SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

**SEMINAR**

# Prirodom inspirirani optimizacijski algoritmi

*Filip Kujundžić*
Voditelj: *Tomislav Burić*

Zagreb, svibanj 2020.

# SADRŽAJ

# 1. Uvod

Današnji napredak u razvoju računalne tehnologije rezultira rješavanjem problema čija je složenost znatno veća nego što se to moglo zamisliti prije samo deset godina. No, razvoj računalne tehnologije povlači i pojavu novih i složenijih problema.

Algoritmi pomoću kojih je najjednostavnije pristupiti rješavanju složenih problema su tzv. algoritmi slijepog pretraživanja. Pretražuju skup svih mogućih stanja dok ne pronađu ono koje ispunjava zadane uvjete. Problem kod takvog pristupa je što postoji mogućnost da se traženo rješenje nalazi "na kraju" skupa stanja koji pretražujemo, odnosno potrebno je proći kroz (skoro) sva stanja da bi do spomenutog rješenja došli. Ovdje je vidljivo da je ovaj pristup jednostavno vremenski neprihvatljiv. Vrijeme potrebno da se prođe kroz sva stanja može se mjeriti u milijunima godina. Zbog toga se okrećemo optimizacijskim algoritmima, algoritmima koji ne mogu garantirati da će naći optimalna rješenja nego ona koja su zadovoljavajuće dobra ali u razumnom vremenskom intervalu.

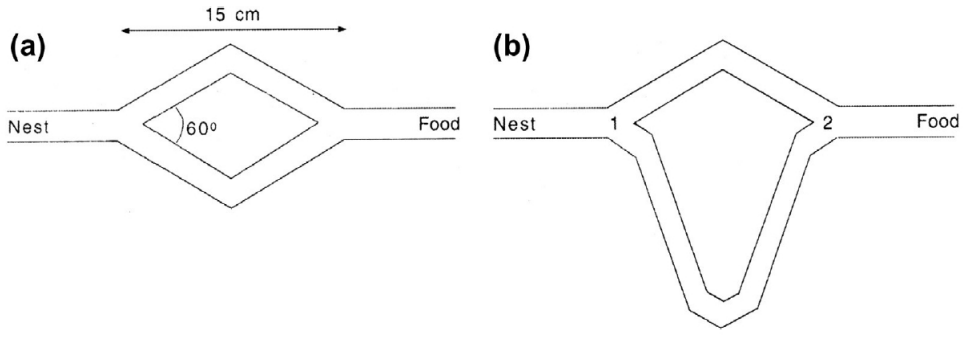
Prilikom proučavanja brojnih optimizacijskih algoritama koji su do danas razvijeni, logično je zapitati se koji je algoritam pretraživanja najbolji. Pri tome se misli na algoritam koji bi potencijalno rješavao sve probleme i to u dovoljno kratkom vremenskom periodu. Wolpert i Macready [1995. i 1997.] su odgovorili na to pitanje - najboljeg algoritma nema. Dakle, za različite probleme, različiti će algoritmi biti najbolji.

Rad opisuje specifičnu skupinu optimizacijskih algoritama, onih koji su inspirirani prirodnim procesima. Opisani su mravlji algoritmi, algoritam roja čestica, umjetni imunološki algoritmi te algoritam diferencijske evolucije. Navedeno je predloženo programsko rješenje problema navedenih algoritama u programskom jeziku Python.

