Evolucijski algoritmi

Lukačić, Sven

Fakultet za matematiku, Sveučilište u Rijeci
Diplomski studij Diskretna matematika i primjene

Sven Lukačić

Evolucijski algoritmi

Rijeka, rujan 2022.

Fakultet za matematiku, Sveučilište u Rijeci
Diplomski studij Diskretna matematika i primjene

Sven Lukačić

Evolucijski algoritmi

Mentorica : dr. sc. Andrea Švob

Rijeka, rujan 2022.

1 Sažetak

Početak razvoja evolucijskih algoritama potječe još iz 1940−ih godina. Inspiraciju su dobili iz biologije i to upravo iz Darwinove teorije evolucije. Dva najbitnija svojstva koja su ujedno i stup evolucijskih algoritama su prirodna selekcija i fenotipske varijacije. Oni osiguravaju da bi se došlo do dobrog i povoljnog rješenja kroz više iteracija. Komponente evolucijskog algoritma su reprezentacija, evaluacijska funkcija, populacija, mehanizam za odabir roditelja, varijacijski operatori i mehanizam za odabir preživjelih. Obzirom na specifikaciju spomenutih komponenata razlikovat ćemo vrste evolucijskih algoritama. Zbog svoje fleksibilnosti, evolucijski algoritmi imaju široku primjenu i moguće ih je korisiti za mnoge probleme kao što su na primjer problem osam kraljica ili problem ruksaka. Evolucijske algoritme dijeli na tradicionalne i moderne. Tradicionalni su genetski algoritam (koji se često koristi kao sinonim za evolucijski algoritam), evolucijske strategije, evolucijsko programiranje i genetsko programiranje, dok u moderne evolucijske algoritme spadaju klasifikatorski sustavi, diferencijska evolucija i optimizacija roja čestica. Kako bi kreirali najbolji mogući evolucijski algoritam za dani problem, potrebno je podesiti i kontrolirati parametre odgovarajućem problemu. Svaki problem je unikatan, pa su tako i unikatni parametri evolucijskog algoritma za taj problem kao što su veličina koraka mutacije, stopa mutacije, koeficijent kazne itd.

Sadržaj

Popis slika

Popis tablica

2 Uvod

Evolucijsko računarstvo je istraživačko područje unutar računarstva. Kao što se može shvatiti po imenu, ta grana računarstva inspiraciju uzima iz procesa evolucije. Ne treba začuditi da je evolucija poslužila kao inspiracija za jednu od grana računarstva jer se savršenstvo evolucije vidi u raznolikosti svih jedinki koje čine svijet i sve su sposobne za preživljavanje.

2.1 Povijest

Ideja kako primijeniti Darwinove principe potječe još od 1940-ih. Alan Turing je 1948. predložio "genetsko ili evolucijsko pretraživanje" te je tijekom 1960-ih došlo je do tri implementacije, u SAD-u su Fogel, Owens i Walsh predstavili evolucijsko programiranje, dok je Holland svoju metodu zvao genetski algoritam. U Njemačkoj su u međuvremenu Rechendberg i Schwefel smislili evolucijske strategije. Idućih 15 godina ta područja su se razvijala zasebno jedan od drugoga, ali od ranih 1990-ih ona su viđena kao različite vrste jedne tehnologije koju znamo kao evolucijsko računarstvo. Također, u ranim 1990-ima četvrta struja se razvila, a to je genetsko programiranje. Današnja terminologija cijelo to područje naziva evolucijskim računarstvom, a njene algoritme naziva evolucijskim algoritmima.

2.2 Inspiracija iz biologije

Darwinova teorija evolucije daje nam objašnjenje biološke raznolikosti i mehanizama iza nje. U svemu tome, centralnu ulogu igra prirodna selekcija. Ako imamo područje u kojem postoji ograničen broj jedinki i ako je osnovni instinkt jedinki reprodukcija, selekcija postaje neizbježna ako veličina populacije ne počne eksponencijalno rasti. Prirodna selekcija favorizira one koji su najbolje prilagođeni uvjetima. Ovaj fenomen je inače poznat kao preživljavanje najsposobnijih. Selekcija obzirom na konkurenciju je jedno od dva stupa evolucijskog progresa. Drugi stup je fenotipske varijacije. Fenotipske varijacije su psihološka i fizička obilježja jedinke koja direktno utječu na njegov odgovor na okolinu. Svaka jedinka (individualac) predstavlja jedinstvenu kombinaciju fenotipskih varijacija.

Ako se ta kombinacija evaluira dobrom, onda individualac ima veće šanse da kreira potomke, inače je individualac osuđen na umiranje bez potomaka. Ako su te fenotipske varijacije nasljedne, a ne moraju sve biti, tada one mogu biti proslijeđene preko potomaka tog individualca. Darwin je tvrdio da se male, nasumične varijacije, odnosno mutacije, u fenotipskim varijacijama mogu dogoditi tijekom reprodukcije. Tijekom tih varijacija se stvara nova kombinacija fenotipa koji će biti evaluirani. Oni najbolji preživljavaju te nastavljaju evoluciju.

3 Evolucijski algoritam

U ovome poglavlju upoznat ćemo se sa pojmom evolucijskog algoritma i njegovih komponenti.

Postoje različite vrste evolucijskog algoritma. Zajednička ideja svih evolucijskih algoritama je da prema danoj populaciji u nekoj okolini sa ograničenim resursima, natjecanje za tim resursima dovodi do prirodne selekcije, odnosno preživljavanja najjačih. Zbog toga dolazi do porasta kvalitete dane populacije. To je gruba ideja koja stoji iza svih evolucijskih algoritama, ali nju ćemo proširiti, odnosno preciznije ćemo opisati algoritam. Imamo funkciju kvalitete (eng. quality function) te nasumice odaberemo skup kandidata koji će predstavljati našu populaciju. Taj skup kandidata će biti domena funkcije kvalitete. Tada možemo primijeniti funkciju kvalitete da bi vidjeli koliko je trenutna populacija kvalitetna. Prema tim vrijednostima, odaberemo određene kandidate koje ćemo koristiti za dobivanje iduće generacije. To dobijemo rekombinacijom i mutacijom. Rekombinacija je operator koji se primjenjuje na dva ili više kandidata (tzv. roditelji) te od nje dobijemo više novih kandidata (tzv. djeca). Mutacija se primjenjuje na jednog kandidata i od nje dobijemo jednog novog kandidata. Tim operacijama na roditelje dođemo do kreiranja novih kandidata (tzv. potomstvo). Tada ti novi kandidati budu evaluirani funkcijom kvalitete i tada se oni natječu sa starijim kandidatima za svoje mjesto u novoj generaciji. Taj proces se ponavlja sve dok nije postignuta dovoljna kvaliteta, odnosno nismo došli do rješenja ili sve dok maksimalan broj iteracija koji je zadan nije završen.

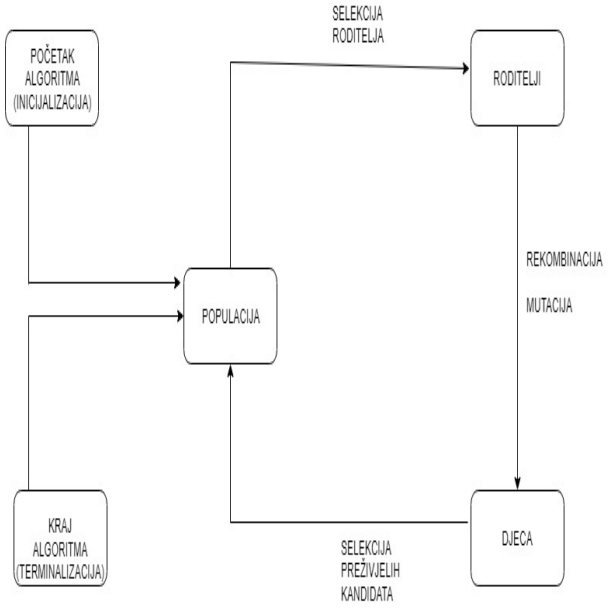
Generalna shema evolucijskog algoritma je sljedeća:

```
POČETAK
    ODABEREMO nasumično populaciju
    EVALUIRAMO svakog kandidata
    PONOVI SVE DOK (UVJET je zadovoljen)
        1. IZABERI roditelja
        2. REKOMBINIRAJ parove roditelja
        3. MUTIRAJ dobiveno potomstvo
```

```
        4. EVALUIRAJ nove kandidate
        5. IZABERI kandidate za novu generaciju
    KRAJ PETLJE
ZAVRŠETAK
```

Evolucijski algoritam spada u familiju "generiraj i testiraj metoda", a razlozi su:

- procesuiraju cijeli skup kandidata u isto vrijeme,
- koriste rekombinaciju, tj. miješaju obilježja dva ili više kandidata da bi dobili novog,
- stohastični su.



