SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Algoritam klonske selekcije i primjena u stvaranju rasporeda sati

Josip Mrđen

Voditelj: Domagoj Jakobović

Zagreb, travanj, 2017.

**Sadržaj**

[1. Uvod 1](#__RefHeading___Toc159987575)

[2. Seminarski rad 2](#__RefHeading___Toc159987576)

2.1 Općenito o evolucijskom računanju i NP problemima……………………….2

2.2 Prikaz rješenja i njegove modifikacije………………………………………….4

2.2.1. Prikaz rješenja………………………………………………………...4

2.2.2 Operacije i modifikacije nad rješenjima……………………………..5

2.3 Umjetni imunološki sustav………………………………………………………7

2.4 Algoritam klonske selekcije (CLONALG)……………………………………...8

2.4.1. Motivacija………………………………………………………………8

2.4.2. Prikaz algoritma i korišteni operatori………………………………..9

2.4.3. Parametri klonske selekcije………………………………………...11

3. Problem stvaranja školskog rasporeda……………………………………………..11

3.1. Motivacija i prikaz problema………………………………………………….12

3.2. Prikaz rješenja…………………………………………………………………13

3.3. Primjena algoritma…………………………………………………………….14

3.3.1. Inicijalizacija………………………………………………………….14

3.3.2. Selekcija………………………………………………………………15

3.3.3. Kloniranje i mutacije…………………………………………………15

3.3.4. Iteriranje programa………………………………………………….15

3.4. Rezultati………………………………………………………………………...17

[4. Zaključak 1](#__RefHeading___Toc159987577)9

[5. Sažetak 20](#__RefHeading___Toc159987578)

[6. Literatura 21](#__RefHeading___Toc159987579)

# Uvod

Revolucija računalne tehnologije je u zadnjih par desetljeća masovno obilježila tijek civilizacije, poglavito zbog rješavanja brojnih računskih problema koji su postojali prije toga. Upotrebom računala i drugih elektroničkih naprava koji mogu obavljati milijune instrukcija po sekundi, do raznih programa i algoritama koji predstavljaju rješenje nekog problema, napredak tehnologije počeo je rasti eksponencijalnom brzinom. Izmišljeni su brojni algoritmi koji daju rješenja u konstantnom vremenu, što se pokazalo savršeno za determinističke probleme.

No, na svijetu postoji enorman broj problema čiji opseg rješenja je takav da iscrpna pretraga nadmašuje ljudski vijek pa čak i više. U skupinu takvih problema spadaju i NP-kompletni problemi, a to su problemi odlučivanja za koje postoji nedeterministički Turingov stroj koji može pronaći rješenje u polinomijalnom vremenu. Oni se javljaju u svim područjima ljudskog života, od traženja ekstrema funkcije više varijabli, nalaženja optimalnog puta, do pronalaženja prave dijagnoze u dijagnostici i medicini, stvaranja rasporeda sati u obrazovnim ustanovama te raznim drugim granama u društvu, tehničkim i prirodnim znanostima. U potrazi za rješenjem takvih problema, čovjek je upotrijebio primjer evolucije i prirodne selekcije koji danas nosi naziv evolucijsko računanje. Evolucijsko računanje i evolucijski algoritmi su postali vrlo popularni zadnjih godina te se gotovo uvijek koriste na kombinatornim problemima s velikim skupom rješenja gdje se do nekog zadovoljavajućeg rješenja treba doći u razumnom vremenu.

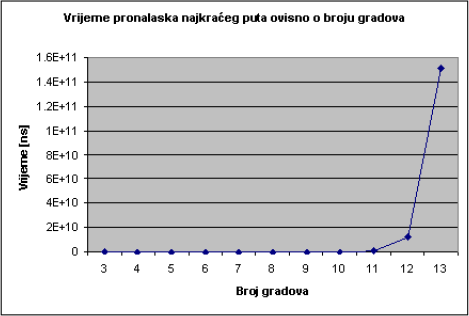
Tema ovog seminara bit će predstavljanje problema stvaranja rasporeda sati u školama i visokim obrazovnim ustanovama. U prvom dijelu seminara uvest će se pojam evolucijskog računanja i evolucijskih algoritama te će se navesti stručni termini pomoću kojih se implementira rješenje problema. Drugi dio će suziti izdašan izbor algoritama na one koje imitiraju imunološki sustav čovjeka, tzv. umjetni imunološki sustav, a unutar te grane bit će analiziran algoritam klonske selekcije. U trećem poglavlju uvest će se problem stvaranja rasporeda sati, navest će se nekoliko različitih načina rješenja te problemi i ograde koje se moraju ukloniti da bi se došlo do validnog rješenja. Zadnji dio seminara predstavlja konkretno rješenje problema pomoću algoritma klonske selekcije te se prikazuju rezultati i uspoređuju s ostalim evolucijskim algoritmima.

# Pregled evolucijskog računanja i analiza umjetnog imunološkog sustava

**2.1 Općenito o evolucijskom računanju i NP problemima**

Evolucijskim računanjem nazivamo granu umjetne inteligencije koja se bavi rješavanjem optimizacijskih problema. Interes za evolucijsko računanje i evolucijske algoritme postao je to veći posljednjih desetljeća kad se broj optimizacijskih rješenja povećao zbog napretka tehničkih i prirodnih znanosti do te mjere da se neki problemi u tim područjima ne mogu riješiti za vrijeme trajanja ljudskog vijeka. Zvuči nestvarno na prvi pogled, no postoji bezbroj problema koji to potvrđuju.

Uzmimo na primjer, problem trgovačkog putnika (*Traveling Salesman Problem*). Njegov cilj je obići sve gradove točno jednom i vratiti se na mjesto od kojeg je krenuo tako da prijeđe najmanje kilometara. Ako trgovac treba obići samo par gradova, nije problem napraviti kombinacije svih mogućih scenarija i odrediti onaj optimalni, no što ako je broj gradova veći od toga? (Slika 1) prikazuje rezultate eksperimenta provedenog na računalu, a iz nje vidimo kako se s povećanjem gradova faktorijelno povećava i vrijeme obrade iscrpne pretrage tog problema. Za samo 15 gradova trebalo bi nam oko 7 dana, dok za 16 ta brojka iznosi 4.7 dana, ovisno o procesoru[4].



