Univerza v Ljubljani
Fakulteta za matematiko in fiziko

Seminar

# Evolucijsko računanje

Avtor: Boštjan Mavrič

Mentor: prof. Rudolf Podgornik

Ljubljana, 28. maj 2012

**Povzetek**

Evolucijsko računanje je veja računalništva, ki raziskuje optimizacijski algoritem, ki deluje na principu evolucije in naravnega izbora. Zaradi svoje robustnosti se takšni algoritmi uporabljajo za tiste optimizacije probleme, kjer mnogi drugi algoritmi odpovejo. V seminarju orišem princip evolucijskega računanja in predstavim sestavne dele evolucijskega algoritma. Podrobneje razložim Hollandovo teorijo standardnega genetskega algoritma in izrek o shemah. Seminar zaključim s predstavitvijo podvej evolucijskega računanja ter primerov njihove uporabe v naravoslovju.

# Kazalo

# 1 Uvod

Darwinova teorija evolucije poskuša pojasniti izvor vrst na našem planetu. Ker so na njem naravni vir omejeni, vsaka populacija, ki je zmožna razmnoževanja, prej ali slej preseže nosilnost okolja, to je velikost populacije, ki jo naravni viri ravno še lahko preživijo. Zaradi tega se med osebki populacije pojavi tekmovalnost, ki povzroči, da imajo pri razmnoževanju prednost bolje prilagojeni posamezniki. Da lahko sestavimo smiselno teorijo evolucije moramo predpostaviti, da se uspešneje razmnožujejo tisti posamezniki, ki so bolje prilagojeni na okolje. To imenujemo naravni izbor. Naravni izbor je prvi izmed postopkov potrebnih za uspešno evolucijo. Poleg njega je potrebno še križanje, to je postopek, pri katerem iz staršev dobimo potomce, ki podedujejo njihove lastnosti. Pri križanju moramo dopustiti napake, kar imenujemo mutacije. Ta postopek nato z iteracijo proizvaja populacijo posameznikov, ki so na okolje prilagojeni bolje od svojih staršev.

Iz raznolikosti življenja na Zemlji vidimo, da je takšen postopek izbire novih posameznikov sposoben ustvariti zelo raznolike primerke, ki zmorejo zelo učinkovito izrabljati danosti okolja. Prav to raznolikost rešitev in robustnost postopka poskušamo pri evlucijskem računanju doseči s posnemanjem evolucije.

# 2 Zgodovina

Prve ideje o uporabi evolucije v računalništvu je v nekaterih svojih člankih že leta 1948 omenil oče sodobnega računalnika, Alan Turing. Uporabnost evolucijskih prijemov je videl predvsem pri razvoju umetne inteligence. V šestdesetih letih prejšnjega stoletja pa so se neodvisno razvile tri šole evolucijskega računanja. V Ameriki so Fogel, Owens in Welsh uvedli evolucijsko programiranje, Holland pa je v tem času neodvisno postavil osnove genetskim algoritmom. Tretja šola izvira iz Nemčije, kjer sta približno v tem času Rechenberg in Schwefel razvila evolucijske strategije. Vse tri smeri so se nato v devetdestih letih prejšnjega stoletje poenotile pod terminom evolucijsko računanje. [3] Te tri različne šole predstavljajo le še imena skupin algoritmov namenjenih reševanju ustreznih problemov. Podrobnejša razlaga razlik je podana v poglavju 5.

To področje računalništva je samostojno slabih dvajset let, saj je bila prva revija, ki je namenjena izključno evolucijskemu računanju (Evolutionary computation), ustanovljena leta 1993, izdaja pa jo MIT.

# 3 Evolucijski algoritem

Evolucijski algoritem je ponavljajoči se postopek, ki deluje na populaciji stalne velikosti. Za problem, ki ga želimo rešiti, sestavimo tako cenilno funkcijo, da ima maksimum pri iskani rešitvi. Pred začetkom izvajanja algoritma izberemo začetno populacijo. Posameznike (kandidate za rešitev) lahko ustvarimo naključno, ali pa sestavimo takšne primerke, za katere predvidevamo, da bodo ustrezni. Posameznike predstavimo z nizom.

Na vsakem koraku postopka, kar imenujemo generacija, vsakemu posamezniku najprej pripišemo neko številsko vrednost, ki jo dobimo, ko na posamezniku izvrednostimo cenilno fukcijo. Da bi dobili novo generacijo, iz prejšnje žrebamo posameznike tako, da je verjetnost za izbiro sorazmerna z vrednostjo cenilne funkcije. Iz izbranih posameznikov nato dobimo potomce, ki predstavljajo novo generacijo.

Pridobivanje novih potomcev je sestavljeno iz dveh korakov. V prvem koraku s kombiniranjem delov staršev sestavimo dva potomca. Najenostavneje to storimo tako, da nakjučno

izberemo mesto v nizu, ki predstavlja posameznike, in zamenjamo del starševskih nizov pred to točko. V drugem koraku z neko majhno verjetnostjo naključno izberemo mesto v nizu in ga spremenimo. Iz populacije nato odstranimo toliko slabih posameznikov, kolikor imamo potomcev. S tako dobljeno novo generacijo nato celoten postopek ponovimo.