Slika 1: Dijagram toka evolucijskog algoritma

Dvije stvari tvore bazu evolucijskog sustava, a to su:

- operatori varijacije, tj. rekombinacija i mutacija, koji tvore nužnu raznolikost unutar populacije,
- selekcija koja služi da bi se poboljšala prosječna kvaliteta rješenja unutar populacije.

Kombinirajući selekciju i varijaciju dolazimo do poboljšanja kvalitete u svakoj idućoj populaciji stoga možemo reći da evolucija optimizira funkciju kvalitete tako da se približava optimalnoj vrijednosti sve više i više s vremenom.

3.1 Komponente evolucijskog algoritma

U ovom poglavlju ćemo detaljnije objasniti komponente evolucijskog algoritma. To su redom:

- reprezentacija (definicija individualaca),
- evaluacijska funkcija, odnosno funkcija kvalitete
- populacija,
- mehanizam za odabir roditelja,
- varijacijski operatori, tj. rekombinacija i mutacija,
- mehanizam za odabir preživjelih (zamjena).

Kako bi kreirali kompletan algoritam, potrebno je specificirati svaku komponentu i definirati proces inicijalizacije. Ako želimo da algoritam stane u nekom trenutku, moramo dati i uvjet terminacije. Naravno, ne mora svaki algoritam imati uvjet terminacije, ali za naše potrebe ćemo ga kreirati.

3.1.1 Reprezentacija

Prvi korak u povezivanju evolucijskog algoritma sa stvarnim svijetom je da uspostavimo vezu između konteksta originalnog problema i prostora u kojem će se evolucija dogoditi, odnosno prostora u kojem ćemo rješavati naš problem. To često podrazumijeva da ćemo pojednostaviti ili izbaciti neke aspekte stvarnog života.
Prvi korak je da odlučimo koliko mogućih rješenja treba biti određeno i spremljeno. Objekti koji tvore moguće rješenje zovemo fenotipi, dok jedinke unutar evolucijskog algoritma zovemo genotip. Prvi korak u dizajniranju evolucijskog algoritma zovemo reprezentacija, a to je specifikacija preslikavanja sa skupa fenotipa na skup genotipa koji ih predstavljaju. Na primjer, dan je optimizacijski problem gdje su moguća rješenja realni brojevi, dan skup realnih brojeva čini skup fenotipa. U ovom slučaju, možemo se odlučiti da ih prikažemo u binarnom sustavu, pa recimo vrijednost 18 bi bila fenotip, dok bi 10010

bio genotip. Važno je naglasiti da skup fenotipa može biti skroz drugačiji od skupa genotipa i da cijela evolucijska potraga se događa u skupu genotipa. Dobar fenotip, tj. rješenje, dobije se dekodiranjem najboljeg genotipa poslije terminacije. Inverz operacije kodiranja fenotipa u genotip je dekodiranje i potrebno je da je operacija kodiranja invertibilna.

3.1.2 Evaluacijska funkcija (funkcija kvalitete)

Uloga evaluacijske funkcije je da prezentira kriterije kojima se populacija mora prilagođavati. To je osnova selekcije i ona dovodi do poboljšanja populacije, odnosno ona govori što je zapravo točno poboljšanje u evolucijskom algoritmu. To je funkcija koja svakom genotipu daje određenu vrijednost koja prikazuje njegovu kvalitetu. Ona se sastoji od inverza reprezentacije kako bi dobili odgovarajući fenotip i tada za taj fenotip daje određenu vrijednost. Na primjer, neka je zadatak naći realni broj x koji maksimizira x^2, kvaliteta genotipa 10010 je definirana tako da prvo dekodiramo taj genotip, vidimo da je njegov fenotip 18 i tom fenotipu dajemo vrijednost kvalitete $18^2 = 324$.
U evolucijskom računarstvu, evaluacijska funkcija se često naziva funkcija kvalitete. Zbog toga bi lako moglo doći do kontradikcije ako je originalni problem minimizacija, a pojam funkcije kvalitete obično se povezuje sa maksimizacijom, ali matematički je trivijalno promijeniti problem minimizacije u problem maksimizacije i obrnuto.

3.1.3 Populacija

Populacija je multiset genoma. Jedinke su statični objekti koji se ne mijenjaju ili prilagođavaju, nego to čini populacija. Ako imamo reprezentaciju, definiranje populacije je jednostavno, to možemo napraviti tako da specificiramo koliko individualaca u njoj ima, odnosno samo odredimo veličinu populacije. U određenim sofisticiranijim evolucijskim algoritmima populacija ima dodatna svojstva kao što su naprimjer mjera udaljenosti ili relacija susjednosti. To je povezano sa time da prava populacija evoluira unutar konteksta geografske lokacije individualaca.
U skoro svim evolucijskim algoritmima je veličina populacije konstanta i ne mijenja se tijekom evolucijske potrage. To dovodi do toga da limitirani resursi koji su na raspolaganju populaciji stvaraju konkurenciju, odnosno potrebu za preživljavanjem. Raznolikost

populacije je broj različitih rješenja. Postojanje samo jedne vrijednosti kvalitete u populaciji ne implicira da samo jedan fenotip tu vrijednost može postići, pošto više fenotipa može postići istu kvalitetu. Također, postojanje samo jednog fenotipa ne znači nužno da ima samo jedan genotip. Ali, postojanje samo jednog genotipa znači da ima samo jedan fenotip i jedna vrijednost kvalitete.

3.1.4 Mehanizam za odabir roditelja

Uloga mehanizma za odabir roditelja je da odijeli dobre potencijalne kandidate i da se od njih dobije nova generacija. Zajedno sa mehanizmom za odabir preživjelih je odgovoran za poboljšanje kvalitete. U evolucijskom računarstvu odabir roditelja je obično probabilistički. Stoga individualci koji su kvalitetniji imaju veće šanse da bi postali roditelji i tako doveli do poboljšanja populacije. Svejedno i manje kvalitetni individualci imaju šanse da budu odabrani za roditelje jer inače bi potraga postala previše pohlepna i populacija bi mogla zaglaviti u lokalnom optimumu.

3.1.5 Varijacijski operatori

Varijacijski operatori su mutacija i rekombinacija. Njihova uloga je da kreiraju nove jedinke od starih. U već spomenutom skupu fenotipa to dovodi do generiranja novih kandidata za rješenje.
Mutacija je unaran varijacijski operator. Primjenjuje se na jedan genotip i dovodi do modificiranog mutanta, djeteta ili potomka. Mutacija je stohastičan operator, odnosno njegov output, dijete, ovisi o seriji slučajnih odabira. Mutacija bi trebala biti slučajna, nepristrana promjena. Ali to ne mora uvijek biti slučaj. U genetskim algoritmima je tradicionalno korištena kao pozadinski operator koji dovodi "novu krv" u skup gena, a u evolucijskom programiranju je to jedini varijacijski operator, te je on jedini zadužen za stvaranje nove generacije jedinki, a u genetskom programiranju se mutacije uopće ne koriste. Varijacijski operatori daju prostoru pretraživanja njegovu topološku strukturu gdje je generiranje novog djeteta jednako tome da dođemo do nove točke u tom prostoru. Stoga, mutacija osigurava da je prostor povezan što će omogućiti evolucijskom algoritmu da otkrije globalni optimum za zadani problem.

Rekombinacija ili križanje je binaran (može biti i n-aran) varijacijski operator. Takav operator spaja svojstva dva roditelja genotipa u jedan ili dva potomka genotipa. Kao i mutacija, rekombinacija je stohastički operator, što znači da odabir koji će dijelovi svakog roditelja biti u rekombinaciji i kako će to biti napravljeno, ovisi o slučajnosti. Kao i kod mutacije, uloga rekombinacije je različita u ostalim dijelovima evolucijskog računarstva. U genetskom programiranju je to često jedini varijacijski operator, dok je u genetskim algoritmima to glavni operator pretrage, a u evolucijskom programiranju se rekombinacija uopće ne koristi. Rekombinacija sa više od dva roditelja je matematički moguća i lako izvediva, ali nema nikakvu biološku podlogu. Iz tog razloga se baš i ne koriste. Princip iza rekombinacije je jednostavan. Uparivanjem dvije jedinke koje imaju različita, ali poželjna svojstva, možemo dobiti potomstvo koje sadrži oba ta svojstva. Rekombinacijski operatori u evolucijskim algoritmima se inače primjenjuju probabilistički sa naravno ne-nul šansom da se oni ne dogode.

Varijacijski operatori su ovisni o reprezentaciji, stoga ćemo za različite reprezentacije definirati različite varijacijske operatore. Na primjer ako je genotip bit-stringovi, onda će invertiranje bita biti korišteno kao mutacijski operatori. Ali, ako moguća rješenja prikažemo u strukturi stabla, onda će još jedan mutacijski operator biti potreban.