# 2. Mravlji algoritmi

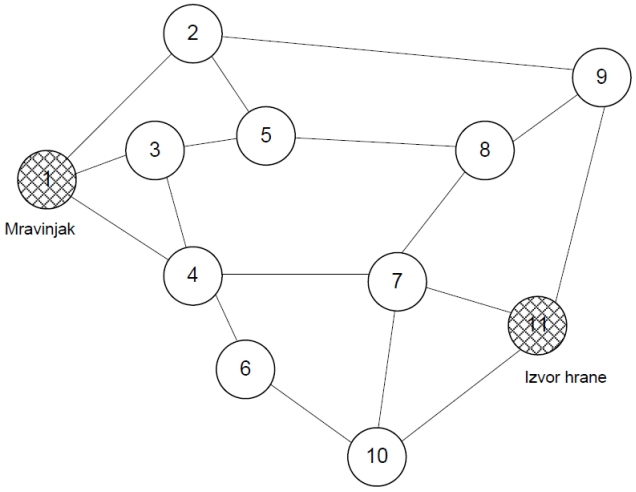
Iako su mravi na prvi pogled vrlo jednostavna bića u usporedbi s čovjekom, možemo primijetiti da su neobično dobro organizirani - od prijenosa hrane do mravinjaka u "mravljim kolonama" sve do organizacije same unutrašnjosti mravinjaka. Zašto su nama mravi zanimljivi? Pokazalo se kako uvijek pronalaze najkraći put između hrane i njihovog mravinjaka te time prenose hranu na najbrži mogući način. Dakle, uspješno rješavaju optimizacijske probleme! Zanimljivo je spomenuti da mravi uopće nemaju razvijen vidni sustav, ili ako imaju, jako je loš.

Deneubourg je sa svojim suradnicima 1989. napravio niz eksperimenata kako bi istražio uzrok takvog ponašanja mrava. Eksperimenti su se sastojali od spajanja mravinjaka i hrane dvokračnim mostom. U prvom eksperimentu (slika 2.1.a), krakovi mosta su bili jednaki i ispostavilo se da se nakon određenog vremena mravi počnu kretati jednim krakom. To se događa jer mravi odabiru smjer svojeg kretanja prema jačini osjeta kemijskog traga - feromona koji ostavljaju drugi mravi. Nekoliko se mravi počne kretati jednim krakom i tako pojačavaju feromonski trag. Time s vremenom sve više i više mrava nastavlja ići jednim krakom. U drugom pokusu (slika 2.1.b) jedan je krak dulji od drugog. Nakon nekog vremena, mravi su odabrali kraći put. Mravima je zatim ponuđen dulji put koji je bio i jedini, te nakon nekog vremena vraćen kraći put. Zbog jače koncentracije feromona, mravi su ostali pri duljem putu.




**Slika 2.1:** Eksperiment dvokrakog mosta

Ovakvo se ponašanje može opisati jednostavnim matematičkim modelom iz 2004. godine (Dorigo and Stützle). Predstavimo tunele kojima mravi trebaju proći od mravinjaka do hrane bridovima grafa (slika 2.2).




**Slika 2.2:** Primjer pronalaska hrane

Na početku, na sve se bridove inicijalno postavi ista (fiksna) količina feromona. Mrav kreće iz čvora 1 i treba odlučiti u koji će čvor ići. Odluku donosi na temelju vjerojatnosti odabira brida $p_{ij}^k$:

$$p_{ij}^k = \begin{cases} \frac{{\tau_{ij}}^{alpha}}{\sum_{l \in N_i^k} \tau_{il}^{\alpha}}, & j \in N_i^k \\ 0, & j \notin N_i^k \end{cases}$$

U izrazu $\tau_{ij}$ predstavlja iznos feromonskog traga na bridu između čvorova $i$ i $j$, a $\alpha$ predstavlja konstantu. Skup $N_i^k$ predstavlja skup svih indeksa svih čvorova u koje je u koraku $k$ moguće prijeći iz čvora $i$. Jednom kad mrav dođe do hrane, zna koliki je put prešao, tj. kolika je dobrota rješenja. Pri povratku u mravinjak, ostavlja feromone tako da ih ostavi više ako je veća dobrota rješenja. No, pokazalo se da ovaj pristup ima i nekih problema. Primjerice, postoji mogućnost da se dogode ciklusi prilikom obilaska grafa, što svakako ne bi željeli. Također, možemo primijetiti da je potrebno ažurirati količinu feromonskog traga nakon što svaki mrav prođe, što vremenski predstavlja problem.

Danas se koristi algoritam Ant System.Kako bi se izbjeglo usmjeravanje pretrage u smjeru koji mravi slučajno odaberu, postavlja se $\tau_0$ koji je nešto veći od količine feromona koju pojedini mrav ostavlja. Mrav odlučuje kojim će putem krenuti na temelju jakosti prethodno deponiranog feromonskog traga te vrijednosti heurističke funkcije. Uvedena je i funkcija isparavanja feromonskog traga kako bi nakon nekog vremena (nekoliko iteracija) put kojim mravi nisu prolazili postao manje prihvatljiv.