Ker je genetski algoritem stohastični iterativni postopek, zanj ne moremo zagotoviti konvergence k rešitvi, kar bi nam omogočalo zapis pogoja za ustavljanje. Namesto tega izvajanje algoritma raje omejimo s številom generacij ali pa z pojavitvijo posameznika z dovolj visoko vrednostjo cenilne funkcije.

Sedaj, ko v grobem poznamo evolucijski algoritem, si še nekoliko podrobenje poglejmo njegove sestavne dele.

## 3.1 Predstavitev posameznikov

Izbira predstavitve je prvi korak pri sestavljanju genetskega algoritma, s katerim želimo reševati naš problem. Z izbiro predstavitve posredno izberemo tudi predstavitev vseh drugih genetskih operatorjev. Strukture, ki predstavljajo možne rešitve našega problema imenujemo *fenotipi*, in so elementi množice $\mathcal{S}$, njihovo predstavitev v genetskem algoritmu pa *genotipi*, ki so elementi množice $\mathcal{G}$. Izbira predstavitve torej pomeni izbiro injektivne preslikave $\rho : \mathcal{S} \mapsto \mathcal{G}$, ki fenotipe preslika v genotipe. Ta preslikava je v večini primerov tudi bijektivna.

Poglejmo si to na enostavnem primeru. Recimo, da želimo s pomočjo genetskega algoritma najti največje celo število na intervalu $[0, 255]$. Za predstavitev pa izberimo računalniku najbolj naravno možnost, torej zapis v binarnem sistemu. Številu 18, ki je fenotip, torej pripada genotip 00001010.

## 3.2 Cenilna funkcija

S cenilno funkcijo opišemo okolje, ki se mu mora prilagoditi fenotip. Cenilna funkcija $f : \mathcal{S} \mapsto \mathbb{R}$ vsakemu fenotipu priredi realno število. Ker je verjetnost za razmnoževanje fenotipa sorazmerna z vrednostjo cenilne fukcije, s tem definiramo, katere lastnosti fenotipa razumemo kot prednosti.

Genetski algoritmi so navadno definirani tako, da maksimizirajo cenilno funkcijo, kar bi lahko bilo nezaželeno, če bi jih želeli uporabiti za minimizacijo. Takšen strah je odveč, saj je vsak minimizacijski problem enostavno prepisati na maksimizacijskega in obratno.

## 3.3 Populacija

Populacija je večkratna množica fenotipov. To pomeni, da dovolimo, da se posamezen fenotip v populaciji pojavi večkrat. Evolucija spreminja populacijo, saj se med delovanjem evolucije spreminja njena sestava, medtem ko ostajajo fenotipi enaki; spreminja se le njihova zastopanost v populaciji.

Če je namen evolucijskega algoritma opazovanje procesa evolucije, potem lahko v populacijo uvedemo še mero oddaljenosti ali pa sosednostno relacijo, s čimer poskušamo posnemati prostorske odvisnosti, ki jih najdemo v biološkem svetu. Takšni pristopi se razen v biologiji redko uporabljajo.

*Raznolikost populacije* je mera za število različnih posameznikov, ki so prisotni v populaciji. Pogosto za mero izberemo število različnih vrednosti cenilne funkcije, število različnih fenotipov ali število različnih genotipov. Pri izbiri velja imeti pred očmi dejstvo, da samo ena vrednost cenilne funkcije ne pomeni, da je prisoten samo en fenotip, in samo en fenotip ne pomeni, da je prisoten samo en genotip. V obratno smer pa velja; samo en genotip povzroči samo en fenotip, in zato samo eno vrednost cenilne funkcije.

## 3.4 Genetski operatorji

Z genetskimi operatorji poizkušamo posnemati biološko dogajanje pri evoluciji. Z uporabo teh operatorjev dobimo iz genotipa posameznika prejšnje generacije genotip posameznika, ki pripada naslednji generaciji. Kot že samo ime pove so to operatorji, ki delujejo na genotipu. Takšno poimenovaje izvira iz dela Hollanda [4], vendar pa sodobnejši avtorji [2] v to skupino štejejo tudi izbiro staršev in nadomeščanje stare generacije z novo, ki pa sta operatorja, ki delujeta na populaciji. Takšna razširitev je razumljiva, saj s tem zajamemo vse operatorje s katerimi iz prejšnje populacije dobimo naslednjo, zato sem se odločil, da takšno poimenovanje uporabim tudi v tem seminarju.

### 3.4.1 Izbira staršev

Vloga izbire staršev je, da izberemo posameznike iz katerih bomo tvorili naslednjo genracijo. Ker hočemo prilagojenost populacije izboljševati želimo, da je verjetnost za izbiro boljših posameznikov večja, vendar pa moramo pri tem paziti, saj lahko, če smo pri tem preveč pohlepni prehitro zmanjšamo raznolikost populacije, in s tem obtičimo v katerem izmed lokalnih minima. Najpogosteje uporabljeni postopki za izbiro staršev so

**Sorazmerna izbira** Takšno način je uporabil Holland v svojem izreku o shemah (razložen v razdelku 4.1). Starše izberemo naključno tako, da je pričakovano število izbir posameznika z vrednostjo cenilne funkcije $f_i$ enako $n_i = f_i/\overline{f}$, kjer je $\overline{f}$ povprečna vrednost cenilne funkcije celotne populacije.