3.1.6 Mehanizam za odabir preživjelih (Selekcija preživjelih)

Kao i kod mehanizma za odabir roditelja, uloga selekcije preživjelih ili selekcije okoline je da odjeli individualce lošije kvalitete od onih bolje kvalitete, ali se koristi tek nakon što smo kreirali potomstvo od odabranih roditelja. Kao što je već spomenuto, veličina populacije je konstantna, pa se se mora provesti neki odabir da se vidi tko će bit član iduće generacije. Tu odluku često donosimo obzirom na vrijednost kvalitete gdje su naravno u prednosti oni sa većom vrijednosti kvalitete iako je koncept godina dosta često korišten. Selekcija preživjelih ima deterministički pristup za razliku od odabira roditelja. Selekcija preživjelih je zapravo zamjenska strategija.

3.1.7 Inicijalizacija

Inicijalizacija je iznimno jednostavna u većini evolucijskih algoritama. Prva populacija je generirana slučajnim odabirom individualaca.

3.1.8 Uvjet terminalizacije

Razlikujemo dva slučaja uvjeta terminalizacije. Ako problem ima optimalnu vrijednost kvalitete, tada će uvjet terminalizacije biti pronalazak jedinke sa tom vrijednosti. Ako znamo da naš model sadrži neke nužno potrebno pojednostavljivanje, možemo prihvatiti i rješenje sa određenom tolerancijom $\epsilon > 0$. Ali, evolucijski algoritmi su stohastični i većinom nemamo garanciju da ćemo postići takav optimum, pa taj uvjet možda nikad neće biti ispunjen i algoritam neće nikad prestati raditi. Iz tog razloga su iduće opcije često korištene:

1. maksimalno koliko CPU vremenski može provoditi algoritam,
2. ukupan broj evaluacija kvalitete,
3. poboljšanje kvalitete ostane ispod neke određene vrijednosti za neki period,
4. raznolikost padne ispod nekog broja.

4 Primjeri i primjena

Sada kada smo detaljnije opisali sve komponente evolucijskog algoritma, pokazat ćemo na primjerima njihovu primjenu.

4.1 Evolucijski krug

Započet ćemo sa jednostavnim problemom, a to je maksimiziranje vrijednosti x^3 za brojeve u rasponu od 0 do 31. Opisat ćemo jedan evolucijski krug tog problema. Prvo moramo odlučiti kako će nam komponente evolucijskog algoritma izgledati, znači zanima nas reprezentacija, mehanizam za odabir roditelja, rekombinacija, mutacija i selekcija preživjelih.

Za reprezentaciju koristit ćemo jednostavno pet-bitno binarno kodiranje koje će preslikati brojeve, odnosno fenotipe, u bit-stringove, odnosno genotipe. Za odabir roditelja ćemo se mjeriti prema kvaliteti gdje će vjerojatnost(p_i) da će individualac i u populaciji P biti izabran za roditelja biti dana sa

$$p_i = \frac{f(i)}{\sum_{j \in P} f(j)}.$$

Cijelu populaciju ćemo zamijeniti od potomstva koje dobijemo iz nje. To znači da ćemo kreirati jednako mnogo potomaka koliko je i članova populacije. Mutacija će biti takva da ćemo generirati slučajan broj 0 ili 1 na svakoj poziciji i ako se vrijednost poboljšala, onda ćemo ostaviti tu mutaciju na toj poziciji. Rekombinacija će biti obavljena na dva roditelja i producirat će dvoje djece tako da nasumice odabere točku križanja unutar stringa i zamijeni bitove roditelja unutar tog mjesta križanja.

Broj Stringa	Inicijalna Populacija	x	Vrijednost kvalitete $f(x) = x^3$	Vjerojatnost	Očekivani broj pojavljivanja	Broj pojavljivanja
1	0 1 1 1 1	15	3375	0.09	0.55	1
2	1 1 0 0 0	24	13824	0.59	2.25	2
3	0 1 0 0 0	8	512	0.03	0.08	0
4	1 0 0 1 1	19	6859	0.29	1.12	1
Zbroj			24570	1.00	4.00	4
Prosjek			6142.5	0.25	1.00	1
Max			13824	0.59	2.25	2

Tablica 1: Inicijalizacija, evalucija i odabir roditelja

U tablici 1 je prikazana nasumična inicijalizacija populacije od četiri genotipa, njima pridužujućih fenotipa i njihovih vrijednosti kvalitete. Evolucijski krug počinje tako da izaberemo roditelje koje ćemo koristiti za dobivanje iduće generacije. Šesti stupac u tablici pokazuje očekivani broj kopija svakog individualca poslije selekcije roditelja. Ti brojevi nisu cijeli brojevi, nego prikazuju distribucije vjerojatnosti i skup za reprodukciju se kreira nasumičnim odabirom iz tog uzorka te distribucije. Zadnji stupac nam govori koliko će stvarni broj kopija biti skupu za reprodukciju. Odabrani individualci su spareni nasumice i za svaki par je nasumice odabrana točka križanja.

Broj Stringa	Skup za reprodukciju	Točka križanja	Potomstvo nakon križanja	x	$f(x) = x^3$
1	0 1 1 1 | 1	4	0 1 1 1 0	14	2744
2	1 1 0 0 | 0	4	1 1 0 0 1	25	15625
3	1 1 | 0 0 0	2	1 1 0 1 1	27	19683
4	1 0 | 0 1 1	2	1 0 0 0 0	16	4096
Zbroj					42148
Prosjek					10537
Max					19683

Tablica 2: Križanje i evaluacija potomstva

Broj Stringa	Potomstvo nakon križanja	Potomstvo nakon mutacije	x	$f(x) = x^3$
1	0 1 1 1 0	1 1 1 1 0	31	29791
2	1 1 0 0 1	1 1 0 0 1	25	15625
3	1 1 0 1 1	1 1 0 1 1	27	19683
4	1 0 0 0 0	1 0 1 0 0	18	5832
Zbroj				70931
Prosjek				17732.75
Max				29791

Tablica 3: Mutacija i evaluacija potomstva

U Tablici 2 su prikazani rezultati križanja na danom skupu za reprodukciju sa njihovim vrijednostima kvalitete. U Tablici 3 se vide rezultati mutacija. U ovom slučaju je došlo do pozitivnih promjena u kvaliteti, ali kako bi radili što više operacija, vidjeli bi u idućim generacijama došlo do toga da bi mutacije dale prosječno iste rezultate jer bi dolazilo do jednakog broja pojavljivanja 1-ice u stringovima. Vidimo u ovom primjeru da je prosječna kvaliteta populacije narasla sa 6142.5 na 17732.75 te je najbolji pojedinac u populaciji imao vrijednost 13824, a 29791 nakon križanja i mutacije.

4.2 Problem osam kraljica

Problem N kraljica je poznat matematički problem koji se koristi u računalnim znanostima i često se zna koristiti kao praktični primjer za rješavanje problema nekom metodom. Ideja problema je kako smjestiti osam kraljica na ploču dimenzija osam puta osam tako da se one međusobno ne napadaju, odnosno da jedna drugoj ne mogu stajati u putanji kretanja. Kako se ovaj primjer problema često koristi, logično je da ima puno različitih pristupa ovom problemu. Obično započnu tako da se jedna kraljica stavi na neko mjesto i zatim nakon što smo stavili n kraljica, pokušamo staviti $(n + 1)$-u kraljicu na neku dopustivu poziciju gdje nova kraljica ne napada niti jednu drugu kraljicu. Obično se koristi neko pretraživanje unatrag te ako nema niti jedne dopustive pozicije za $(n+1)$-u kraljicu, onda pomičemo n-tu kraljicu na neku drugu poziciju. Evolucijski pristup ovom problemu

je podosta drugačiji.

Naša potencijalna rješenja su kompletna, a ne parcijalna, konfiguracija šahovske ploče, što znači da specificiraju pozicije svih osam kraljica. Fenotipski prostor P je skup svih takvih konfiguracija. Očito da većina elemenata skupa P nisu moguća jer ne ispunjavaju uvjet da se sve kraljice međusobno ne mogu napadati. Kvaliteta, $q(p)$, svakog fenotipa $p \in P$ jednostavno može biti izračunata brojem kraljica koje se međusobno mogu napasti. To znači da što je $q(p)$ manja, to je kvaliteta fenotipa bolja. Iz ovoga vidimo da za vrijednost $q(p) = 0$ dobijemo rješenje, što znači da će ovaj problem biti zapravo problem minimizacije. Iako još nismo definirali skup genotipa, možemo definirati funkciju kvalitete genotipa kao neki inverz od $q(p)$. Najjednostavnija opcija bi bila uzeti $\frac{1}{q(p)}$, ali dolazimo do problema jer ne možemo dijeliti sa nulom. Taj problem bi mogli zaobići tako da kada bi se taj slučaj dogodio, jednostavno bi rekli da smo došli do rješenja. Možemo to zaobići i na drugi način. Možemo dodati proizvoljno malu vrijednost ϵ, pa dobijemo $\frac{1}{q(p)+\epsilon}$. Druge opcije još su da koristimo $-q(p)$ ili $M - q(p)$, gdje je M dovoljno veliki realni broj takav da sve vrijednosti koje funkcija kvalitete može postići budu pozitivne. Npr. $M \geq \max\{q(p) : p \in P\}$. Ova funkcija kvalitete nasljeđuje svojstvo od q da postoji znani optimum M.

