanja tog dana. Fakultet ima referencu ka profesorima/asistentina, predmetima, smenama, zahtevu i rasporedu ukoliko je generisan. Zahtevi fakulteta predstavljaju ograničenja koliko jedan zaposleni može maksimalno/minimalno predavanja da ima u toku nedelje, koliko predavanja može biti održano u jednoj smeni i koliko maksimalno zaposleni može da ima predavnja u toku dana. Raspored u sebi sadrži za koja je smena dodeljena profesoru za održavanje zadatog predavanja. Ostali entiteti predstavljaju predavanja koji imaju svoj naziv i koliko puta se nedeljno drže i smene koje imaju od kad do kad je i kog dana.

Schedule

Nakon što su se svi potrebni podaci dobavili i preprocesirali (uzeti samo potrebni podaci iz klasa i stavljeni u određene strukture kako bi bila jednostavnija manipulacija podacima tokom rada algoritma) započinje simulacija algoritma kao što je opisana gore sa malom izmenom. Izmena se zasniva u tome što se pre određivanja fitness za populaciju namesti model rasporeda koji će se čuvati za svaku populaciju.

Za mutaciju je bila odabrana TwoPointCrossover, za selekciju između elite je odabran Roulette. Za veličinu populacije je izabrano 20 a maksimalan broj iteracija 15 hiljada jer nakon toga se ustanovilo da dolazi do stagniranja.

Pošto je rađeno sa listama imali smo siguran raspored elemenata i time je za hromozom izabrana lista veličine ukupnog broja predavanja koji se održavaju na fakultetu gde vrednost niza predstavlja broj veličine od 1 do 5 što se mapiralo na enum radnih dana u nedelji. Na osnovu toga se birao dan i smena koja je ostala slobodna za taj dan, ukoliko nije bilo slobodne smene raspored je otpao i dat mu je najmanji fitness.

*Go vs Rust*

U oba slučaja su se koristile postojeće biblioteke za genetički algoritam, koje su služile kao kostur, gde su se override-ovale glavne metode i birale mutacija, selekcija i propratni parametri.

Implementacija

Go je pružao više fleksibilnosti u razvoju dok je Rust zbog svog načina rada sa podacima i strukturama davao dosta ograničenja gde su morale da se nađu zaobilaznice, takođe Go nam pomoću svojih gorutine daje mogućnost paralelizacije rada algoritma i time ubrzanost dok smo u Rust-u imali mogućnost samo jedne niti. Takođe biblioteka koja je nađena za Go je bila više generična i davala mogućnosti za lakim proširivanjem klasa i odabira šta ulazi u scope algoritma. Za Rust su morale da se koriste globalni objekti koji su uvodili „unsafe“ ali nije bilo drugih opcija za čuvanje podataka tokom izvršavanja algoritma.

Rezultati

Kao i što je bilo očekivano, Rust je davao znatno bolje rezultate u odnosu na Go. Rezultati su sakupljeni tako što se generisanje rasporeda izvršavalo više puta uzastopno za iste podatke.

Rust je postizao rezultate za isto vreme kao što je Go sa 10 gorutina, ali to je zahvaljujući statičkoj tipiziranosti Rust-a.