**Linearna izbira po rangu** Za ta namen posameznike razvrstimo po vrednosti njihove cenilne funkcije. Verjetnost za izbiro posameznika z rangom $r$ v populaciji z $N$ posamezniki je $p(r) = 2(N-r)/N(N+1)$. Čeprav je takšen način izbire pogosto uporabljen v praksi, saj zmanjšuje pomembnost visoko prilagojenih posameznikov v populaciji in s tem preprečuje prehitro konvergenco, pa je v teoriji genetskega računanja neuporaben, saj ne ustreza Hollandovemu izreku o shemah. Na podoben način lahko uvedemo nelinearno izbiro po rangu, ki pa prej omenjenega zadržka ne odpravi.

**Izbira s turnirjem** Pri izbiri s turnirjem iz populacije naključno žrebamo $k$ posameznikov ($k$-turnir) in izmed njih izberemo tistega, ki ima najvišjo vrednost cenilne funkcije.

### 3.4.2 Križanje

Pri križanju informacijo, ki jo nosita genotipa staršev, rekombiniramo v genotipa dveh potomcev. Če bi na populaciji izvajali samo izbiranje, brez križanja, bi bila najboljša vrednost cenilne funkcije, ki bi jo lahko dobili, kar enaka najboljši vrednosti prisotni v začetni populaciji. S križanjem namreč poskušamo iz že poznanih rešitev producirati nove, z upanjem da bodo boljše. Za križanje običajno izbremo dva starša, kar pa je nekoliko antropocentrično, saj se izkaže, da lahko z izbiro večih staršev dosežemo boljšo konvergenco postopka, [3] vendar pa se takšen pristop le redko uporablja.

Operator križanja je odvisen od reprezentacije. Najenostavnejši in najbolj nazoren je operator pri standardnemu gentskemu algortimu. Med običajnimi postopki so

**Enotočkovno križanje** Genotipa staršev prerežemo na istem naključno izbranem mestu. Na ta način dobimo par levih in par desnih odsekov. Genotipa potomcev dobimo tako, da desna odseka zamenjamo.

**Večtočkovno križanje** je enostavna posplošitev enotočkovnega križanja. V tem primeru genotip prerežemo na več mestih in nato odseke zamenjamo. V nekaterih primerih je večtočkovno križanje boljše, vendar lahko postane v poznejših generacijah preveč agresivno in ne izboljšuje rešitev, zato je priporočljivo število prerezov manjšati, ko postaja populacija vse bolje prilagojena. Kot bomo videli, ponuja za ta nasvet Hollandova teorija jasno razlago.

**Enakomerno križanje** Nov genotip sestavljamo tako, da za vsako mesto posebej izžrebamo starša, čigar gen bomo uporabili na tem mestu. Posledica takšnega naključnega izbiranja genov je, da se v populaciji ne tvorijo odseki genotipa, ki so sami s sabo dobro usklajeni, kar pomeni, da je zaradi tega konvergenca algoritma precej počasnejša.

Ker je križanje ključen operator evolucijskega računanja, moramo pri njegovi konstrukciji uporabiti čim več znanja o problemu. To lahko dosežemo z večjo domiselnostjo pri sestavljanju operatorja ali pa zgolj z izbiro primernih točk za rezanje genotipa.

### 3.4.3 Mutacija

Operator deluje na enem samem genotipu in naključno z majhno verjetnostjo spremeni vrednost vsaj enemu izmed njegovih alelov. Tako kot križanje je implementacija mutacije močno odvisna od problema, ki ga skušamo rešiti, saj lahko mimogrede dobimo rešitve, ki so prepovedane.

Mutacija je pomembna zato, ker nam zagotavlja ergodičnost iskanja, torej da lahko algoritem v svojem teku res doseže vse točke iskalnega prostora, kar je potreben pogoj za vse postopke, s katerimi želimo iskati globalne ekstreme.

### 3.4.4 Nadomeščanje posameznikov

Po tem, ko smo zgenerirali nove posameznike jih moramo še vstaviti v populacijo. Ker zahtevamo, da je število posameznikov v populaciji vseskozi enako, moramo iz populacije odstraniti ustrezno število posameznikov prejšnje generacije. Nekateri uveljavljeni postopki so

**Menjava generacij** Na vsakem koraku celotno populacijo zamenjamo z novo. V tem primeru se lahko zgodi, da se prilagojenost najboljšega posameznika v novi populaciji zmanjša. Na ta način tudi preprečimo prevlado nadpopvrečno dobro prilagojenih posamenikov in se s tem obvarujemo pred prehitro konvergenco k kateremu izmed lokalnih minimumov.

**$n$-elitizem** Najboljših $n$ posameznikov iz prejšnje generacije je nespremenjenih prisotnih v naslednji generaciji. Takšna izbira omogoča nesmrtnost najboljšega posameznika. Prezgodnjo konvergenco lahko preprečimo tako, da tudi na teh izbranih $n$ posameznikov delujemo z mutacijo. V tem primeru postopek imenujemo *šibki elitizem*.

**Izbris $n$ najslabših** Pri zamenjavi zamenjamo le $n$ najslabših. Če je $n$ enak velikosti populacije je to že prej omenjena menjava generacij, če pa je $n = 1$ govorimo o nadomeščanju v stacionarnem stanju.