Kako bi definirali evolucijski algoritam koji će pretraživati prostor P, moramo prvo definirati kako ćemo reprezentirati fenotipe iz prostora P. Očito rješenje bi bilo korisiti matrice, ali postoji i jedno zgodnije rješenje. Genotip je permutacija brojeva $1, ..., 8$ i dani $g = (i_1, ..., i_8)$ označava jedinstvenu konfiguraciju ploče gdje se u n-tom stupcu nalazi točno jedna kraljica na i-tom mjestu. Znači da će permutacija $g = (1, ..., 8)$ označavati konfiguraciju ploče gdje će sve kraljice biti smještene na glavnoj dijagonali. Znači da je prostor genotipa G skup svih permutacija $1, ..., 8$ i definirali smo preslikavanje $F : G \to P$. Ovakva reprezentacija daje pretragu konfiguracija ploča gdje su horizontalni uvjet, odnosno da kraljice ne smiju biti u istom retku, i vertikalan uvjet, odnosno da kraljice ne smiju biti u istom stupcu, uvijek ispunjeni. Ostaje jedino minimizirati broj rješenja gdje je dijagonalan uvjet ispunjen.

Idući korak je da definiramo operatore varijacije, odnosno mutaciju i križanje, za našu reprezentaciju. Najbitnije svojstvo prikladnog operatora varijacije je da ne možemo "izaći"

iz prostora G, odnosno da je kodomena minimalno podskup skupa G. Potomci pemutacija također moraju biti permutacije. Za proces mutacije možemo koristiti operator koji nasumice odabire dvije pozicije unutar danog genotipa i zamjenjuje vrijednosti na tim pozicijama. Definirati dobro križanje za permutacije nije tako jednostavno.

```
1.  Nasumice odaberi točku križanja između 1, ... , 7
2.  Podijeli oba roditelja na dva segmenta u toj točki
3.  Kopiraj prvi segment od roditelja 1 u dijete 1 te isto
napravi za prvi segment roditelja 2 u dijete 2
4.  Skeniraj roditelja 2 slijeva na desno i popuni drugi
segment djeteta 1 sa vrijednostima roditelja 2, zanemareći
vrijednosti koje već ima
5.  Isto napravi za roditelja 1 i dijete 2
```

Ovaj algoritam nazivamo "cut and crossfill".Najbitnije svojstvo tih varijacijskih operatora je da mutacija stvara male neizravne promjene, dok križanjem dobijemo djecu koja nasljeđuju genetski materijal oba roditelja.

Idući korak u kreiranju evolucijskog algoritma je odlučiti kako će izgledati proces selekcije i proces obnavljanja populacije. U svakom evolucijskom ciklusu odabrat ćemo dva roditelja koji će dati dvoje djece i nova populacija veličine n će biti predstavljena od n najboljih pojedinaca do $n + 2$ pojedinaca koji u tom trenutku čine populaciju. Odabir roditelja će biti napravljen tako da ćemo nasumice odabrati pet pojedinaca iz populacije te ćemo od tih pet odabrati dvoje najboljih i njih ćemo uzeti za razmnožavanje. To će nam osigurati pristranost k roditeljima veće kvalitete te će brže dovesti do rješenja. Strategija koja je opisana spaja populaciju i potomstvo, rangira ih te na temelju kvalitete, riješi se dvoje najgorih.

Kako bi dobili cjelovitu specifikaciju našeg algoritma, možemo odlučiti popuniti incijalnu populaciju nasumice odabranim permutacijama i uvjet terminacije staviti kada nađemo rješenje ili kada se odvije 10000 procjena kvalitete populacije, što god od tog dvoje se dogodi prije. Nadalje možemo odlučiti koristiti veličinu populacije 100 i da se varijacijski operatori koriste u određenoj frekvenciji, npr. da uvijek provodimo križanje dvoje izabranih roditelja i da u 80% slučajeva koristimo mutaciju na potomstvu.

Kada skupimo sve što je opisano, dobijemo evolucijski algoritam koji je opisan u tablici 4.

Reprezentacija	Permutacije
Rekombinacija	"cut and crossfill" algoritam
Vjerojatnost rekombinacije	100%
Mutacija	Zamjena
Vjerojatnost mutacije	80%
Odabir roditelja	Najbolji dvoje od nasumičnih pet
Selekcija preživjelih	Zamjena najgorih
Veličina populacije	100
Broj potomaka	2
Inicijalizacija	Nasumice
Uvjet teminalizacije	Rješenje ili 10000 evaluacija kvalitete

Tablica 4: Opis evolucijskog algoritma za problem osam kraljica

4.3 Problem ruksaka

Problem ruksaka je generalizacija mnogih industrijskih problema. Imamo skup od n stvari gdje svaka ima neku svoju vrijednost v_i i svoju cijenu c_i. Zadatak je odabrati podskup stvari koji maximizira vrijednost dok je u istom trenutku cijena tog podskupa manja od neke određene vrijednosti C_{max}. Na primjer, kada pakiramo ruksak za neko putovanje, moramo izbalansirati važnost svake stvari u odnosu na njene veličine i težine.

Prirodno se nameće da reprezentacija kandidata za rješenje bude binarni string duljine n, gdje 1 na određenoj poziciji predstavlja stvar koja je odabrana, a 0 predstavlja stvar koja nije odabrana. Skup korespondirajućih genotipa G je skup svih takvih stringova veličine 2^n. Sada moramo definirati preslikavanje sa skupa genotipa u skup fenotipa. Prva reprezentacija, u smislu preslikavanja, koju ćemo uzeti je da ona uzima skup fenotipa P i skupa genotipa G kao identične. Kvaliteta rješenja q, koji je reprezentiran genotipom g, je određen tako da sumira vrijednosti stvari koje su odabrane, tj. $q(p) = \sum_{i=1}^{n} v_i * g_i$. Ali, ovakva jednostavna reprezentacija preslikavanja dovodi do

određenih problema. Korištenje takvog preslikavanja može doći do nevažećeg rješenja koje ima cijenu veću od dopuštene cijene, tj. $\sum_{i=1}^{n} v_i * g_i > C_{max}$. Druga reprezentacija preslikavanja koja će riješiti ovaj problem je da uvedemo funkciju dekodiranja. Ona funkcionira na način da kad kreiramo rješenje čitamo string slijeva na desno i bilježimo cijenu svih odabranih stvari. Kad naiđemo na jedinicu u stringu, prvo provjerimo da li će ta odabrana stvar dovesti do toga da je cijena veća od dopuštene. Ovakvo preslikavanje osigurava da svi binarni stringovi prezentiraju valjano rješenje sa jedinstvenom vrijednosti kvalitete.

Prikladni operator rekombinacije je takozvani "one-point" križanje gdje uzmemo dva roditelja i nasumično odaberemo točku u njima. Dva potomka dobijemo tako da zamijenimo repove roditelja u toj točki. Taj operator rekombinacije ćemo primijeniti sa 70% vjerojatnošću, tj. za svaki par roditelja postoji 70% šansa da će kreirati dva nova potomka križanjem i 30% posto šanse da će ta djeca biti samo kopije svojih roditelja. Za mutaciju koristit ćemo "bit-flipping" što znači da ćemo u svakoj poziciji zamijeniti vrijednosti sa malom vjerojatnošću $p_m \in [0, 1\rangle$.

U ovom slučaju kreirat ćemo jednak broj potomaka kao kod inicijalne populacije. Kao što je gore opisano, kreirat ćemo dva potomka od dva roditelja koja ćemo upariti nasumice. Za odabir roditelja koristit ćemo turnir gdje svaki put odabiremo dva člana populacije nasumice i onaj sa većom vrijednosti $q(p)$ pobjeđuje te postaje roditelj. Način selekcije preživjelih će biti da u svakoj iteraciji odbacujemo cijelu populaciju i zamijenjujemo njihovim potomstvom.

Na kraju moramo razmisliti o incijalizaciji i terminalizaciji. Inicijalizacija će biti nasumični odabir nula i jedinica na svakoj poziciji naše incijalne populacije. Terminalizaciju bi inače postigli tako da imamo neku graničnu vrijednost za cijenu, ali u ovom slučaju nju nemamo, pa ćemo vrtiti algoritam sve dok se kvaliteta najboljeg člana populacije nije poboljšala u već 25 generacija.

Već smo defnirali vjerojatnost križanja, a stopa mutacije će biti $p_m = \frac{1}{n}$, dok će populacija sa kojom ćemo raditi biti 500. Cijeli evolucijski algoritam za ovaj problem je sumiran u tablici 5.