# 3. Algoritam roja čestica

Tijekom pokušaja simuliranja kretanja jata ptica, potpuno slučajno, otkriven je algoritam roja čestica (engl. Particle Swarm Optimization). C.W.Reynolds je u svojem radu 1987. godine, promatrao jato ptica kao sustav čestica u kojem svaka ptica predstavlja jednu česticu te u letu poštuje pravila. Ta pravila su: izbjegavanje kolizije s bliskim pticama, usklađivanje brzine leta s njima te pokušaj ostanka u blizini drugih ptica. Na temelju ovoga, a i sličnih radova, Eberhart, Simpson i Dobbins 1996. godine izdaju knjigu u kojoj opisuju uspješnu primjenu algoritma na treniranje unaprijedne umjetne neuronske mreže. Rad o prilagodbi algoritma za rad nad diskretnim domenama objavili su 1997. godine Eberhart i Kennedy.

Algoritam roja čestica je populacijski algoritam - niz populacijskih jedinki (čestica) pretražuju višedimenzijski prostor te temeljem vlastitog i iskustva bliskih jedinki mijenjaju položaj. Svaka jedinka koristi dva faktora u određivanju smjera kretanja - svoje najbolje rješenje (individualni faktor) te najbolje pronađeno rješenje svoje bliske osobe (socijalni faktor). Ako je veći utjecaj individualnog faktora, jedinka radi istraživanje prostora stanja, a ako je veći utjecaj socijalnog faktora, jedinka podešava pronađeno rješenje. Na taj se način postiže kombiniranje globalnog pretraživanja prostora stanja i lokalne pretrage kojom se obavlja podešavanje rješenja.

Na početku algoritma inicijalizira se populacija. Pojedina se čestica smjesti na slučajno odabranu lokaciju te joj se dodijeli slučajno odabrana brzina. Glavni se dio algoritma ponavlja sve dok se ne ispuni uvjet zaustavljanja - pronalazak dovoljno dobrog rješenja ili dostizanje maksimalnog broja iteracija.

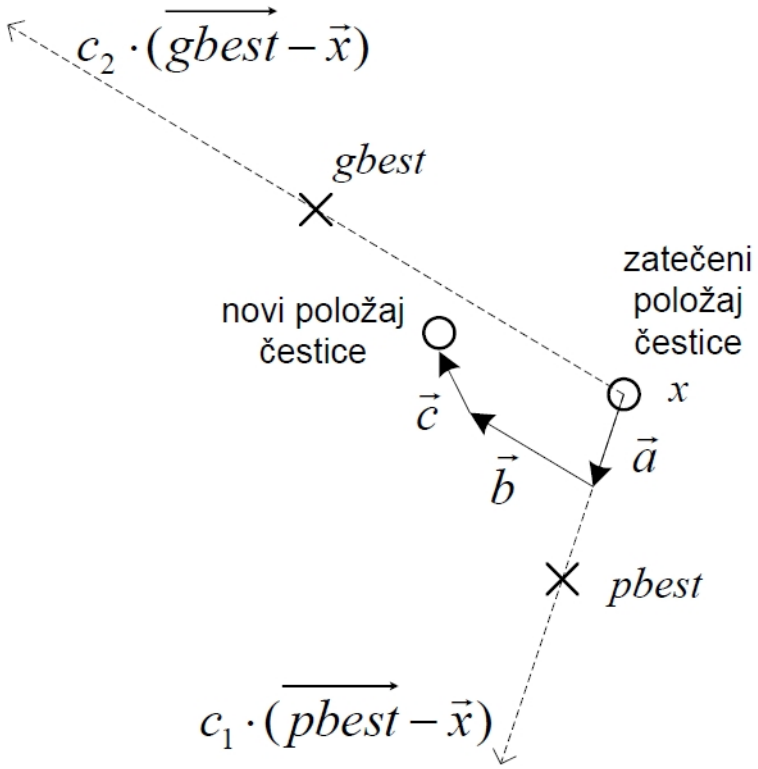
Postupci u glavnom dijelu algoritma:

- svakoj se čestici pridruži vrijednost funkcije u točki koju predstavlja
- uspoređuje se novo rješenje s do tada najboljim; ako je novo bolje, ono postaje najbolje (varijabla $pbest$, engl. particles best solution); pamti se rješenje i vrijednost funkcije u tom rješenju
- u čitavoj se populaciji pronađe najbolje rješenje, i ako je prethodno zapamćeno rješenje lošije, ažurira se na novo pronađeno; navedeno se pamti u polju $gbest$ (engl. global best solution). Također se pamti rješenje i vrijednost funkcije u tom rješenju.
- za svaku česticu ažuriranje trenutne brzine i položaja

Dakle, ažuriranje se obavlja prema izrazu:

$$v_{i,d} = v_{i,d} + c_1 \cdot rand() \cdot (pbest_{i,d} - x) + c_2 \cdot rand() \cdot (gbest_d - x) \tag{3.1}$$

Uobičajena vrijednost faktora $c_1$ i $c_2$ je 2.0. Faktorom $c_1$ regulira se individualni faktor, dok faktor $c_2$ jači naglasak stavlja na socijalni faktor. Na slici 3.1. su prikazane "sile" koje djeluju na kretanje čestice. Rand() je slučajno generirani broj.




**Slika 3.1:** Grafički prikaz pomaka čestice kod algoritma roja čestice

U algoritam moramo ugraditi ograničenja koja će nam omogućiti da algoritam funkcionira upravo onako kako želimo. Na primjer, maksimalna brzina ne smije biti prevelika da ne bismo riskirali "prelijetanje" preko područja u kojem se nalazi rješenje. Isto tako, brzina ne smije biti niti premala jer se čestica u tom slučaju ne može oteti utjecaju lokalnog optimuma. Zato se maksimalna brzina tipično stavlja na 10% do 20% raspona prostora koji se pretražuje. Što su veći brojevi $c_1$ i $c_2$, čestica će biti više vezana uz ostale te će moći manje istraživati. Veličina populacije je između 20 i 30 jedinki. Poželjno je dodati i faktor inercije bi mogli dobiti prvo grubo pretraživanje prostora, a nakon pronalaska zanimljivih područja, njihovo detaljno istraživanje.

# 4. Umjetni imunološki algoritmi

Genetski algoritmi koji simuliraju proces evolucije operatorima križanja i mutacije u svrhu dobivanja najkvalitetnijeg mogućeg genetskog materijala, rezultirali su još jednom novom skupinom algoritama - umjetnim imunološkim algoritmima. Imunološki sustav služi za obranu od antigena. Upravo je taj mehanizam motiv za algoritme iz ove skupine.

Jednostavni imunološki algoritam (engl. SIA - Simple Immunological Algorithm) razvili su 2002. godine Cutello i Nicosia. Može se koristiti za izradu klasifikatorskih sustava te za optimizaciju. Ovdje će biti opisan s optimizacijskog aspekta. Radi s populacijom antitijela, pri čemu svako antitijelo predstavlja jedno rješenje problema koji se optimira. Afnitet pojedinog antitijela (rješenja) prema antigenu (funkciji) tada je predstavljen kvalitetom (tj. dobrotom; engl. fitness) samog rješenja.(Čupić, 2013)

Na početku algoritma se stvara inicijalna populacija antitijela određene veličine. Rješenja se mogu prikazivati dekadski i binarno. U slučaju da je odabran binarni način, jedinke će biti predstavljene nizovima nula i jedinica. Nakon toga se svaka jedinka vrednuje, tj. računa se njena dobrota. Glavna petlja u algoritmu traje dok se ne ispuni uvjet zaustavljanja ili dok se ne prekorači maksimalan broj dozvoljenih iteracija. Svako antitijelo klonira se određeni broj puta, čime nastaje populacija klonova. Slijedi hipermutacija (nasumična promjena receptora antitijela u pokušaju da se antitijelo bolje prilagodi povezivanju s antigenom) svakog člana klonirane populacije. Vrednuju se dobivena rješenja, te se iz unije početne populacije i hipermutirane izabire n antitijela (n je veličina početne populacije) s najvećim afinitetom. Tako izabrana antitijela postaju nova populacija. Iz ovoga se vidi da je algoritam automatski elitistički - samo se najbolje jedinke prenose u novu populaciju.