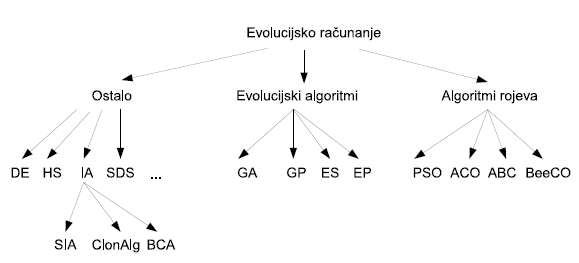
Slika . Ovisnost vremena o broju gradova kod iscrpne pretrage za problem

Takvi problemi se ne javljaju samo u matematici i tehnici, već i u mikrobiologiji, kemiji, financijskoj matematici itd. Kao jedan od najvažnijih problema u mikrobiologiji navodi se predviđanje kako se molekula DNA ukomponira u strukturi proteina. Proteini se sastoje od aminokiselina, i recimo da je njihova duljina d = 100. Budući da na svakom mjestu može doći bilo koja od 20 mogućih aminokiselina, dobit ćemo neizmjerno puno mogućih rješenja. Ispitivanje svojstava svakog rješenja na pojedini virus ili bakteriju bilo bi suludo.

Ovo su samo neki od problema koji nisu rješivi u polinomijalnom vremenu te za koje nemamo do sad efikasne algoritme za njihovo rješavanje, a od sad ćemo takve probleme zvati NP teški problemi. U suštini, svima njima je zajednički opseg rješenja koji je neizmjerno velik, te je većinom jedno od njih optimalno. Kod implementacije algoritama za rješenje ovakvih problema trebalo je primijeniti neku drugu strategiju. Postavlja se pitanje može li se doći do optimalnog ili barem zadovoljavajućeg rješenja u razumnom vremenu a da pritom ne koristimo tehniku grube sile (brute force) ?

Odgovor na to leži u evolucijskom računanju koji se bave upravo takvim problemima. Evolucijsko računanje je vrsta metaheuristike, a to je skup algoritamskih koncepata koji traženje usmjerava na prostor rješenja u kojem se nalaze dobra ili zadovoljavajuća rješenja. Pritom moramo pripaziti što točno znači riječ „traženje”. Nije svejedno traži li se nekakav podatak primjerice u bazi podataka, ili traženje nekog rješenja koje ne možemo predvidjeti. Podatak u bazi možemo naći recimo binarnim stablom, jer je ono sačuvano u točno određenom „pretincu”, dok je traženje općenitog rješenja nekog problema dosta širok pojam te takva informacija nije eksplicitno spremljena, već se rješenja nameću sama po sebi kako se pretraga nastavlja.

Primjeri metaheuristika su algoritmi evolucijskog računanja, tabu pretraživanje, simulirano kaljenje itd. Mi ćemo se u ovom poglavlju osvrnuti samo na algoritme evolucijskog računanja, koji su proizašli iz čovjekove imitacije evolucije te sustava u prirodi koji su se pokazali uspješnim za napredak čovječanstva i živih bića.



Slika . Podjela evolucijskog računanja na glavne glavne i izdvojeni algoritmi

Na (Slika 2) opisana je podjela evolucijskog računanja na razne algoritme[4]. Možemo odmah razlučiti 3 velike grupe algoritama. Jedni od njih su algoritmi rojeva, potaknuti eksperimentima biologa o ponašanju mrava u prirodi. Odmah nakon njih imamo evolucijske algoritme gdje se kao glavni nameću genetski algoritmi, nadahnuti evolucijom, čiji je opseg primjena znatno porastao posljednjih godina. Oni algoritmi koji ne pripadaju u prethodne dvije svrstavaju se u grupu „ostali” između kojih se nalazi umjetni imunološki sustav, o kojem će biti riječi u sljedećem poglavlju. Iz slike možemo također zaključiti kako se svaka navedena podskupina evolucijskih algoritama grana na konkretne implementacije, koje mogu varirati. Konačan rezultat toga je veliko bogatstvo algoritama koje možemo primijeniti na NP teške probleme. Ostaje nam još odgovor na pitanja koliko su ti algoritmi efikasni i možemo li između njih izdvojiti najbolji?

Odgovor na to pitanje je neodređen, te ovisi o problemu s kojim se suočavamo i koliko dobro rješenje će zadovoljiti naše potrebe. Implementacije evolucijskih algoritama temelje se na pronalaženju nekog rješenja ili skupa rješenja koje će biti dovoljno dobro za naš problem. U većini slučajeva to neće biti i optimalno rješenje jer je skup rješenja prevelik, te se takvo nerealno razmatranje svodi na traženje igle u plastu sijena. Što se tiče pitanja koji je algoritam najbolji, ni tu nemamo konkretan odgovor. Ovisno o nekom problemu će izvjesni podskup evolucijskih algoritama imati bolje performanse od drugog podskupa i obratno. Wolpert i Macready su 1995.godine dokazali teorem pod nazivom *no-free lunch*, a radi se o tome da ako algoritam X nadmaši algoritam Y u nekom skupu problema, postoji isto toliko problema u kojem će algoritam Y nadmašiti algoritam X. Prema tome, što više usvojimo principa implementiranja raznih evolucijskih algoritama, veće su šanse da ćemo izabrati onaj pravi koji će nam ponuditi jako dobro rješenje.

Valja napomenuti i to da je kvaliteta pronađenog rješenja također upitna ovisno o tome kako smo računalno implementirali simulaciju nekog događaja. Ako se radi o nekom realnom fenomenu, on se većinom drastično pojednostavi da se može uopće kompjuterski pratiti i da rezultati budu razumljivi. S druge strane, realističniji modeli trebaju mnogo više vremena za simulaciju i pate od istog problema s kojim se susrećemo i u stvarnom svijetu, a to je taj da proizvode toliku količinu podataka i informacija koje se mogu jako teško interpretirati.