**Izbris $n$** Namesto, da bi izbrisali $n$ najslabših iz populacije odstranimo $n$ naključno izbranih posameznikov. Takšen postopek sicer upočasni konvergenco algoritma, vendar pa je učinkovit način za izogibanje prezgodnji konvergenci.

**Nadomeščanje s turnirjem** Iz trenutne populacije izberemo $n$ posameznikov. Enako število posameznikov izberemo tudi iz potmcev. Med izbranima skupinama nato izvedemo tekmovanje in zmagovalce postavimo v novo populacijo.

# 4 Teoretične osnove genetskih algoritmov

Čeprav je kvalitativno zelo enostavno razumeti delovanje genetskih algoritmov, pa se ti zaradi svoje kompleksnosti še vedno upirajo natančni matematični razlagi. Ker je v vsaki implementaciji genetskega algoritma prisotnih veliko prostih parametrov, je dobrodošla vsakršna teorija, ki nam določi izbiro parametrov algoritma, za katere bo ta gotovo konvergiral. Vsaj enako koristna pa je teorija, če nam razloži zakaj algoritem ne najde rešitve.

## 4.1 Hollandova teorija standardnega genetskega algoritma in izrek o shemah

Teorija genetskih algoritmov, ki jo je John H. Holland postavil v knjigi [4] predstavlja osnovno delo in izhodiščno točko mnogih poznejših analiz genetskih algoritmov. Ker je avtor knjigo pisal tudi z mislijo na biološke sisteme, je v njej analiziran poseben primer evolucijskega algoritma, ki je pozneje dobil ime stanardni genetski algoritem. Analiziran algoritem uporablja predstavitev fenotipov z binarnim nizom konstantne dolžine, enotočkovno križanje in sorazmerno izbiro staršev. Poglejmo si sedaj nekaj bistvenih ugotovitev njegove teorije.

Shema je predloga, ki opiše podmnožico nizev, ki so enake na določenih mestih. V iskalnem prostoru je to neka hiperploskev. Ker uporabljamo binarni zapis lahko shemo zapišemo kot niz znakov iz abecede {0, 1, #}, kjer simbol # predstavlja mesto, na katerem vrednost ni določena, torej lahko zavzame katerokoli, 0 ali 1. Kot primer si poglejmo shemo [0##1]. Tej shemi ustrezajo nizi [0111], [0101], [0011] in [0001]. Velja tudi obratno, vsakemu nizu lahko pripišemo več shem. Nizu [1101] tako ustrezajo sheme [#101], [1#01], [11#1]... Enostavno je pokazati, da vsak binarni niz z dolžino $l$ ustreza $2^l$ shemam. Občutek za število shem, ki jih želimo opazovati dobimo, če izračunamo število možnih shem v algoritmu, ki uporablja binarne nize dolge 200 znakov. V tem primeru vidimo, da posameznik ustreza $2^{200} \approx 3 \cdot 10^{95}$ shemam, kar je več kot je število atomov v vesolju.

Za abecedo s $k$ črkami in dolžino nizov $l$ obstaja $(k+1)^l$ shem. V populaciji $N$ posamenikov najdemo kvečjemu $Nk^l$ shem, saj se lahko zgodi, da enake sheme ustrezajo različnim posameznikom. V populaciji {[101], [111]} lahko tako najdemo samo osem shem, medtem ko v populaciji {[000], [111]} najdemo 15 shem, saj se ponovi le shema [###]. Prav tako se med tekom algoritma lahko začnejo v populaciji pojavljati enaki posamezniki, kar še naprej zmanjša število v njej prisotnih shem.

Lastnosti shem za potrebe Hollandove teorije opišemo z dvema lastnostima. Prva je red sheme $o(H)$ definiran z

$$o(H) = l - l_{\#}, \tag{1}$$

kjer je $l_{\#}$ število simbolov # v nizu, $l$ pa dolžina niza. Red sheme nam torej pove število predpisanih mest v shemi. Drugo lastnost pa imenujemo aktivna dolžina sheme $\delta(H)$, to je razdalja med skrajnim levim in skrajnim desnim predpisanim mestom. V primeru enotočkovnega križanja predstavlja število možnih točk prereza, ki shemo spremenijo. Vidimo, da ima shema [#11##0##] red 3, aktivno dolžino pa 4. Tako smo si sestavili teoretično ogrodje, s katerim lahko zapišemo Hollandov izrek o shemah.