Još jedan algoritam iz skupine umjetnih imunoloških algoritama je Clonal Selection Algorithm (kratica CLONALG). Algoritam su predložili i razradili De Castro i Von Zuben između 1999. i 2002. godine.

Glavna ideja algoritma slična je kao i kod jednostavnog imunološkog algoritma. Antitijela se podvrgavaju postupku kloniranja proporcionalno njihovom afinitetu te postupku hipermutacije obrnuto proporcionalno njihovom afinitetu. Antitijela su sada nizovi od $n$ elemenata, pa se mogu promatrati kao točke u $n$ dimenzijskom prostoru.

Rad algoritma izgleda ovako. Stvori se početna populacija nekim slučajnim mehanizmom. Zatim se ulazi u petlju koja završava kad se ispuni uvjet zaustavljanja. Nakon vrednovanja, odabiru se antitijela koja će se klonirati. Izvorno, odabiru se sva antitijela, no taj je broj moguće smanjiti. Slijedi kloniranje antitijela tako da antitijela s većom dobrotom dobivaju proporcionalno veći broj klonova. Hipermutacija kloniranih jedinki obrnuto je proporcionalna dobroti antitijela. Nakon hipermutacije računa se dobrota svih antitijela s obzirom na antigen. Određeni broj, $z$ najboljih antitijela uzima se iz hipermutirane populacije za novu populaciju, a antitijela s najmanjom dobrotom zamijene se slučajno generiranim, novim, antitijelima.

Važna stvar kod imunoloških algoritama su vrste operatora. Danas se uobičajeno koriste tri vrste operatora.

- Operator kloniranja

  Stvara populaciju klonova iz postojeće populacije antitijela. Ako je operator statički, svako antitijelo iz izvorne populacije klonira se konstantni broj, $n$, puta. Operator proporcionalnog kloniranja radi tako da ona antitijela koja imaju veću dobrotu proporcionalno toj dobroti klonira. Dakle, ona antitijela s većom dobrotom imat će više klonova. Vjerojatnosno kloniranje definira parametar $p$ temeljem kojeg iz izvorne populacije bira antitijela koja će biti klonirana.

- Operatori mutacije

  U pravilu se operator mutacije primjenjuje nakon operatora križanja. Unosi slijepe promjene nad genima. Statička hipermutacija ograničena je konstantom $c$ te ne ovisi o funkciji dobrote antitijela. U proporcionalnoj hipermutaciji broj mutacija nad jedinim antitijelom proporcionalan je funkciji dobrote antitijela iz trenutne populacije. Inverzno proporcionalna hipermutacija rezultira većim brojem mutacija nad antitijelom manje dobrote. Konačno, hipermakromutacija broj mutacija ne ovisi o funkciji dobrote niti o konstanti $c$. Odaberu se nasumično dva indeksa $i$ i $j$ tako da su manji od duljine antitijela a da je $i$ manji od $j$, te se mutiraju geni između $i - tog$ i $j - tog$.

- Operator starenja

  Glavna svrha ovog operatora je osigurati da antitijelo ne ostane predugo u populaciji. Statički operator starenja definira konstanta koja određuje koliko iteracija antitijelo može preživjeti u populaciji. Nakon toga, antitijelo se briše iz populacije, neovisno o tome kolika mu je dobrota. Klonovima se prepisuje starost roditelja. Slijedi vrednovanje, nakon kojeg se klonovima koji su bolji od svojih roditelja starost resetira na nula. Može se uvesti i elitizam tako da se jedinki s najvećom dobrotom starost također resetira na nula. U stohastičkom operatoru starenja jedinka može biti obrisana i prije isteka vremena, što je definirano vjerojatnošću preživljavanja koje se smanjuje povećanjem starosti jedinke.