**2.2 Prikaz rješenja i njihove modifikacije**

**2.2.1 Prikaz rješenja**

Budući da postoji mnoštvo algoritama za implementaciju rješenja nekog NP problema, mogući su razni prikazi implementacije rješenja. Navodimo neke od njih:

- prikaz bitovima

- prikaz permutacijama i matricama

- prikaz poljem ili vektorom brojeva

- prikaz stablima itd...

Posebnu ćemo pažnju kasnije posvetiti matričnom prikazu s kojim ćemo se baviti kod optimizacije školskog rasporeda. Ovisno o problemu koji razmatramo, različito ćemo pristupiti s implementacijom prikaza rješenja, te on može značajno utjecati na učinkovitost algoritma.

Najjednostavniji je onaj s prikazom niza bitova (bit-string) koji se pokazao i najučinkovitijim u većini primjera gdje bi se on mogao iskoristiti. Također treba uzeti u obzir i način kodiranja bitova (primjerice pogodnije je uzeti Grayev kod nego binarni zapis jer Hammingova udaljenost 2 broja daje vjerniju sliku različitosti nego potonji). Još su česte i implementacije poljima i matricama, kod kojih je zgodno raditi zamjene, inverzije, premještanja i umetanja kako bi se dobila nova, bolja rješenja. Termin se naziva mutacija, a bit će objašnjen kasnije.

Prikaz stablima tipičan je u algoritmima koji zahtijevaju neku odluku, primjerice optimizaciji kolonije mrava u kojem mravi moraju donijeti odluku u kojem smjeru krenuti u potrazi za feromonima.

**2.2.2. Operacije i modifikacije nad rješenjima**

Naravno, da bi došli do valjanog i zadovoljavajućeg rješenja, moramo početi s početnim „rješenjem”. U razmatranju operatora i načina modificiranja rješenja problema poslužit će nam pseudokod Algoritam 1. pomoću kojeg će se objasniti stručni termini koji se koriste u evolucijskim algoritmima. Slika prikazuje jedan općeniti genetski algoritam koji se sastoji od nekoliko ključnih dijelova koji se ponavljaju sve dok se ne izgenerira konačno rješenje. Valja napomenuti da se u većini evolucijskih algoritama koristi slična ili identična terminologija s ponekim razlikama što ih čini lakim za usvajanje kad se jednom dobije osjećaj za koncept samog pojma evolucijskog algoritma. Sami termini i metode pretraživanja oponašaju tijek biološke evolucije.

|  |
| --- |
| **Evolucijski algoritam**  P = **inicijalizirajNasumičnuPopulaciju**();  **evaluirajDobrotu**(P);  t = 0;  **dok** (uvjet\_zaustavljanja nije zadovoljen)  R1, R2 = **odaberiRoditelje**;  **rekombiniraj**(R1, R2);  **mutiraj**(P(t));  evaluirajDobrotu(P(t));  P(t+1) = **odaberiSljedećuGeneraciju**(P(t));  t = t+1;  **kraj**; |

Algoritam . Okvirni prikaz pseudokoda evolucijskog algoritma

1. inicijalizacija

Algoritam 1 se započinje s nasumično generiranom populacijom od n jedinki (u literaturi se može pronaći i naziv kromosom kod genetskih algoritama, te antitijela kod imunoloških). Nakon početne generacije, ulazi se u kružni proces gdje se svaka iteracija procesa naziva generacija. Općenito cijeli algoritam može sadržavati i do preko 1000 generacija.

1. evaluacija dobrote

Svaku jedinku trebalo bi evaluirati na neki način da se dobije povratna informacija o tome koliko je trenutno rješenje prihvatljivo. Ovisno o vrsti problema se prema tome grade različite funkcije dobrote *(engl. fitness function)* koje mogu vraćati različite rezultate, najčešće realne brojeve od 1 do 100 koje aproksimiraju dobrotu trenutnog rješenja. U samoj općenito se definiraju neka ograničenja funkcije pomoću kojeg se generira konačno rješenje koje se optimira kako se algoritam privodi kraju. U pravilu bi dobar genetski algoritam trebao generirati iz generacije u generaciju nove populacije čiji je faktor dobrote sve bolji i bolji.

1. selekcija

Uloga selekcije jest čuvanje dobrih svojstava jedinki i prenošenje na sljedeće generacije, što ga čini jednim od važnijih faktora algoritma. Ovisno o tome koliki je faktor dobrote pojedine jedinke, šanse su veće da će ta jedinka preživjeti u trenutnoj generaciji i opstati u daljnjim iteracijama, po uzoru na prirodnu selekciju i darvinizam.

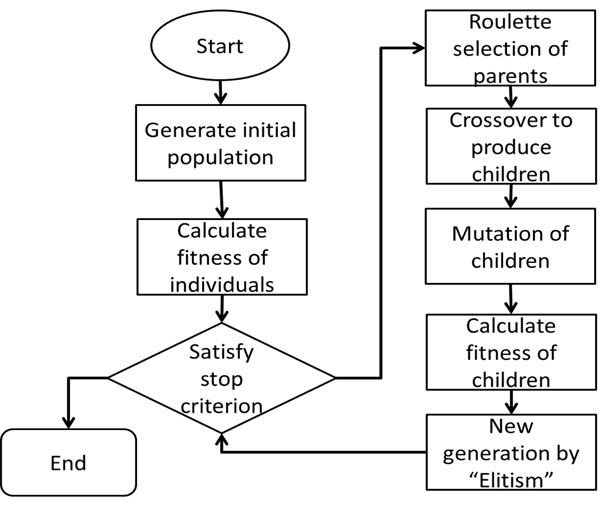
Ovdje bi se moglo diskutirati o tome treba li najsposobnije jedinke samo prenijeti u novu generaciju ili ih križati s ostalima, no prva varijanta bi dovela do prerane konvergencije algoritma i nemogućnošću napretka. Naime, moguće je i da one lošije jedinke sadrže jedan dio genetskog materijala koji je dobar za konačno rješenja, te bi se i lošim jedinkama trebalo također dati neka šansa (doduše dosta manja) za preživljavanje.