Predpostavimo, da je posameznikova verjetnost za razmnoževanje sorazmerna z njegovo ustreznostjo $f_i$. V tem primeru se število posameznikov $m(H,t)$, ki ustrezajo neki shemi $H$, v populacijah ob času $t+1$ izraža s tistim iz časa $t$ kot

$$m(H, t+1) = m(H,t)\frac{f_H(t)}{\langle f(t)\rangle}, \tag{2}$$

kjer je $f_H(t)$ povprečna vrednost cenilne vseh genotipov, ki ustrezajo shemi $H$ in $\langle f(t)\rangle$ povprečna vrednost cenilne funkcije genotipov v populaciji ob času $t$. Če predpostavimo, da neka shema

zadrži povprečno vrednost za $c\langle f(t)\rangle$ višjo od povprečne vrednosti cenilke na populaciji $t'$ generacij, potem vidimo, da število posameznikov, ki ustrezajo takšni shemi, narašča eksponentno

$$m(H, t + t') = (1 + c)^{t'} m(H, t). \tag{3}$$

Posledica tega rezultata je, da se v populaciji prisotnost shem z nadpopvprečno vrednostjo cenilne funkcije veča eksponentno. Takšnega obnašanja v resnici žal ne opazimo, saj se z večanjem števila nadpopvrečnih shem začne zviševati tudi povprečna vrednost cenilke na populaciji in s tem manjšati konstanta $c$, kar ustavi eksponentno rast. Rezultat nam tudi predstavlja namig, zakaj se genetski algoritmi obnašajo tako dobro, kot se. S tem, ko se v populaciji veča število ustreznih shem, se povečuje tudi število preizkusov genotipov, ki jim ustrezajo. Tako algoritem samodejno bolje razišče okolico točk v iskalnem prostoru z visoko vrednostjo cenilne funkcije.

Poglejmo, kako se v zgornjo enačbo vnese še prispevek križanja. Vpliv križanja genotipa na prisotne sheme, je močno odvisen od sheme, ki jo opazujemo. Vzemimo genotip [011011] in opazujmo, kaj se pri križanju zgodi s shemama $H_1 = [\#\#10\#\#]$ in $H_2 = [0\#\#\#\#1]$. Če križanje izvedemo za drugim mestom [01|1011] opazimo, da bo shema $H_1$ zagotovo spet prisotna v populaciji, medtem ko je prisotnost sheme $H_2$ odvisna od drugega genotipa, s katerim križamo. Če točko križanja vsakokrat izberemo naključno, potem je verjetost $P_H$, da shema $H$ ne bo prisotna v naslednji generaciji gotovo manjša kot

$$P_H < p_c \frac{\delta(H)}{L - 1}, \tag{4}$$

kjer je $p_c$ verjetnost za križanje. Tako lahko zvezo (3) izboljšamo z upoštevanjem križanja in navzdol ocenimo število $m(H, t)$ z

$$m(H, t + 1) \geq m(H, t) \frac{f_H(t)}{\langle f(t)\rangle} \left(1 - p_c \frac{\delta(H)}{L - 1}\right). \tag{5}$$

Upoštevati moramo še vpliv mutacije. Če je verjetnost za mutacijo nekega mesta v nizu $p_m$, potem je verjetnost $P_m$, da se shema $H$ zaradi mutacije ne spremeni

$$P_m = (1 - p_c)^{o(H)}. \tag{6}$$

Ker je verjetnost za mutacijo zelo majhna, lahko izraz razvijemo in ga dodamo v enačbo (5). S tem dobimo Hollandov izrek o shemah [4].

$$m(H, t + 1) \geq m(H, t) \frac{f_H(t)}{\langle f(t)\rangle} \left(1 - p_c \frac{\delta(H)}{L - 1} - p_m o(H)\right). \tag{7}$$

Vidimo, da se v populaciji eksponentno hitro množijo sheme s kratko aktivno dolžino, nizkim redom in visoko povprečno vrednostjo cenilne funkcije. Takšnim shemam pravimo *gradniki*. Iz tega opažanja sledi hipoteza o gradnikih, ki pravi, da genetski algoritem raziskuje rešitve blizu optimalne, tako da preiskuje, združuje in znova preiskuje nadpovprečne sheme s kratko aktivno dolžino in nizkim redom. Hipoteza nam pove, da je izbira predstavitve posameznikov bistvenega pomena, saj mora omogočati tvorbo gradnikov.

Dokaz te hipoteze je tema mnogih raziskav, toda zaradi kompleksnosti algoritma se lahko, razen za trivialne probleme, sklicujemo le na empirične rezultate.

Na tem mestu povzemimo še en Hollandov rezultat. Na vsakem koraku genetskega algoritma je v populaciji prisotnih med $2^l$ in $N2^l$ shem. Nekaj izmed teh shem algoritem uporabi za iskanje optimalne rešitve in jih križanje in mutacija ne zmotita. Take sheme imenujemo *koristne*, njihov delež v populaciji pa eksponentno raste. Holland je ocenil, da je v populaciji $N$ posameznikov takšnih shem $\mathcal{O}(N^3)$. To je imenoval *implicitni paralelizem*, saj algoritem te sheme obravnava navidezno samodejno, brez poseganja uporabnika. Kasneje je bilo pokazano, da ta ocena velja le za $N \propto 2^k$ [5].

### 4.2 Ostali pristopi in drugi pomembni rezultati

Holland je svojo teorijo lahko dobro razvil, ker je z njo obravnaval poseben razred evolucijskih algoritmov, ki je, razen v biologiji, zelo redko uporabljan. Drugi algoritmi, ki so uporabni v praksi, pa so pogosto preveč zapleteni, da bi jih bilo mogoče zaobjeti v tako elegantno teorijo.