Reprezentacija	Binarni string dužine n
Rekombinacija	"one-point" križanje
Vjerojatnost rekombinacije	70%
Mutacija	Zamjena svake vrijednosti sa nezavisnom vjerojatnošću p_m
Vjerojatnost mutacije p_m	$\frac{1}{n}$
Odabir roditelja	Najboljih dvoje od nasumično odabranih
Selekcija preživjelih	Generacijski
Veličina populacije	500
Broj potomaka	500
Inicijalizacija	Nasumice
Uvjet teminalizacije	Nema pobojšanja u zadnjih 500 generacija

Tablica 5: Opis evolucijskog algoritma za problem ruksaka

5 Razne vrste evolucijskih algoritama

Postoje različite vrste evolucijskih algoritama koje dijelimo na tradicionalne i modernije te ćemo ih opisati u ovome poglavlju.

5.1 Genetski algoritam

Genetski algoritam je najpoznatiji tip evolucijskog algoritma, pa se često uzima i kao sinonim za evolucijski algoritam. Osmislio ga je Holland kao alat u učenju adaptivnog ponašanja. Genetski algoritmi su ponajviše smatrani kao optimizacijska metoda. Najpoznatija vrsta i ona na koju se najčešće referira kada se priča o genetskim algoritmima je jednostavan genetski algoritam(skraćenica: JGA). On ima binarnu reprezentaciju, malu vjerojatnost mutacije, proporcionalnu selekciju kvalitete i naglasak na genetski inspiriranoj rekombinaciji koja služi da bismo dobili nova potencijalna rješenja.
Proporcionalna selekcija kvalitete ili ruletna selekcija je genetski operator koji se uvelike koristi u genetskim algoritmima. On radi na način da funkcija kvalitete dodijeli vrijednost kvalitete f_i svakom mogućem rješenju, odnosno kromosomu. Tada dobijemo vjerojatnost kojom će neki kromosom biti izabran, $p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$. Inspiracija za ovaj postupak je dobivena iz kotača za rulet u kasinu.
Genetski algoritmi tradicionalno imaju fiksiran tijek rada, tj. danu populaciju od μ individualaca razmješamo te kreiramo nasumične parove i križamo ih sa vjerojatnosti p_i i novi potomci automatski zamjenjuju roditelje. Tada ta novodobivena populacija prolazi kroz mutaciju jedan po jedan gdje je svaki od l bitova modificiran mutacijom sa nezavisnom vjerojatnošću p_m. Skicu svega toga možemo vidjeti u tablici 6.

Reprezentacija	Binarni string dužine n
Rekombinacija	"one-point" križanje
Mutacija	Zamjena bitova
Odabir roditelja	Proporcionalnost kvalitete
Selekcija preživjelih	Generacijski

Tablica 6: Skica jednostavnog genetskog algoritma

U ranim godinama razvoja ovog područja, značajna pozornost se pridavala tome da se odrede prikladne vrijednosti za parametre genetskog algoritma kao što su veličina populacije i vjerojatnosti za križanje i mutaciju. Preporuke za stopu mutacije su bile između $\frac{1}{l}$ i $\frac{1}{\mu}$, vjerojatnost križanja između 0.6 i 0.8, i populacijska veličina se izražavala u pedesetinama ili malim stoticama, ali te su vrijednosti donekle bile povezane sa snagom kompjutera u 80-ima i 90-ima.

Unatoč svojoj jednostavnosti, JGA je jako raširen i korišten, i to ne samo u svrhe učenja i kao osnova novih algoritama, nego i za relativno jednostavne probleme u kojima je binarna reprezentacija prikladna.

5.2 Evolucijske strategije

Evolucijske strategije su osmislili Rechenberg i Schwefel ranih 1960-ih. Najranije evolucijske strategije bili su jednostavni dvočlani algoritmi koji su se označavali $(1 + 1)$ te su oni radili u vektorskim prostorima. Potomstvo je generirano dodavanjem nasumičnog broja neovisnog od roditelj-vektora svakog elementa i prihvaća ga ako je veće kvalitete. Alternativna verzija koja se označava $(1, 1)$ uvijek zamjenjuje roditelja sa potomkom, tako zaboravivši prijašnja rješenja po definiciji. Nasumični brojevi dolaze iz Gaussove distribucije sa aritmetičkom vrijednosti 0 te standardnom devijacijom σ gdje je σ veličina koraka mutacije. Jedna od ključnih prekretnica u razvoju evolucijskih strategija je da se predložio jednostavan mehanizam za prilagodbu veličine koraka, a to je poznato pravilo "$\frac{1}{5}$ stopa uspjeha" koje je osmislio Rechenberg. U 70-ima je osmišljen koncept multičlanske evolucijske strategije u kojoj se nalazi μ individualaca u populaciji, a u svakom ciklusu se stvara λ potomaka. Obzirom na to, takva evolucijska strategija se označava sa (μ, λ). To je dovelo do povećanja vjerojatnosti više sofisticiranih formi kontrole koraka i do razvoja jako korisnog svojstva u evolucijskom računarstvu, a to je samoadaptacija parametara strategije. Samoadaptivnosti znači da neki parametri evolucijskog algoritma variraju na specifičan način. Parametri su uključeni u kromosome i koevoluiraju sa rješenjima. To znači da evolucijske strategije rade zajedno sa produženim kromosomima (x, p) gdje je $x \in \mathbb{R}$ vektor iz domene funkcija cilja koje moramo optimizirati, dok p predstavlja parametre algoritma. Moderne evolucijske strategije uvijek samoadaptiraju korak mutacije.

Rekombinacija u evolucijskoj strategiji uključuje dva roditelja koji kreiraju jedno dijete. Da bi dobili λ potomaka, potrebno je napraviti λ križanja. Dvije se vrste rekombinacije razlikuju obzirom na rekombinaciju roditeljskih alela. Aleli, alelni geni, su naziv za različite oblike gena koja određuju istu osobinu. Korištenjem diskretne (lokalne) rekombinacije jedan od alela roditelja je nasumice izabran sa jednakom vjerojatnosti da svi budu odabrani. U srednjoj rekombinaciji uzimamo srednju vrijednost alela roditelja. Produžetak te metode je da uzmemo više od dvije rekombinacije jer su dva roditelja nasumice izabrani. Multiroditeljsku verziju zovemo globalna rekombinacija. Evolucijske strategije najčešće koriste globalnu rekombinaciju. Skicu svega ovo vidimo u tablici 7.

Reprezentacija	Realni vektori
Rekombinacija	Diskretna ili posredna
Mutacija	Gaussian perturbacija
Odabir roditelja	Uniformno nasumično
Selekcija preživjelih	Deterministički elitistička zamjena
Specijalnost	Samodapatacija koraka mutacije

Tablica 7: Skica evolucijske strategije

Stopa selektivnosti u evolucijskoj strategiji je vrlo visoka jer obično je λ puno veći od μ. Tradicionalno se uzima omjer $1:7$, ali u zadnje vrijeme se dosta koristi i omjer $1:4$. Vrijeme preuzimanja (τ^*) danog mehanizma selekcije je definiran brojem generacija koje je potrebno da bi se dobila populacija sa najboljim individualcima.

$$\tau^* = \frac{ln\lambda}{ln\frac{\lambda}{\mu}}.$$

Na primjer, za evolucijsku strategiju kojoj je $\mu = 15$ i $\lambda = 100$, imamo $\tau^* \approx 2$. Dok recimo kod genetskog algoritma sa $\mu = \lambda = 100$ imamo $\tau^* = \lambda ln\lambda = 460$ što znači da je evolucijska strategija agresivniji optimizator od jednostavnog genetskog algoritma.

5.3 Evolucijsko programiranje

Evolucijsko programiranje je razvio Fogel 1960-ih kako bi simulirao evoluciju kao proces učenja sa ciljem generiranja umjetne inteligencije. Na inteligenciju se gledalo kao na spo-

sobnost sustava da prilagodi svoje ponašanje kako bi postigao određeni cilj u različitim okolinama. Adaptivno ponašanje je ključno svojstvo u tome i mogućnost da se predvidi okolina je nužno potrebna.

Današnje evolucijsko programiranje često koristi realne vrijednosti reprezentacije, pa je stoga dosta slično evolucijskoj strategiji. Glavna razlika je u biološkoj inspiraciji. U evolucijskom programiranju na svakog individualca se gleda kao na posebnu jedinku, pa nema rekombinacije. Također su i mehanizmi selekcije bitno drugačiji. U evolucijskoj strategiji roditelje biramo stohastično, zatim deterministički biramo μ najboljih pojedinaca od njih $\mu + \lambda$, gdje je λ broj potomaka koliko smo dobili. Dok naprotiv, u evolucijskom programiranju svaki roditelj generira točno jednog potomka ($\lambda = \mu$), ali su onda ti roditelji i potomci stavljeni u turnir za preživljavanje.

Današnje područje evolucijskog programiranja je preuzelo pragmatičan pristup gdje izbor reprezentacije, stoga i mutacije, ponajprije bude strogo prilagođen problemu. Stoga će skica evolucijskog programiranja više predstavljati neku okvirnu smjernicu, nego nužno standardnu verziju algoritma.