1. rekombinacija

Jedan od ključnih dijelova algoritma za samu njegovu uspješnost jest križanje svojstava dvaju rješenja kako bi se dobila bolja i prenijela u novu generaciju. Postoje razne varijacije mada je najučestalija ta da se odaberu 2 jedinke te se pola genetskog materijala jedne i pola druge spoje u jednog jedinstvenog potomka.

1. mutacija

Da bi se očuvala raznovrsnost jedinki također se uvodi i mutacija svake jedinke. Naime, sva dobra svojstva ne trebaju biti generirana odmah u prvoj generaciji te bi zbog tog nesretnog slučaja moglo doći do nepronalaska optimalnog rješenja. Da bi se to izbjeglo, primjenjuje se upravo mutacija gdje svako rješenje ima malu šansu da izmijeni svoj sadržaj odnosno mutira. Treba uzeti u obzir da prevelik postotak šanse za mutiranje može dovesti do negativnog efekta jer se može narušiti funkcija dobrote.



Slika . Dijagram genetskog algoritma. Slika preuzeta iz literature[7].

Na (Slika 3) vidimo i dijagram jednog tipičnog genetskog algoritma. Valja napomenuti da se dijagrami ne razlikuju previše od algoritma do algoritma te da se na kraju sve svodi na slične termine i tehnike pronalaska zadovoljavajućeg rješenja. Za one koji žele znati više preporuča se pregledavanje literature[4] na zadnjim stranicama seminara gdje su detaljnije objašnjeni neki drugi algoritmi koji ovdje neće biti obrađivani.

**2.3. Umjetni imunološki sustav**

Iako se kod evolucijskog računanja za inspiraciju nad algoritmima koristi većinom evolucija, početkom 90.-ih godina razvila se metoda koja je kao predložak koristila upravo imunološki sustav čovjeka. Ta metoda je danas poznata pod terminom Artificial Immune Systems (AIS)[5] , a bazirana je na obrambenom mehanizmu čovjeka. Naime, kad tijelo napadne neki virus ili bakterija, naš obrambeni sustav ima ulogu uništavanja štetne tvari (odsad nazvane antigenom) te čak i pamćenja tog antigena za slučaj da tijelo bude istim napadnuto kasnije. Pamćenje i prepoznavanje se dokazalo kao moćno svojstvo u rješavanju računarskih problema te je AIS otada iznjedrio niz novih algoritama kao što su B-Cell algoritam, Jednostavni imunološki algoritam, a nama će posebno važan biti algoritam klonske selekcije. U sljedećem poglavlju vidjet ćemo kako općenito radi taj algoritam a bit će vidljive i sličnosti s drugim evolucijskim algoritmima

**2.4. Algoritam klonske selekcije (CLONALG)**

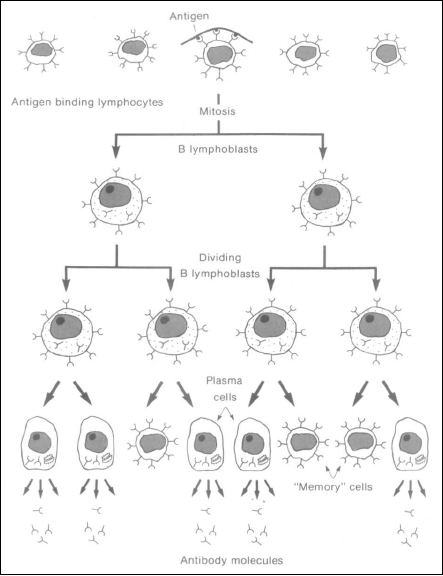
**2.4.1. Motivacija**

Sam algoritam postulirali su Burnet, Jerne i Talmadge koristeći jednu od varijacija imunološkog sustava. Kao inspiracija za algoritam klonske selekcija (CLOnal selection ALGorithm) se koriste B limfocitne stanice koje mogu stvarati antitijela da neutraliziraju antigen. Posebnost B stanica je mogućnost prepoznavanja vlastitog tijela od stranog. Antitijela koje proizvode B stanice kao obrambeni mehanizam sadrže receptore koji se mogu spojiti s antigenom i pomoći u procesu njegova uništavanja.

Definiramo afinitet kao kompatibilnost antitijela s antigenom. Što je afinitet veći, B stanica bolje radi svoj posao te će manje trebati izmijeniti svoj sadržaj (mutirati) kako bi se lakše borila protiv antigena. Također, što je afinitet veći, broj kloniranja za pripadajuću B stanicu postaje veći te se takvim mehanizmom dovodi do uspješnog branjenja od većine bolesti. No to nije sve. Novonastale B stanice ostaju u organizmu te ako neki antigen iste vrste ponovo uđe u organizam, on će reagirati ovaj put još efikasnije, a to svojstvo zovemo memorija.

Iz ovoga vidimo i malu sličnost s Darvinističkom teorijom a to je ta da se jače i sposobnije jedinke više umnožavaju dok slabije moraju efikasno mutirati da bi uopće konkurirale za preživljavanje.

Na (Slika 4) je slikovito prikazan proces dijeljenja B stanica i produciranja antitijela. Sam algoritam klonske selekcije je jako sličan uvodu u ovo poglavlje te se u bitnom ne razlikuje od njega.



Slika . Motivacijski prikaz rada imunološkog sustava. Slika preuzeta iz literature[6]

**2.4.2 Prikaz algoritma i korišteni operatori**

|  |
| --- |
| **CLONALG(Ag, n, d, beta)**  t = 0;  inicijaliziraj P(0) = {x1, x2, ..., xd};  **ponavljaj** dok nije zaustavi(P(t));  **evaluiraj** (P(t), Ag);  P(t) = **odaberi**(P(t), n);  Pclo = **kloniraj**(P(t), beta);  Phyp = **hipermutiraj**(Pclo);  evaluiraj(Phyp, Ag);  P’ = odaberi(Phyp, n);  P(t + 1) = **zamijeni**(P’, Pbirth);  t = t+1;  kraj ponavljanja;  evaluiraj(P(t), Ag);  **kraj**; |

Algoritam . Pseudokod algoritma klonske selekcije. Pseudokod preuzet iz literature[4]

U odsječku Algoritam 2 je prikazan pseudokod Clonal Selection Algorithm-a (CSA). Odmah na početku možemo razlučiti neke dijelove koji ga povezuju s motivacijskim primjerom kao što je kloniranje i hipermutacija, no krenimo redom i rastavimo ga na cjeline i operatore koji su potrebni kako bi algoritam djelovao u skladu s očekivanjima. Nazovimo antigenom funkciju koju optimiramo, a antitijelom neko rješenje za tu funkciju.