Izmed mnogih drugih pristopov, ki med drugim vključujejo teorijo dinamičnih sistemov, redukcionistični pristop in statistično fiziko, so še najbolj uporabni rezultati pridobljeni z uporabo analize verig Markova. Takšna analiza je mogoča za evolucijske algoritme, ki delujejo na končnih iskalnih prostorih. Stanja verige predstavljajo različne možne populacije. Z ustreznim modelom nato določimo prehodne matrike za genetske operatorje. S takšnim pristopom je mogoče pokazati, da vsak evolucijski algoritem, ki optimizira funkcijo nad končnim iskalnim prostorom skonvergira k optimalni vrednosti z verjetnostjo 1, če hkrati velja da

- je vsak posameznik z verjetnostjo različno od nič lahko izbran za starša,
- vsak posameznik lahko z neničelno verjetnostjo preživi,
- za izbiro preživelih uporabljamo elitizem in
- z genetskimi operatorji lahko z neničelno verjetnostjo sestavimo vsakega posameznika.

Vidimo, da problem optimalne izbire parametrov, kljub znatnim teoretičnim naporom ostaja brez rešitve. Tako nam ostanejo le hevristični načini določanja parametrov in moramo za vsak problem posebej določiti parametre, ki zagotavljajo dobro konvergenco. Da neka čudežna izbira parametrov, ki bi enako dobro delovala za vse možne evolucijske algoritme, ne obstaja, nam zagotavlja "No free lunch theorem", ki pravi, da je uspešnost povprečena po vseh možnih cenilnih funkcijah enaka za vsak optimizacijski algoritem.

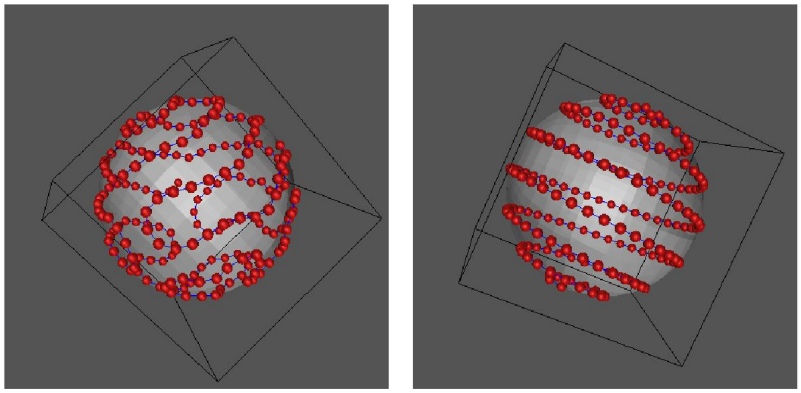
## 5 Primeri praktične uporabe

Od svojih začetnih dni je evolucijsko računanje našlo uporabo v različnih področjih naravoslovja, vsako področje pa si je idejo evolucijskega algoritma priredilo nakoliko po svoje. Poglejmo si eno izmed bolj grobih delitev evolucijskega računanje ter primere uporabe.

### 5.1 Genetski algoritem

Z genetskimi algoritmi označujemo postopke, ki najnatančneje povzemajo ideje evolucijskega algoritma. Posameznike predstavimo z nizom celih ali realnih števil. Mutacijo izvedemo ali z drsenjem (vrednosti v nizu prištejemo ali odštejemo neko majhno število) ali pa z naključno izbiro. Za križanje namesto večtočkovnega križanja pogosto uporabimo aritmetično rekombinacijo. Pri tem postopku izberemo neko točko v nizu in v oba potomca na mestih, ki so desno od izbranega, postavimo vrednost $\alpha x_i + (1-\alpha)y_i$, kjer sta $x_i$ in $y_i$ vrednosti iz starševskih nizev, $\alpha$ pa naključno izbrano število z intervala [0, 1].

Genetski algoritmi so presenetljivo učinkoviti za reševanje Tomsonovega problema [7]. Na tem primeru lepo vidimo robustnost genetskega algoritma. Avtorji namreč križanje izvedejo tako, da dve starševski rešitvi razpolovijo z ravnino in nato zamenjajo hemisferi. Mutacijo nato izvedejo tako, da energijo minimizirajo z metodo konjugiranega gradienta. Takšna metoda je zelo učinkovita za iskanje globalnih minimumov, saj se mnoge druge metode (simulirano ohlajanje) ujamejo v kakšnega izmed lokalnih minimumov. Takšen algoritem lahko nato uporabimo tudi za reševanje Tomphsonovega problema za povezane naboje [8].



Slika 1: Dve različni razporeditvi povezanih nabojev na površini krogle dobljeni z uporabo genetskega algoritma. [8]

### 5.2 Evolucijske strategije

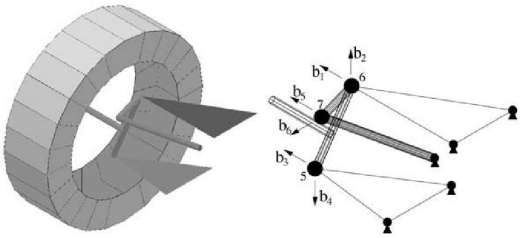
Pri evolucijskih strategijah genom predstavimo z nizom realnih števil, križanje pa je večtočkovno. Posebnost te družine evolucijskih algoritmov je mutacija. Izvedemo jo tako, da z neko verjetnostjo $i$-temu mestu v nizu prištejemo naključno število, porazdeljeno po Gaussovi porazdelitvi z varianco $\sigma_i$. S tem smo v algoritem uvedli še toliko dodatnih parametrov, kolikršna je dolžina niza, njihove vrednosti pa evolucijske strategije določajo s samoadaptacijo. Genom razširimo še z nizom teh parametrov, ki pa jih ne določamo enako, kot vrednosti genoma. Izkaže se, da jih je najugodneje določiti tako, da je ena petina mutacij uspešna. Spreminajnje parametrov $\sigma_i$ je določeno z [6]

$$\sigma_i = \begin{cases} \frac{\sigma_i}{c} & p_i > \frac{1}{5}, \\ \sigma_i c & p_i < \frac{1}{5}, \\ \sigma_i & p_i = \frac{1}{5} \end{cases} \tag{8}$$