Reprezentacija	Realni vektori
Rekombinacija	Ne postoji
Mutacija	Gaussian perturbacija
Odabir roditelja	Deterministička gdje svaki roditelj kreira jednog potomka mutacijom
Selekcija preživjelih	Vjerojatnosno (preko turnira)
Specijalnost	Samodapatacija koraka mutacije

Tablica 8: Skica evolucijskog programiranja

5.4 Genetsko programiranje

Genetsko programiranje je relativno novi član skupa evolucijskih algoritama. Razlika od ostalih evolucijskih algoritama je u području primjene i u tome da se kod genetskog programiranja koriste stabla za reprezentaciju kromosoma. Do sad obrađene vrste evolucijskih algoritama se pretežito koriste u optimizacijskim problemima, dok se genetsko programiranje obično koristi u strojnom učenju. Znači da u genetskom programiranju

tražimo model koji će najbolje biti prilagođen podacima. Problemi modeliranja mogu biti gledani kao na poseban slučaj optimizacije. Točno to se i radi u genetskom programiranju. Vrijednost kvalitete jedninke je vrijednost modela koja se želi maksimizirati. Skica algoritma genetskog programiranje je dana u tablici 9.

Reprezentacija	Struktura stabla
Rekombinacija	Razmjena podstabala
Mutacija	Nasumična promjena u strukturi stabla
Odabir roditelja	Proporcionalnost kvalitete
Selekcija preživjelih	Generacijska zamjena

Tablica 9: Skica genetskog programiranja

Inicijalizaciju možemo sprovesti na različite načine obzirom da radimo sa strukturom stabla. Najčešća metoda koja se koristi u genetskom programiranju je tzv. "pola-pola" metoda. U ovoj metodi prvo je odabrana maksimalna dubina stabla $D_{\max}$, a zatim je svaki član inicijalne populacije kreiran iz skupa funkcija F i terminala T koristeći jednu od iduće dvije metode. Prva se metoda zove "Full method". U njoj svaka grana stabla ima dubinu $D_{\max}$. Sadržaji vrha na dubini d su izabrani iz F ako je $d < D_{\max}$ ili iz T ako je $d = D_{\max}$. Druga metoda se zove potpuna ili na engl. "Growth method". U toj metodi grane imaju različite dubine do $D_{\max}$. Stablo je konstruirano od korijena gdje je sadržaj vrha izabran stohastički iz skupa $F \cup T$ ako je $d < D_{\max}$. Te dvije metode se izabiru sa jednakom vjerojatnošću.

Preselekcija se često koristi kada dođe do prevelike populacije što dosta često zna biti slučaj u genetskom progamiranju gdje zna doći do populacije od nekoliko tisuća. Prvo rangiramo pojedince unutar populacije, zatim ih podijelimo u dvije grupe. Jedna grupa je top $x\%$ dok je druga grupa ostalih $(100 - x)\%$. Kada odaberemo roditelje, 80% selekcije dolazi iz prve grupe, a ostatak iz druge.

Preživljavanje najjačeg je fenomen koji se pojavljuje u genetskom programiranju gdje prosječna veličina stabla raste za vrijeme algoritma. Zašto dolazi do toga uopće? Jedan od mogućih razloga je činjenica da imamo kromosome različite dužine, pa zbog toga tijekom evolucije dolazi do povećanja veličine kromosoma. Najjednostavniji način sprječavanja

"napuhavanja" je da uvedemo maksimalnu veličinu stabla i zabranimo varijacijskom operatoru da djeluje ako veličina djece bude veća od maksimalne veličine stabla koju smo uveli. U tom slučaju na to možemo gledati kao na dodatan parametar rekombinacije i mutacije. Također su razvijene i neke druge metode u sprječavanju tog problema. Najpoznatija je "parsimony pressure". U njoj se koristi uvjet kazne u funkciji kvalitete gdje se smanjuje kvaliteta velikih kromosoma ili korištenje multiobjektnih tehnika.

5.5 Klasifikatorski sustavi

Klasifikatorski sustavi su kognitivni sustavi koji imaju mogućnost spoznaje događaja iz svoje okoline i reakcije na te događaje. Klasifikatorski sustavi se ponajprije koriste u problemima u kojima je cilj razviti sustav obzirom na trenutno stanje okoliša, tj. inputa u sustav, predlažeći odgovor koji će na neki način maksimizirati nagradu u budućnosti iz okoliša.

Klasifikatorske sustave dijelimo na klasifikatorske sustave koji nemaju sposobnost učenja (non-learning classifier systems, nLCS) i klasifikatorske sustave koji imaju sposobnost učenja (learning classifier systems, LCS). U ovom poglavlju ćemo se ponajviše baviti klasifikatorskim sustavima koji imaju sposobnost učenja. LCS je kombinacija klasifikatorskog sustava i algoritma učenja. Klasifikatorski sustav se obično sastoji od skupa pravila gdje svaki od njih vodi određeni input do neke akcije. Cijeli skup pravila određuje model koji pokriva prostor mogućih inputa i predlaže optimalnu akciju za svaki input. Komponente algoritma učenja su implementirane evolucijskim algoritmom čiji članovi populacije predstavljaju pravila za svaku jedinku ili kompletan skup pravila, što je inače poznato kao Michigan i Pittsburgh pristup. Funkcija kvalitete koja se provodi tijekom evolucijskog procesa je inspirirana mnogim različitim vrstama učenja. Trenutno ćemo se ograničiti na učenje s nadzorom gdje se u svakom trenutku sustav dobiva trening signal, odnosno nagradu, od okoliša obzirom na output koji predlaže. To nam pomaže kako bi razlikovali Michiganov pristup od Pittsburghovog pristupa. U prvome su podaci prezentirani sustavu jedan po jedan i pravila jedinki su nagrađena obzirom na njihove predikcije. Nasuprot toga, kod Pittsubrghovog pristupa svaka jedinka reprezentira kompletan model, pa bi kvaliteta bila normalno izračunata prezentirajući cijeli skup podataka i računajući

prosječnu točnost predikcija.

Michigan stil LCS-a je prvi puta opisao Holland 1976. godine kao okvir za proučavanje učenja u sustavu uvjet/akcija koristeći genetski algoritam kao glavnu metodu za otkrivanje novih pravila i nagrade za uspješne jedinke. Svaki član populacije je jedno pravilo koje reprezentira parcijalni model. Stoga, cijela populacija čini naučeni model. Svako pravilo je torka $(uvjet, akcija, nagrada)$. Uvjet specificira dio mogućih inputa na koje se pravilo odnosi. Dio uvjeta može sadržavati wildcard ili "nije me briga" dio za pojedine varijable ili može opisivati skup vrijednosti koje dana varijabla može biti, kao npr. da je to neprekidna varijabla. Pravila mogu biti razlikovana brojem wildcardova koje sadrže ili mogu biti razlikovana rasponom koje određena varijabla može poprimiti. Dosta čest slučaj je da se uvjeti podudaraju, pa je moguće da određeni input zadovoljava više pravila. Podskup pravila čiji uvjeti odgovaraju trenutnom inputu od okoliša zove se odgovarajući skup. Ta pravila navode na određenu akciju. Akcija specifično može biti određena akcija koju treba poduzeti ili predikcija sustava. Podskup odgovarajućeg skupa koje zagovara odabranu akciju zove se skup akcije. Ideja je da snaga pravila radi predikciju vrijednosti nagrade koju će sustav dobiti ako poduzme određenu akciju, ali to se nažalost pokazalo teško za provesti u praksi.

Tijek rada algoritma je sažet u idućim koracima:

1. Novi skup pravila je dobiven od okoliša.
2. Bazično pravilo se ispituje kako bi se pronašao odgovarajući set pravila.
    - Ako je odgovarajući set prazan, tada se generira jedan ili više odgovarajućih pravila nasumičnom akcijom.
3. Pravila u odgovarajućem skupu se grupiraju obzirom na njihove akcije.
4. Za svaku od tih grupa se računa prosječna točnost pravila.
5. Odabrana je akcija i njena odgovarajuća grupa je set akcije.
    - Ako je sustav u "iskorištavajućem" krugu, akcija sa najvećom prosječnom točnošću je odabrana.

- Ako je sustav u "istraživačkom" krugu, akcija je odabrana nasumice ili pomoću proporcionalno-kvalitativne selekcije obzirom na prosječne točnosti.

6. Akcija se tada sprovodi i nagrada je dobivena od okoliša.

7. Procijenjena točnost i procijenjena nagrada su aktualizirani za pravila koja su se nalazila i trenutno se nalaze u setu akcija, obzirom na primljene nagrade i procijenjenu isplatu koristeći Widrow-Hoff mehanizam.

8. Ako je sustav u "istraživačkom" krugu, onda evolucijski algoritam se pokreće trenutnim skupom akcija, kreirajući nova pravila (sa isplatom i prosječnom točnošću svojih roditelja) i brisajući ostala.

Skica algoritma se nalazi u tablici 10.