1. Inicijalizacija

Algoritam započinje stvaranjem početne populacije **P(0)**. Broj generacija nazvat ćemo parametrom **t** te ćemo ga postupno povećavati za 1 tijekom svakog iteriranja kroz sam postupak algoritma. Konačni rezultat dat će nam opis algoritma u smislu koliko je generacija bilo potrebno za dobiveno rješenje

Naravno, zbog svojstva nasumičnosti inicijalizacije i mutacije, broj generacija će također biti broj koji varira. Nakon inicijalizacije ulazimo u petlju dok se zadani uvjet ne zadovolji. Taj uvjet može biti npr. dovoljno dobar afinitet ili sam broj generacija.

1. Računanje afiniteta (evaluacija)

Računanje afiniteta zasniva se na vrednovanju rješenja s obzirom na funkciju koja se optimira. Ovdje možemo povući poveznicu s genetskim algoritmima gdje se računanje afiniteta zove također i fitness funkcija. Ovisno o problemu koji rješavamo ona može različito izgledati. Možemo uzeti za primjer usporedbu 2 niza bitova jednake duljine. Za to bi nam pogodno bilo računati njihovu Hammingovu udaljenost kao afinitet. Hammingova udaljenost ili funkcija afiniteta u ovom slučaju izgledala bi ovako:

|  |
| --- |
| gdje je δ = (1) |

gdje je D konačni dobiveli afinitet za neko antitijelo, a L duljina niza bitova. Ovdje (1) vidimo da Hammingova udaljenost raste s većom razlikom bitova, pa je afinitet obrnuto proporcionalan dobivenom rezultatu.

1. Kloniranje

Jedan od ključnih svojstava CSA. Afinitet koji je proizašao iz evaluacije svih rješenja pridonosi kloniranju tako da što je afinitet za neku jedinku bolji, veći će biti broj klonova te jedinke. Tako se omogućuje množenje jedinki s boljim svojstvima, dok one s lošijim otpadaju. Jedna od formula (2) koje autori dokazano navode kao pouzdanu jest.

|  |
| --- |
| (2) |

gdje je β parametar kloniranja dok veliko N označava broj antitijela u populaciji dok je i trenutno antitijelo ϵ [1, n]. Po tome je onda totalni broj klonova izračunat kao:

|  |
| --- |
| (3) |

gdje je NC konačni broj klonova, a n je broj odabranih antitijela.

1. Mutacija

Još jedan od ključnih svojstava CSA, negdje nazvan i hipermutacija. Ovisno o afinitetu se mutiraju jedinke tako da što je afinitet za neku jedinku lošiji, ona će doživjeti veći broj mutacija, dok će one s boljim afinitetom doživjeti manji broj, jer su one te koje u tom trenutku nose najbolja stvojstva za konačno rješenje. Još jedan efekt mutacije jest taj što čuva raznovrsnost jedinki. Naime, moguće je nekad zaglaviti u lokalnom maksimumu i ne izvući se nikada iz njega. Mutacije negiraju taj efekt i teže globalnom maksimumu raspršujući se po čitavom skupu rješenja.

1. Selekcija

Nakon mutacije, ponovo se evaluiraju jedinke te se na kraju izabire n najboljih antitijela. Da bi se dodatno proširila raznovrsnost, uvodi se i još izvjestan broj novih antitijela kojim se zamjenjuju antitijela s najmanjim afinitetom. Ovaj korak može i ne mora biti proveden, već je prije napomenuto da postoje razne varijacije koje na kraju pridonose jako malu razliku u samoj izvedbi.

**2.4.3 Parametri klonske selekcije**

Parametara za efikasnost algoritma ima nekoliko, a ovdje ćemo ih navesti. Na prvu odmah možemo uzeti u obzir veličinu populacije koja bitno ne utječe na izvršavanje algoritma. No, broj odabranih jedinki za daljnji prijenos u novu generaciju može biti dvosjekli mač. Naime, veći broj odabranih jedinki značit će usporavanje algoritma, dok će s druge strane postojati više antitijela oko optimuma rješenja.

Drugi parametar koji može poboljšati efikasnost jest faktor kloniranja (β). Što je on niži, traženje će se svesti na neki lokalni optimum, dok će veći faktor kloniranja povećati broj jedinki i usporiti algoritam.

Također možemo i u obzir uzeti broj generacija, no treba biti pažljiv ako se izabere veći broj, jer se može lako zaglaviti u lokalno optimalnoj soluciji koji se teško mogu ukloniti jer je pritom afinitet visok, ali ne toliko visok za izvjestan broj mutacija da učine pomak prema globalnom maksimumu.

Možemo također spomenuti i operator starenja koji se zna često koristiti za uklanjanje problema lokalnog optimuma. Naime, može se zadati i konačni broj generacija koji predstavlja maksimalni život jedinke ukoliko ona preživi sve do tad. Također imamo i soluciju elitističkog pristupa koja bi rješenju s najvećim afinitetom dodijelila na početku svake iteracije nultu starost kako ipak najbolja ne bi ispala iz opsega rješenja. Iz ovog svega vidimo kako se u ne samo CSA već i u drugim sličnim algoritmima pripisuje velika fleksibilnost kod određivanja parametara ili same implementacije algoritma, dok se optimalni rezultati dobivaju pomnim ispitivanjem parametara za svaki problem.