# 5. Algoritam diferencijske evolucije

Iako algoritam diferencijske evolucije ne spada u skupinu biološki inspiriranih algoritama, ima puno sličnosti s genetskim algoritmom. To je također populacijski algoritam koji nove potomke stvara uporabom operatora diferencijacije. Razvoj algoritma počeo je 1994. godine kad je Storn u časopisu Dr. Dobbs Journal opisao algoritam genetskog kaljenja. Uslijedio je niz radova, sve do 1999. godine.

Algoritam je u svojoj prvoj inačici bio napravljen za rješavanje optimizacijskih problema funkcija u kontinuiranom prostoru. Zato su sva rješenja D-dimenzijski vektori. Osnovna ideja algoritma je modificirati jedinke trenutne populacije linearnom kombinacijom jedinki iste populacije. U slučaju da su jedinke dosta raspršene, promjene će biti dosta velike. S druge strane, ako su jedinke na maloj udaljenosti jedna od druge, modifikacije će biti male što u konačnici rezultira konvergencijom.

Ako pretpostavimo da rješavamo problem pronalaska vektora $x$ koji minimizira zadanu funkciju $f$, algoritam možemo opisati koracima:

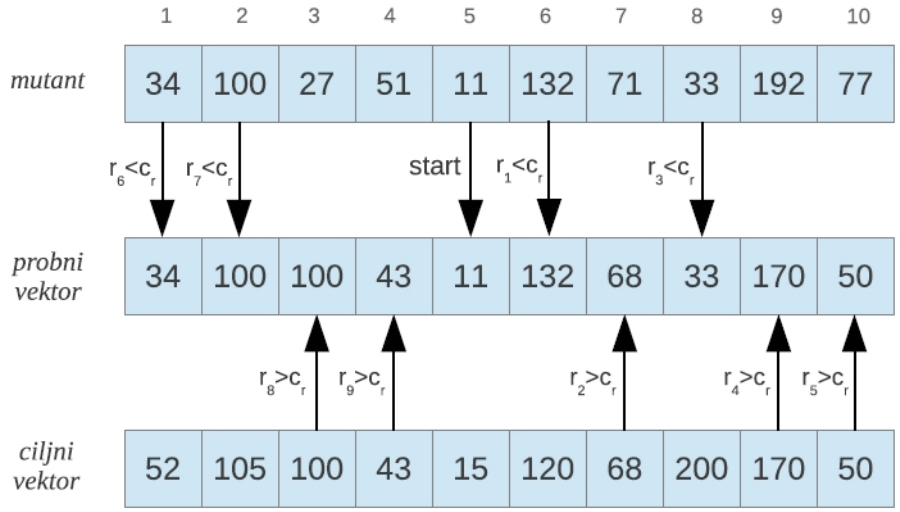
- Stvori početnu populaciju
- Ponavljaj
  - Kreni po svim jedinkama
    - Trenutnu jedinku prozovi ciljnim vektorom
    - Generiraj bazni vektor i mutiraj ga; rezultat je vektor mutant
    - Križaj ciljni vektor i vektor mutant: rezultat je probni vektor
    - Primijeni operator selekcije: u iduću generaciju pošalji probni vektor ako nije lošiji od ciljnog vektora; inače pošalji ciljni vektor
  - Stvorena populacija je nova trenutna generacija

Veličina populacije koja se stvara je $n$, a jedinke su D-dimenzijski vektori. Za svaku jedinku trenutne generacije g potrebno je stvoriti vektor mutant od te jedinke. Taj vektor služi kako bi se stvorila nova jedinka - kandidat za ubacivanje u sljedeću generaciju. Jedinku zovemo probni vektor. Promatramo i-tu jedinku populacije - ciljni vektor. Da bismo dobili vektor mutant, potrebno je stvoriti bazni vektor. Jedna od strategija je slučajno odabrati još tri jedinke iz trenutne populacije. Prvi odabrani vektor nazvat ćemo baznim vektorom,

druga dva nam služe za implementaciju mutacije baznog vektora. Vektor mutant bit će zbroj baznog vektora i skalirane razlike preostala dva slučajno odabrana vektora.