Reprezentacija	torke (uvjet, akcija, isplata i točnost)
Rekombinacija	One-point križanje na uvjetima/akcijama
Mutacija	Binarno/tercijarno resetiranje obzirom na odgovarajući uvjet/akciju
Odabir roditelja	Proporcionalnost kvalitete sa dijeljenjem unutar određenog dijela okoliša
Selekcija preživjelih	Stohastički, obrnuto povezano sa brojem pravila koja pokrivaju dio okoliša
Kvaliteta	Isplati se aktualizira procijenjena nagrada i točnost pravila unutar skupa akcija

Tablica 10: Skica Michigan stila LCS-a

Pittsburgh stil LCS-a je nešto drugačiji, ali sličniji algoritmu genetskog programiranja. Svaki član populacije predstavlja kompletan model pridruživanja iz prostora inputa u prostor outputa. Svaki gen u jedinki obično predstavlja pravilo. Također kao što je već prije spomenuto, i u ovoj vrsti algoritma novi input može odgovarati većem broju pravila te se tada u tom slučaju uzima prvo podudaranje, odnosno prvo pravilo koje odgovara zadanom inputu. Zbog toga je reprezentacija izražena u obliku poredane liste te su tako dvije jedinke koje odgovaraju istom pravilu (ali to pravilo koje se ne nalazi na istom mjestu unutar te dvije liste) zapravo skroz drugačiji modeli. Učenje kompleksnog modela je odrađeno pomoću varijabilne duljine reprezentacije tako da se nova pravila

mogu dodavati u bilo kojem trenutku. Ovakav pristup ima nekoliko konceptualnih prednosti, pogotovo zato što se kvaliteta nagrađuje kompletnim skupovima pravila, tada se modeli mogu naučiti za kompleksne višesložne probleme. Mana takve fleksibilnosti je što Pittsburgh stil LCS-a, kao i algoritam genetskog programiranja, često može dovesti do "napuhavanja" i tada prostor pretraživanja postane potencijalno beskonačan. Unatoč tome, ako imamo dovoljno dobre komputacijske resurse i korištenjem metode "parsimony pressurea", taj se problem može izbjeći. Pittsburgh stil LCS-a se pokazao odličnim u nekoliko domena strojnog učenja, pogotovo u primjeni u bioinformatici i medicini gdje je ljudska intepretacija razvijenih modela ključna i veliki skupovi podataka su dostupni kako bi se sustav razvio te minimizirao grešku predikcije.

5.6 Diferencijska evolucija

Diferencijska evolucija je jedna od mlađih vrsta evolucijskih algoritama, ali zato i jedna od snažnijih. Prvi puta se pojavljuje 1995. kada su Storn i Price objavili tehničko izvješće u kojem su opisivali glavne principe nove heurističke metode za minimiziranje nelinearnih i nediferencijabilnih neprekidnih funkcija. Izraženo svojstvo koje je dovelo do imena te metode je zaokret na uobičajenu operaciju reprodukcije, tzv. diferencijska mutacija. Neka je dana populacija kandidata za rješenje u $\mathbb{R}^n$. Tada je novi mutantski vektor $x^{'}$ dobiven dodajući perturbacijski vektor (p) na već postojeći vektor

$$x^{'} = x + p,$$

gdje je perturbacijski vektor zapravo skalirani vektor razlike između dva nasumično odabrana vektora iz populacije, tj.

$$p = F * (y - z),$$

gdje su y, z nasumično odabrani vektori iz populacije, a $F > 0$ je realan broj koji kontrolira brzinu kojom populacija evoluira. Drugi reprodukcijski operator je uobičajeno križanje koje sadrži jedan parametar, a to je vjerojatnost križanja $Cr \in [0, 1]$ koje definira šansu da za bilo koju poziciju roditelja koji su u tom trenutku u procesu križanja, da će alele prvih roditelja biti uključeni u dijete. Diferencijska evolucija također ima mali zaplet kod operacije križanja. Na jednoj nasumice odabranoj poziciji, dječje alele su uzete od prvog

roditelja, ali ne nasumičnim odabirom. To osigurava da dijete ne bi bila kopija drugog roditelja. U diferencijskoj evoluciji su populacije liste, a ne skupovi (ili čak multiskupovi). Tada i-ti element u listi korespondira elementu koji je na i-tom mjestu u listi. Poredak individualaca u populaciji $P = (x_1, ..., x_i, ..., x_m)$ nije određen njihovom vrijednošću. Evolucijski krug započinje kreiranjem mutantskog vektora populacije $M = (v_1, ..., v_m)$. Za svakog novog mutanta v_i, tri vektora su odabrana nasumice iz populacije P, bazni vektor da mutira i ostala dva za perturbacijski vektor. Nakon što smo dobili mutantski vektor populacije, tzv. pokusni vektor populacije $T = (u_1, ..., u_m)$ je kreiran, gdje je u_i rezultat križanja v_i i x_i. U zadnjem koraku deterministička selekcija se provodi na svaki par x_i i u_i. Tada je i-ti individualac nove generacije u_i ako je $f(u_i) > f(x_i)$, a x_i ako je $f(u_i) < f(x_i)$.

Skica algoritma diferencijske evolucije je dana u tablici 11.

Reprezentacija	Realni vektori
Rekombinacija	Uniformno križanje
Mutacija	Diferencijska mutacija
Odabir roditelja	Uniformna nasumična selekcija 3 vektora
Selekcija preživjelih	Deterministička elitistička zamjena (roditelj protiv dijeteta)

Tablica 11: Skica diferencijske evolucije

Diferencijska evolucija u pravilu ima 3 parametra. Faktor skaliranja F, veličinu populacije m i vjerojatnost križanja Cr. Vrijedno je spomenuti da se ponekad Cr može gledati kao i na stopu mutacije, odnosno na vjerojatnost da će alele biti nasljeđene od mutanta. Broj nasljeđenih alela ima binomnnu distribuciju, pošto je broj alela definiran kao konačan broj nezavisnih pokusa u kojem imamo dva ishoda sa konstantnim vjerojatnostima.

Kroz godine nekoliko je vrsta diferencijskih evolucija smišljeno i objavljeno. Jedna od tih vrsti sadrži malu modifikaciju, a to je kod odabira baznog vektora kod izgradnje mutantske populacije M. Može biti nasumice izabran za svaki v_i kao što je opisano ranije, ali također može biti fiksirano, odnosno da uvijek uzimamo najbolji vektor u populaciji i samo mijenjamo perturbacijski vektor. Također je moguća modifikacija da se dopusti više

od jednog razlikovnog vektora kako bi se definirao perturbacijski vektor u operaciji mutacije. Na primjer, ako koristimo dva razlikovna vektora, tada jednadžba za perturbacijski vektor glasi:

$$p = F * (y - z + y^{'} - z^{'}),$$

gdje su $y, z, y^{'}, z^{'}$ nasumično odabrani vektori iz populacije.

Kod oznake različitih vrsti diferencijskih evolucija, notacija je $DE/a/b/c$ gdje a označuje način odabira baznog vektora, npr. "rand" ili "best", b je broj razlikovnih vektora koji čine perturbacijski vektor, te c označava shemu križanja, npr. "bin" označava uniformno križanje (jer ono prati binomnu distribuciju alela kao što je ranije rečeno). Koristeći takvu notaciju, osnovna verzija diferencijske evolucije bi bila $DE/rand/1/bin$.

5.7 Optimizacija roja čestica

Optimizacija roja čestica je algoritam koji je nešto drugačiji od do sada spomenutih algoritama. Razlika je u tome što je inspiriran ponašanjem jata ptica ili riba. Na prvi pogled nema evolucije u optimzaciji roja čestica, ali algoritamski odgovara shemi evolucijskih algoritama.

Optimizacija roja čestica se prvi puta pojavila 1995. kada su Kennedy i Ebehart objavili seminarski rad o konceptima optimizacije nelinearnih funkcija koristeći metodologiju optimizacija roja čestica. Slično kao kod diferencijske evolucije, svojstvo koje ga odvaja od ostalih algoritama je zaplet na operatore križanja. Optimizacija roja čestica ne koristi križanje i njegove mutacije su definirane kroz zbrajanje vektora. Optimizacija roja čestica se razlikuje od diferencijske evolucije i većine ostalih vrsti evolucijskih algoritama je u tome da svaki kandidat za rješenje $x \in \mathbb{R}^n$ dolazi u paru sa svojim perturbacijskim vektorom $p \in \mathbb{R}^n$ što je jako slično kao kod evolucijske strategije koja koristi veličinu koraka mutacije u dijelu perturbacijskog vektora.

Kako bi lakše shvatili algoritam optimizacija roja čestica, opisat ćemo ga u dva dijela. Prvo ćemo dati opis koji će pojasniti bit sustava koristeći oznaku perturbacijskog vektora p, a kao drugo ćemo prikazati tehničke detalje vektora za brzinu v i najbolji osobni rezultat b.