**3. Problem stvaranja školskog rasporeda**

Stvaranje rasporeda je NP težak problem te je njegovo određivanje jedan od najvažnijih problema škola i visokoobrazovnih ustanova. Većinom se za rješavanje takvog problema koristi pojednostavljeni model, no u stvarnosti je broj problema znatno veći. Broj studenata, asistenata i profesora na nekom fakultetu može doseći i brojku od preko 10 tisuća te je upotreba računala time neophodna. Osim problema stvaranja tjednog rasporeda, u takvim ustanovama javlja se i problem sastavljanja rasporeda za ispite, laboratorijske vježbe te drugih nastavnih i izvannastavnih aktivnosti. Budući da se nastava održava na svim fakultetima, evidentno je da rješenja postoje, pitanje je samo na koji način se dolazi do njih

**3.1. Motivacija i prikaz problema**

U praksi se za stvaranje školskog rasporeda koriste brojne metode poput metode bojanja grafova, matematičkog programiranja, metode klastera, zaključivanja na temu baze slučaja i druge. Nameće se pitanje zašto bi evolucijski algoritmi bili pogodni za ovakav problem. Već smo u par uvodnih primjera bili dali dovoljnu intuiciju za takvo nešto, a ovdje ćemo navesti još jedan. Zamislimo da imamo 40 termina na koji možemo staviti 1 ispit, te otprilike 100 kolegija. Za svaki predmet mogli bismo odabrati jedan od 40 termina te bi to isnosilo 40100 kombinacija. Budući da smo zadovoljni i sa dovoljno dobrim rješenjem a ne samo onim optimalnim, evolucijski algoritmi se čine kao logično rješenje problema.

Općenito, generiranje rasporeda aktivnosti se sastoji od pronalaska prave alokacije vremena u limitiranom vremenskom periodu (npr. jednom tjednu) izvjesnog broja događaja (npr. predavanja) i mapiranja istog vremenskog perioda nekom profesoru ili prostoriji na takav način da su sva ograničenja zadovoljena.

Ograničenja koja se koriste kod stvaranja rasporeda aktivnosti obično se dijele u 2 skupine: čvrsta i blaga ograničenja[2] . Čvrsta ograničenja (Tablica 1) se u stvaranju nikad ne mogu prekršiti, dok je dobrota samog rješenja ovisna o tome koliko se blagih ograničenja „pogazi”.

Tablica . Učestala čvrsta ograničenja koja onemogućuju izvođenje nastave

|  |  |
| --- | --- |
| ***No.*** | ***Čvrsta ograničenja*** |
| 1 | Grupa se može smjestiti samo u slobodnu dvoranu |
| 2 | Profesor ne može predavati u dvije grupe istovremeno |
| 3 | Grupa ne može imati više od jednog predmeta u isto vrijeme |
| 4 | Dvorana mora imati dovoljno mjesta da smjesti sve studente |
| 5 | Ako se zahtijeva dodatna oprema, dvorana mora biti posebno opremljena |

Blaga ograničenja su takva da ona nužno ne onemogućavaju izvođenja nastave te imaju niži prioritet. Međutim, iskustvo nalaže da što se više blagih ograničenja pregazi, raspored će izgledati lošije. Naravno, ako pokušamo udovoljiti svim blagim ograničenjima, uzet ćemo danak na izvođenju programa te će se on izvoditi nešto duže od očekivanog budući da prostor rješenja postaje uži udovoljavanjem svakom od ograničenja.

Tablica . Blaga ograničenja koja ne utječu na izvođenje nastave

|  |  |
| --- | --- |
| ***No.*** | ***Blaga ograničenja*** |
| 1 | Nastavnici bi trebali imati nastavu bez „rupa u rasporedu |
| 2 | Studenti bi trebali imati nastavu bez „rupa” u rasporedu |
| 3 | Trebalo bi udovoljiti studentima žele li nastavu ujutro ili popodne |
| 4 | Broj sati ne bi se trebao drastično razlikovati od dana do dana |
| 5 | Uzimanje u obzir u kojoj bi dvorani nastavnici htjeli predavati da poboljšaju predavanja |
| 6 | Raspored bi trebao biti uniformno raspoređen tokom tjedna |

Ovo su samo neka od blagih ograničenja koje vidimo (Tablica 2), kojih ima jako velik broj, a iako nisu prioritetno toliko bitna, mogu utjecati na izgled rasporeda što u konačnici rezultira i uspjehom studenta na fakultetu. Također, osim kreiranja rasporeda, problem se može proširiti i na druge nastavne aktivnosti poput kreiranja rasporeda za laboratorijske vježbe. Nakon što je oformljen raspored, u preostali broj termina i ograničen broj laboratorijskih dvorana trebalo bi razmjestiti studente, poštujući ista ograničenja. Još jedan primjer jest stvaranje rasporeda za međuispite koje može biti krucijalno za uspjeh studenata, a teži se jednolikom rasporedu međuispita nakon nastave.

Iz ovakvog prikaza problema, jasno je da je evolucijsko računanje jedan od boljih pristupa ovom problemu. U nastavku je opisano rješenje problema algoritmom klonske selekcije opisanom u poglavlju 2, te se navode rješenja i modifikacije rješenja.

**3.2. Prikaz rješenja**

Prije nego što navedemo koncept algoritma, morali bismo prikazati i parametre koji se koriste u algoritmu:

Skup termina, T = {t0,t1,t2...,tm}. Jedan termin definira se kao interval u kojem se održava predavanje ili laboratorijska vježba. Većina termina na fakultetu traje oko 2 sata, mada ima i onih od jednog ili od 3. Uz akademsku četvrt, možemo pretpostaviti da je trajanje ukupno 1 sunčani sat.

Skup dvorana, R = {r0,r1,r2,...,rn}. Svaki događaj treba imati prihvatljivu dvoranu za održavanje. Dvorane možemo podijeliti na dvorane za predavanja, laboratorijske prostorije i slično.

Skup predmeta, S = {s0,s1,s2,...,sp}. Predmet se definira šifrom, nazivom te brojem studenata koji su ukljueni u njega.

Skup grupa za predavanja, C = {c1,c2,c3,...,cv}. Studenti se dijele na grupe za predavanja te je predviđeno da svaki student sluša kolegije u svojoj grupi za predavanja. Grupe se dijele ovisno o tome u kojoj se dvorani sluša predavanje ili na neki drugi način. Ako upotrijebimo prvi, mogući broj grupa jest n/KapacitetDvorane ili n/KapacitetLaboratorijskeProstorije.