Nakon izgradnje vektora mutanta, provodi se križanje između njega i ciljnog vektora. Rezultat je probni vektor. Eksponencijalno križanje se provodi tako da odaberemo komponentu koja će se sigurno kopirati iz vektora mutanta u probni vektor. Slučajnim mehanizmom uz vjerojatnost, određujemo hoće li se kopirati iduća komponenta. Sve dok je odgovor da, provodi se kopiranje. Prilikom prvog nailaska na odluku ne, kopiranje staje, i preostali dio se preuzima iz ciljnog vektora.

Uniformno (binomno) križanje po konceptu je slično uniformnom križanju binarnih kromosoma kod genetskog algoritma. No, ovdje ne radimo s bitovima nego s komponentama vektora. Postupak počinje odabirom komponente koja će se sigurno kopirati iz vektora mutanta u probni vektor. Na slici je ta pozicija označena sa start. Potom se posredstvom slučajnog mehanizma uz vjerojatnost $C_r$ za svaku komponentu pitamo hoćemo li ju preuzeti iz vektora mutanta. Ako je odgovor da, izvršimo preuzimanje dok je za odgovor ne rezultat preuzimanje iz ciljnog vektora.




**Slika 5.1:** Uniformno križanje

Nakon što je za svaki ciljni vektor izgrađen po jedan probni vektor, probni vektori se vrednuju. Nad parovima (ciljni vektor, probni vektor) primjenjuje se operator selekcije. Operator u sljedeću generaciju propušta probni vektor ako mu je dobrota bolja ili jednaka dobroti ciljnog vektora, inače se propušta ciljni vektor.

# 6. Programsko rješenje

Rješenje analiziranih problema napisano je u programskom jeziku $Phyton$. U nastavku su priloženi i objašnjeni bitni dijelovi kôda razmatranih algoritama. Potpuni je kôd dostupan online u GitHub repozitoriju.[1]

- Mravlji algoritam

```
def spread_pheronome(self, all_paths, n_best, shortest_path):
    sorted_paths = sorted(all_paths, key=lambda x: x[1])
    for path, dist in sorted_paths[:n_best]:
        for move in path:
            self.pheromone[move] += 1.0 / self.distances[move]
```

**Slika 6.1:** Širenje feromona mrava

Metoda $spread\_pheromone$ služi kako bi mrav ostavio feromonski trag za sobom. Kao argument zahtijeva polje svih putova, broj najboljih mrava koji ostavljaju feromone (koji prolaze najkraćim putem) i najkraći put. Svaki mrav ažurira vrijednost feromona u ovisnosti njegovog položaja.

- Algoritam roja čestica

```
def update_velocity(self,pos_best_g):
    w=0.5       # constant inertia weight (how much to weigh the previous velocity)
    c1=1        # cognative constant
    c2=2        # social constant

    for i in range(0,num_dimensions):
        r1=random.random()
        r2=random.random()

        vel_cognitive=c1*r1*(self.pos_best_i[i]-self.position_i[i])
        vel_social=c2*r2*(pos_best_g[i]-self.position_i[i])
        self.velocity_i[i]=w*self.velocity_i[i]+vel_cognitive+vel_social
```

**Slika 6.2:** Računanje brzine u algoritmu roja čestica

[1] https://github.com/filipkujundzic/diplseminar

Prilikom ažuriranja vrijednosti brzine pojedine čestice, u obzir se uzimaju dvije komponente: individualna i spoznajna. U računanju je korištena nasumično generirana vrijednost dobivena funkcijom rand(). Na kraju zbroj brzina obje komponente je brzina čestice.

- Umjetni imunološki algoritam

```
def affinity(vector1, vector2):
    """Compute the affinity (Normalized !! distance) between two features vectors
    :param vector1: First features vector
    :param vector2: Second features vector
    :return: The affinity between the two vectors [0-1]
    """

    d = 0
    for i, j in zip(vector1, vector2):
        d += (i - j) ** 2
    euclidian_distance = math.sqrt(d)
    return euclidian_distance / (1 + euclidian_distance)
```