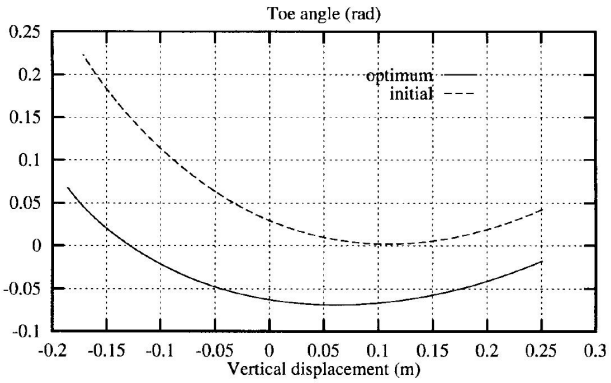
kjer je $p_i$ verjetnost, da je mutacija $i$-tega mesta uspešna, ter $c \in [0.817, 1)$. Zaradi robustnosti, ki jo evolucijske strategije pridobijo s samoadaptacijo, je njihova uporaba zelo razširjena.

Ker evolucijske strategije naravno optimiziriajo nize realnih števil, se pogosto uporabljajo za učenje nevronskih mrež ali pa optimizacijo kakšnih drugih algoritmov [6]. Zelo pomembne pa so v tehniki, kjer se jih uporablja za računalniško snovanje elementov.

Pri kompleksnih konstrukcijah lahko pride do problema optimizacije več spremenljivk, ki so med seboj nelinearno povezane. Takšna primera sta optimizacija avtomobilskega vzmetenja ali pa priklopa pri tramvaju Cityrunner [9]. Pri optimizaciji vzmetenja so s spreminjanjem dolžin sestavnih delov poskušali odpraviti eno izmed pomankljivosti uporabljenega vpetja. Zaradi vertikalnih odmikov vzmetenja namreč kolo pri vožnji naravnost nekoliko zavija. Na sliki 2 vidimo, da so odmike bistveno zmanjšali, oblike krivulje pa zaradi izbire vpetja ni mogoče spremeniti. Vpetje med vagoni tramvaja Cityrunner pa so poskušali optimizirati tako, da je horizontalni pospešek težišča vagona ob zavoju s hitrostjo 3.6 metra na sekundo v devetdesetstopinjski ovinek s polmerom 14 metrov minimalen. To so poskušali doseči s spreminajnjem koeficientov vzmeti in faktorjev dušenja. Končna odvisnost je prikazana na sliki 3.

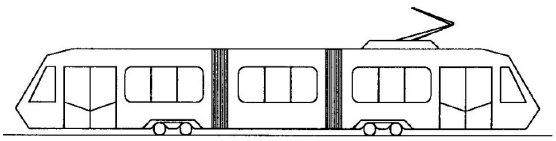



(a) Skica problema. Optimiziramo po vseh na sliki označenih spremenljivkah.

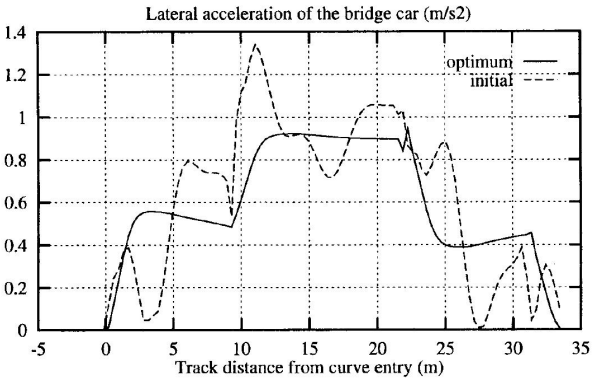



(b) Primerjava obnašanja optimiziranega in neoptimiziranega sistema.

Slika 2: Optimizacija avtomobilskega vzmetenja. [9]



(a) Skica tramvaja Cityrunner. Optimiziramo gibanje sredinskega vagona.



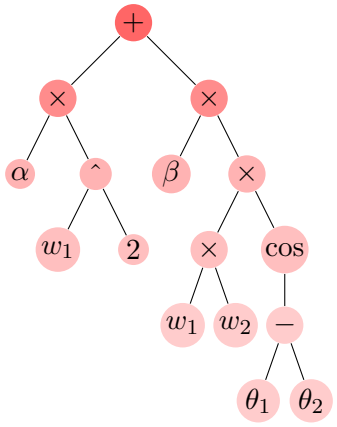

(b) Primerjava prečnega pospeška optimiziranega in neoptimiziranega sistema.