Na nekim sveučilištima postoji također i više podgrupa za predavanja unutar jedne, npr. napredna grupa, grupa za početnike itd. U ovom radu taj parametar nije uključen.

Skup događaja, E = {e1,e2,...,ew} definiran je s 4 elementa: grupom za predavanja, predmetom, dvoranom i terminom.

Sam prikaz rješenja (Tablica 3.) bit će matričnog oblika Edvorana x termin .Recimo da ima n dvorana, m termina i w događaja za raspoređivanje. E[ri][tj] = ek neka onda predstavlja događaj koji se održava u dvorani ri u terminu tj.

Tablica . Prikaz rješenja školskog rasporeda

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Termin 1 | Termin 2 | ... | Termin m |
| Dvorana 1 |  |  |  |  |
| Dvorana 2 | {P02,MAT1,  D1,12-14} |  |  |  |
| ... | ... |  |  |  |
| Dvorana n |  |  |  |  |

**3.3. Primjena algoritma**

**3.3.1. Inicijalizacija**

Algoritam bi započeo kreiranjem početne populacije s izvedivim rješenjima odnosno antitijelima[3] . Oni se kreiraju neovisno o afinitetu koji će kasnije dobiti. Svako antitijelo mora zadovoljiti sva čvrsta ograničenja. Mogu se postaviti i prioriteti, npr. da se prvo stvaraju rješenja koja uklanjaju najveći broj konflikata. To može biti održavanje predavanja u nekoj fiksnoj dvorani ili održavanje laboratorijske vježbe. Također dvorane s većim kapacitetom će se prije rasporediti nego one za manje grupe. Ostali događaji stvaraju se nasumično bez kršenja čvrstih ograničenja.

**3.3.2. Selekcija**

Selekcijski proces započinje dodjelom afiniteta svakom antitijelu. Budući da su sva čvrsta ograničenja zadovoljena, najbolje je afinitet dodijeliti kao kaznu zbog kršenja blagih ograničenja. Kao rezultat, antitijela bi se poredala u sortiranu listu od najvećeg prema najmanjem afinitetu

|  |
| --- |
| (4) |

U ovom slučaju, k bi bio ukupan broj blagih ograničenja, dok umnožak wi \* ni predstavlja kaznu pomnoženu s blagim ograničenjem. Naravno, kazna će biti 0 ako je ograničenje s indeksom i u rješenju zadovoljeno. Nakon kloniranja i mutacija, n antitijela s najvećim afinitetom se odabiru za novu populaciju Pn.

**3.3.3. Kloniranje i mutacije**

Antitijela u populaciji će se replicirati proporcionalno afinitetu po formuli (3).

Prema fomuli, i predstavlja trenutno antitijelo, β je multiplikator dok je N ukupni broj antitijela. Ako je N = 100 a β = 1, prvo antitijelo bi po tome imalo 100 klonova, drugo 50 itd.

Nakon toga slijedi proces mutacija. Stupanj ili broj mutacija inverzno je proporcionalan afinitetu, zadržavajući najbolja antitijela pri vrhu da se malo po malo optimiraju, a davajući također šansu jedinkama manjeg afiniteta da mutiraju više i time poboljšaju svoju konkurenciju. Mutacija bi se sastojala od uzimanja tablice E iz selekcijskog procesa. Što je manji afinitet, više iteracija mutacija će se primijeniti nad tablicom. Zaustavljanje iteracija može značiti da je došlo do maksimuma iteracija koje je zadao korisnik ili je zadovoljen minimum blagih ograničenja koja se mogu prekršiti.

Svakom iteracijom jedan događaj bi se nasumično premjestio na drugo mjesto. Druga opcija je da se istražuju susjedne pozicije, tražeći time najbolji potez (gradijentni spust). U svakom slučaju, potez ne smije dovesti do neizvedive tablice (ne smije kršiti nijedno čvrsto ograničenje). Nakon toga se računa je li potez doveo do boljeg rješenja ili ne. Ako je to ispunjeno, ili nije bilo promjena, nova tablica će zamijeniti staru te će se algoritam iterativno nastaviti opisanim konceptom.

**3.3.4. Iteriranje programa**

Kao primjer koncepta za rješenje ovog problema uzet je znanstveni rad studenata sa sveučilišta u Singapuru, a na temelju toga odlučio sam napraviti sličnu implementaciju u programskom jeziku Java. U nastavku je prikazan sažeti pseudokod algoritma koji rješava problem stvaranja rasporeda te njegov opis:

|  |
| --- |
| inicijalizirajPodatke(termini, dvorane, studenti, nastavnici)  postaviParametre();  populacija = nasumičnoGenerirajRješenja;  *dok* (brojIteracija < max && afinitetNajboljeg < zadovoljavajuciAfinitet) {  povecajBrojIteracija;  evaluiraj(populacija);  kloniraj(paramKloniranja);  hipermutiraj();  evaluiraj(populacija);  odaberi(brojSelektiranih);  zamijeni(brojZamjenskihSvakeIteracije);  afinitetNajboljeg = dohvatiNajboljeg.dohvatiAfinitet;  *kraj;*  hipermutiraj() :  *za* i = 0 do brojAntitijela  izracunajBrojMutacija();  *za* j = 0 do brojMutacija  E = slučajnoOdaberiDogađaj();  T = slučajnoOdaberiSlobodanTermin();  premjesti(E, T);  noviAfinitet = evaluiraj();  *ako*(noviAfinitet < stari) ponovi();  *inače* nastavi();  *kraj;*  *kraj;* |

Algoritam 3. Okvirni prikaz algoritma klonske selekcije za problem stvaranja školskog rasporeda.[1]