**Slika 6.3:** Izračun afiniteta (dobrote) čestice

Prilikom izračuna afiniteta (dobrote) u ovom primjeru umjetnog imunološkog algoritma korištena je euklidska udaljenost. Računa se za sve vektore, pa je potrebno iterirati kroz njih dvjema varijablama.

```
def mutate(self):
    _range = 1 - self.stimulation
    mutated = False
    new_vector = []

    # hardcoded min, max values for each feature
    min_features = [4.3, 2.0, 1.0, 0.1]
    max_features = [7.9, 4.4, 6.9, 2.5]

    for idx, v in enumerate(self.vector):
        change = random.random()
        # could have been something like min_features[idx]*(1 - some-percentage) and max_features[idx] * (1 + some-percentage)
        change_to = random.uniform(min_features[idx], max_features[idx])

        if change <= MUTATION_RATE:
            new_vector.append(change_to)
            mutated = True
        else:
            new_vector.append(v)
    return ARB(vector=new_vector, _class=self._class), mutated
```

**Slika 6.4:** Primjer mutacije

Na istom primjeru možemo i pogledati funkciju mutacije kakve se koriste i u algoritmu diferencijske evolucije. Krećemo se po vektoru, te koristimo unaprijed zadane minimalne i maksimalne vrijednosti. Ako je izračunata varijabla change, koja ovisi o nasumičnoj vrijednosti rand() manja od vrijednosti MUTATION_RATE, dodaje se na novi vektor, inače se na novi vektor dodaje vrijednost starog vektora.

# 7. Zaključak

Proučavanje procesa iz prirode omogućilo nam je pristup optimizacijskim problemima s jednog posve novog stajališta, koje se pokazalo jako učinkovito. Prirodom inspirirani optimizacijski algoritmi mogu nam pomoći u rješavanju pojedinih problema koje ne bismo mogli riješiti načinom grube sile. Kako svaki od navedenih algoritama ima nešto detaljniju pozadinsku teoriju, važno je biti upoznat s njom kako bi se sve prednosti algoritama mogle iskoristiti na najbolji mogući način.

Svaki od navedenih algoritama - mravlji algoritam, algoritam roja čestica, umjetni imunološki algoritam te algoritam diferencijske evolucije ima svoje prednosti i nedostatke. Najveći nedostatak je nemogućnost pronalaska ikakvog, pa čak i približno dobrog rješenja. Zato postoji regulacija kojom u glavnoj petlji kontroliramo koliko puta se izvršila (izbjegavanje beskonačne petlje). Prednosti su mogućnost dobivanja novih jedinki operatorima mutacije i križanja, te prilagodljivost velikom broju različitih problema.

Unatoč tome što je većina ovih algoritama relativno nedavno otkrivena i istražena, svi su dostupni za veliki broj programskih jezika u obliku već gotovih knjižnica koje se mogu koristiti prilikom razvijanja vlastitih programskih rješenja u aplikacijama koje zahtijevaju rješavanje problema ove naravi.

# 8. Literatura

Marko Čupić. Prirodom inspirirani optimizacijski algoritmi. metaheuristike., 2013. URL java.zemris.fer.hr/nastava/pioa/knjiga-0.1.2013-12-30.pdf.

# 9. Sažetak

Prirodom inspirirani optimizacijski algoritmi skupina su algoritama koji nam omogućuju rješavanje optimizacijskih problema iako najčešće ne pronalaze optimalno rješenje nego dovoljno dobro rješenje. Ne postoji jedan algoritam koji bismo mogli primijeniti na sve probleme. Svakom problemu potrebno je pristupiti individualno i pronaći odgovarajući algoritam iz skupa poznatih da bi dobili najbolje moguće rješenje.

Najveća prednost ovih (a i općenito svih algoritama iz optimizacijske skupine) je vrijeme potrebno da se izvrše. To je vrijeme realno, za razliku vremena koje se mjeri u milijunima godina da bi dobili rješenje algoritmima slijednog pretraživanja i grube sile. Navedeno se postiže tako da se pretraga usmjerava u smjeru u kojem bi moglo biti željeno rješenje.

Svaki od navedenih algoritama takvo usmjeravanje radi na drugačiji način, no izuzetno uspješno u skupinama problema za koje su namijenjeni. Danas se pojavljuje sve više problema koji su rješivi ovim algoritmima, te ih je nužno poznavati kako bi se riješili problemi i tako dobio željeni rezultat u konačnom proizvodu (primjerice, optimalan rad mobilne aplikacije).