Slika 3: Optimizacija vpetja vagonov tramvaja Cityrunner. [9]

### 5.3 Genetsko programiranje

Evolucijski algoritem lahko uporabimo tudi za razvoj programov. Tej podveji evolucijskega računanja pravimo genetsko programiranje. Zelo zanimiv je v tem primeru način kodiranja posameznikov. Genom namreč predstavimo z drevesom, pri tem je vsako vozlišče v drevesu binarni operator, listi pa so argumenti. Na ta način lahko zapišemo vse bistvene sestavine programov; aritmetične in logične formule ter programsko kodo. Primeri predstavitve aitmetičnega izraza je podana na sliki 5.

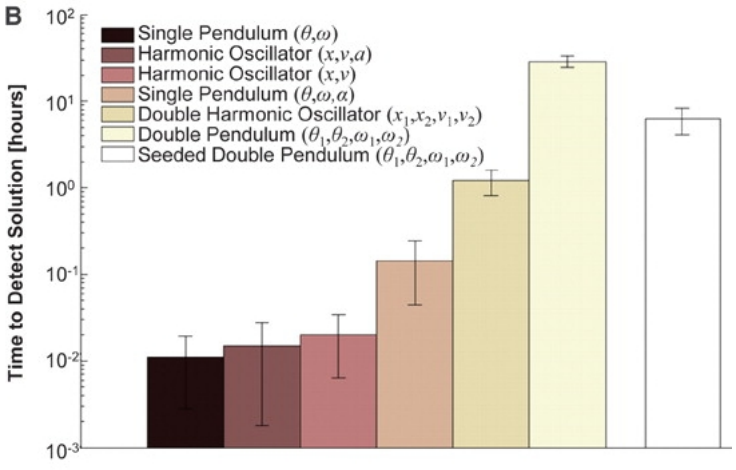
Križanje izvedemo s tako, da izberemo naključni poddrevesi v starševskih drevesih in ju med seboj zamenjamo, mutacijo pa tako, da naključno zgeneriramo drevo določene globine in z njim nadomestimo eno izmed vozlišč v drevesu, na katerem izvajamo mutacijo.




Slika 4: Predstavitev izraza $\alpha\omega_1^2 + \beta\omega_1\omega_2\cos(\theta_1\theta_2)$ z drevesom.

Ker tovrstni postopki že mejijo na umetno inteligenco, lahko takšne postopke uporabimo pri iskanju naravnih zakonitosti [10]. Avtorji v tem članku s pomočjo genetskega programiranja poiščejo izraze za konstante gibanja nekaterih enostavnih fizikalnih sistemov. Časi izračunov in obravnavani fizikalni sistemi so prikazani na sliki **??**. Uspešnost algoritma je sicer močno odvisna od zapletenosti fizikalnega sistema. Za matematično nihalo in harmoski oscilator je sistem uspel napovedati Lagrangevo funkcijo in enačbe gibanja, pri čemer je za matematično nihalo uspel rekonstruirati tudi enačbo krožnice, po kateri se nihalo giblje. Za bolj zapletene sisteme (dvojno nihalo oz. sklopljena harmonska oscilatorja), pa je določil le Lagrangevo in Hamiltonovo funkcijo. Zanimivo je tudi dejstvo, da ob preskopem naboru baznih funkcij algoritem sam uporabi Taylorjev razvoj ali kakšno drugo enakost, da lahko čim bolje opiše obravnavani sistem.

Genetsko programiranje se sicer uporablja tudi pri optimizaciji računalniških programov. Tako so z njim uspeli ponovno odkriti najhitrejši način za množenje matrik $3 \times 3$ [11], pri čemer so odkrili zelo dober približen postopek za množenje, ki pri izračunu uporablja namesto 23 le 22 množenj.




Slika 5: Čas izračuna konstant gibanja za različne fizikalne sisteme. [10]

## Literatura


[1] David A. Coley, *An Introduction to Genetic Algorithms for Scientists and Engineers*, (World Scientific, Singapore, 1999)

[2] Michael Affenzeller et al., *Genetic algorithms and genetic programming: modern concepts and practical applications*, (Chapmann & Hall, elektronska izdaja, 2009)

[3] A.E. Eiben in J.E. Smith *Introduction to Evolutionary Computing*, (Springer, elektronska izdaja, 2007)

[4] John H. Holland *Adaptation in natural and artificial systems*, (The University of Michigan Press, Michigan, 1975)

[5] A. Taranenko, Genetski algoritmi, *diplomsko delo*, Maribor 2001

[6] A. Dobnikar in B. Šter, *Mehko računanje za modeliranje, razpoznavanje in regresijo*, (Založba FE in FRI, Ljubljana, 2008)

[7] J. R. Morris, D. M. Deaven in K. M. Ho, *Phys. Rev. B*, **53**, (1996) R1740

[8] A. Slosar in R. Podgornik, *Europhys. Lett.*, **53** (4), 631, (2006)

[9] S. Datoussaid, O. Verlinden, C.Conti, *Multibody System Dynamics*, **8** 393, 2002

[10] M. Schmidt in H. Lipson, *Science*, **324** 81, 2009

[11] A. Joo, *IEEE Transactions on Evolutionary Computation*, **99**, 2012