Iz prikaza pseudokoda vidimo kako bi otprilike izgledao jedan algoritam klonske selekcije koji bi mogao riješiti problem stvaranja školskog rasporeda. Na početku se zadaju parametri nužni za kloniranje i selekciju. Generira se početna populacija i ulazi se u proces iteriranje gdje je maksimum zadan ili zadovoljavajućim afinitetom, ili samim brojem iteracija. Na početku se evaluiraju sve jedinke te se računa njihov afinitet. Ovisno o afinitetu, u funkciji *kloniraj(paramKloniranja)* se rješenja kloniraju proporcionalno njihovom afinitetu. Time će se osigurati da će najbolja rješenja imati i najviše svojih primjeraka te će jače konkurirati za ostanak u vrsti. Hipermutiraj() je dio koji zapravo doprinosi pomaku u traženju boljih rješenja. Nasumičnim odabiranjem već ubačenog događaja u raspored, on se premješta na neko drugo mjesto. Prilikom toga, funkcija premjesti(E, T) mora također paziti da ne dođe do kršenja čvrstih ograničenja. Zatim se računa afinitet, temeljem konflikata sa slabim ograničenjima. Ako je afinitet izračunat nakon premještanja lošiji, ponavlja se iteracija, dok se u suprotnom nastavlja proces. Na taj način se omogućuje stvaranje samo boljih rješenja u odnosu na prošla. Nedostatak ovog principa je slabija konvergencija kako vrijeme ide, jer je kasnije sve teže dobiti bolja rješenja.

**3.4. Rezultati**

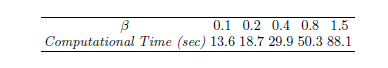
Algoritam je bio isproban na 2 sveučilišta u Singapuru, gdje su uzete sve grupe jedne akademske godine te se na temelju toga kreirao fiktivni raspored [1]. Tjedni raspored sastojao se u jednom od sveučilišta od 116 događaja, a uključivao je 450 studenata. Svaki student je mogao imati maksimalno 30 događaja u tjednu, a događaj je mogao primiti najviše 200 studenata.

Rezultati koji su dobiveni bili su u skladu s očekivanjem, a provedena su i ispitivanja s obzirom na promjenu parametara (n, β) = (broj antitijela, parametar kloniranja).

Kad bi se povisio broj antitijela, manje posla se trošilo oko računanja afiniteta jer veći broj antitijela značio je i veći broj boljih rješenja. S druge strane, taj parametar kočio je brzinu programa, jer se zbog većeg broja antitijela povećalo i trošenje memorije.

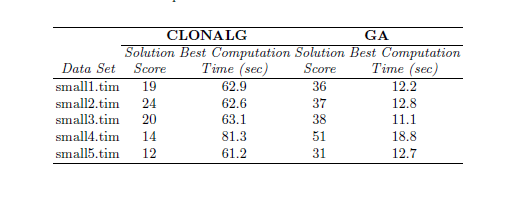
Parametar β bio je redom podešavan na vrijednosti 0.1, 0.2, 0.4., 0.8, 1.5, dok se n fiksirao na N a to je broj populacije. Rezultati su bili opet očekivani kao i u prošlom dijelu, smanjenje troška računanja afiniteta, a povećevanje rada vremena (Tablica 4)

Tablica . Ovisnost vremena računanja o parametru β



U nastavku je dana i usporedba s genetskim algoritmom (Tablica 5). CLONALG je davao bolja rješenja ali uz veće vrijeme računanja zbog već spomenutih ovisnosti o parametru broja klonova i parametra kloniranja. Rezultat se bilježio s brojem prekršenih blagih ograničenja, a rezultate možemo vidjeti u tablici ispod.

Tablica .Usporedba klonske selekcije i genetskog algoritma na istom skupu rješenja. Uz nešto veće vrijeme računanja, klonska selekcija imala je bolji rezultat od GA.



# 4. Zaključak

Ovaj seminar pisan je s namjerom da se oni koji žele znati ponešto o evolucijskim algoritmima upoznaju s jednom od njihovih implementacija, a to je algoritam klonske selekcije.

Iako se većina takvih algoritama svodi na isti kalup, možemo i uvidjeti razlike između klonske selekcije koji vuče inspiraciju iz imunološkog sustava i relacije antigen-antitijelo, te recimo genetskog algoritma koji svoju priču veže uz darvinističku evoluciju. Kad se sve uzme u obzir, možemo navesti veliku prednost klonske selekcije, a to je proširenje lokalnih optimuma rješenja zbog svog načina reprodukcije.

Također, mutacije ovog algoritma uvjetovane su isključivo funkcijom afiniteta, dok je u većini drugih algoritama taj parametar podesiv. Jedan od nedostataka je vjerojatno trošenje memorije na kloniranje, s čime se povećava i vrijeme samog izvođenja programa.

Usprkos tome, primjenjivost ovog algoritma na razne vrste nedeterminističkih problema ogromna je i daje programerima moćno oružje u pretrazi enormnih količina podataka u svrsi pronalaženja zadovoljavajućeg rješenja.

# 5. Sažetak

U ovom seminarskom radu daje se kratki uvod u evolucijske algoritme te se predstavlja jedan iz njihove porodice, a to je algoritam klonske selekcije. Navodi se motivacija te osnovni parametri algoritma. U nastavku se opisuje konkretan problem na kojem je isproban algoritam, a to je problem stvaranja rasporeda sati. Prikazuje se struktura rješenja te sam koncept algoritma, a na kraju su dani i rezultati.

# 6. Literatura

[1] Automatic Timetabling using Artificial Immune System – Yulan He, Siu Cheung Hui, Edmund Ming-Kit Lai

[2] Raspoređivanje nastavnih aktivnosti evolucijskim računanjem – Marko Čupić, Doktorski rad

[3] Constructing University Timetables using an Extended Artificial Immune System – Alexander del Pino, Philip Heck, Christopher Klink, Johannes Klug, J.Philippe Neumann, Nicolai Reuschling

[4] Prirodom inspirirani optimizacijski algoritmi. Metaheuristike – Marko Čupić

[5] Artificial Immune Systems – Mark Read, Paul Andrews, Jon Timmis

[6] Clonal Selection Theory & Clonalg – Jason Brownlee

[7] www.flowcart.logbookloandeals.com/flowchart-of-genetic-algorithm/ - dijagrami raznih algoritama odakle je preuzet genetski algoritam