Svaki član populacije u optimizacijskom roju čestica se gleda kao par (x, p) gdje je $x \in \mathbb{R}^n$

vektor kandidat za rješenje, a $p \in \mathbb{R}^n$ perturbacijski vektor koji određuje kako ćemo iz vektora rješenja dobiti novi vektor. Glavna ideja je da se novi par vektora $(x^{'}, p^{'})$ dobije iz (x, p) računajući perturbacijski vektor $p^{'}$ i onda njega nadodamo na x, tj.

$$x^{'} = x + p.$$

Perturbacijski vektor je zapravo vektor brzine v, a novi vektor brzine $v^{'}$ je definiran kao skalirana suma tri komponente: to su v i razlike vektora.

$$v^{'} = w * v + \alpha * U_1 * (y - x) + \beta * U_2 * (z - x)$$

gdje su w, α, β težine, a U_1, U_2 randomizirane matrice koje množe svaku koordinatu od $y - x$ i $z - x$. Vektori x, y, z označuju redom: trenutnu poziciju, najbolju poziciju koju je taj član populacije imao u prošlosti i najbolju poziciju bilo kojeg člana populacije u prošlosti. Kao što vidimo, potrebno je da se u memoriji spremaju najbolje pozicije iz prošlosti, tj. y i z. Iz tog razloga optimizacija roja čestica za spremanje populacije koristi listu gdje se može pristupiti i-toj jedinki. Kao i kod diferencijske evolucije, poredak nije povezan sa kvalitetom jedinki. Perturbacijski vektor nije direktno spremljen kao što bi se iz notacije (x, p) dalo zaključiti, nego je spremljen preko vektora brzine v_i i "najboljeg" vektora b_i i-te jedinke populacije. Stoga je i-ta jedinka zapravo trojka (x_i, v_i, b_i). Tijekom svakog evolucijskog kruga je trojka (x_i, v_i, b_i) zamijenjena mutantskom trojkom $(x_i^{'}, v_i^{'}, b_i^{'})$ koristeći formulu:

$$x_i^{'} = x_i + v_i^{'}$$

$$v_i^{'} = w * v + \alpha * U_1 * (b_i - x_i) + \beta * U_2 * (z - x_i)$$

te je $b_i^{'} = x_i^{'}$ ako je $f(x_i^{'}) < f(b_i)$, a inače je $b_i^{'} = b_i$.
Algoritam optimizacije roja čestica sažet je u tablici 12.

Reprezentacija	Realni vektori
Rekombinacija	Nema je
Mutacija	Zbraja se s vektorom brzine
Odabir roditelja	Deterministički (svaki roditelj kreira jednog potomka mutacijom)
Selekcija preživjelih	Generacijski (dijete zamjenjuje roditelja)

Tablica 12: Skica optimizacije roja čestica

6 Podešavanje parametara

Sada kada smo se upoznali sa različitim vrstama evolucijskih algoritama, vrijeme je da se pozabavimo parametrima algoritama i njihovim podešavanjem. Kako bi dobili konkretan evolucijski algoritam, korisnik mora specificirati određene detalje, odnosno parametre. Promotrimo najjednostavniji genetski algoritam. Kao što je opisano u poglavlju 4.1, to je algoritam sa malo stupnja slobode. Na primjer, možemo podešavati parametre operatora križanja, stopu križanja i veličinu populacije. Kako bi dobili cjelovitu verziju algoritma, za spomenute parametre moramo dati točno određene vrijednosti, kao što na primjer to mogu biti "one-point" križanje, stopa križanja od 0.5 i veličina populacije neka bude 100. Razlika u različitim vrstama istog algoritma je jednostavno u domeni unutar koje mogu parametri biti postavljeni. Parametar operatora križanja ima konačnu domenu bez ikakog numeričkog poretka unutar nje, npr. ["one-point" križanje, uniformno, prosječno], dok je domena parametra $p_c \in [0, 1]$ podskup skupa realnih brojeva. Svojstvo da li je domena dana sa ikakvim poretkom unutar nje je od iznimne važnosti jer ako je dana domena sa određenim numeričkim poretkom, tada možemo koristiti heurističku pretragu i optimizacijske metode kako bi pronašli optimalnu vrijednost. U slučaju da domena nema nikakav numerički poredak unutar sebe, onda nam jedina opcija preostaje da koristimo metodu testiranja. Razlikujemo dva tipa parametara, a to su simbolički i numerički parametri. Za oba tipa parametara se elementi domene parametara zovu parametarske vrijednosti. Ovisno o izboru dizajna algoritma, možemo imati različit broj parametara za dani evolucijski algoritam. Na primjer, ako se uvede simbolički parametar odabir roditelja turnirom, to dovodi do pojave novog numeričkog parametara, a to je veličina turnira. Također može doći do određene hijerarhije unutar parametara. Najčešće su simbolički parametri na višoj poziciji unutar hijerarhije od numeričkih parametara. Pomoću razlike između simboličkih i numeričkih parametara lakše dolazimo do razlike i klasifikacije evolucijskih algoritama. Možemo gledati na simboličke parametre kao na bitnije parametre, tj. više hijerarhije jer oni definiraju bit evolucijskog algoritma, dok su numerički parametri manje bitniji, odnosno niže hijerarhije te za njih kažemo da definiraju različite varijante istog tipa određenog evolucijskog algoritma. Zbog toga, u prethodnom poglavlju smo

se isključivo bavili simboličkim parametrima jer oni govore o razlici između evolucijskih algoritama. Stoga, dva evolucijska algoritma su različita ako se razlikuju u barem jednom simboličkom parametru, kao recimo npr. ako imaju različite operacije mutiranja. Ako su vrijednosti svih parametara specificirane (kako simboličkih, tako i numeričkih) tada dobijemo tip evolucijskog algoritma. Ako se dva tipa evolucijskog algoritma razlikuju u nekim vrijednostima numeričkih parametara, kao recimo npr. stopa mutiranja i veličina turnira, tada su to dva različita tipa istog evolucijskog algoritma. U tablici 13 su prikazana tri tipa evolucijskog algoritma, od kojih dva pripadaju istoj vrsti evolucijskog algoritma.

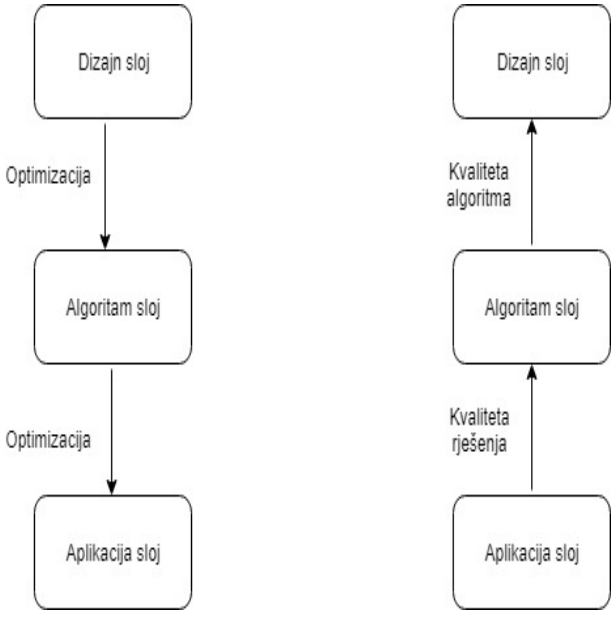
	A_1	A_2	A_3
SIMBOLIČKI PARAMETRI			
Reprezentacija	bitstring	bitstring	realni brojevi
Rekombinacija	one-point	one-point	prosječno
Mutacija	zamjena	zamjena	Gaussian $N(0, \sigma)$
Odabir roditelja	turnirski	turnirski	uniformno nasumice
Odabir preživjelih	generacijski	generacijski	$(\mu.\lambda)$
NUMERIČKI PARAMETRI			
p_m	0.01	0.1	0.05
σ	-	-	0.1
p_c	0.5	0.7	0.7
μ	100	100	10
λ	jednak μ	jednak μ	70
κ	2	4	-

Tablica 13: Primjer tipova evolucijskih algoritama

6.1 Dizajniranje evolucijskih algoritama

Dizajniranje algoritma uključuje sve odluke potrebne da bi se specificirali parametri algoritma za određeni problem (zadatak). Kako detalji dizajna evolucijskog algoritma, kao što su recimo vrijednosti parametara, imaju velik utjecaj na izvedbu i učinak algoritma,

možemo reći da je dizajniranje algoritma, pogotovo evolucijskih algoritama, je i sam optimizacijski problem.
Kako bi lakše razumjeli taj problem, izdvojit ćemo tri dijela, a to su aplikacija, algoritam i dizajn. Cijela shema je prikazana grafički na slici 2.



Slika 2: Dijagram toka evolucijskog algoritma

Na shemi vidimo da imamo kontrolni tok (lijevo) i informacijski tok (desno). Kod kontrolnog toka entitet na danom sloju optimizira entitet na sloju ispod njega, dok je kod informacijskog toka obrnut slučaj. U terminologiji evolucijskih algoritama koristimo izraz kvaliteta kada govorimo o aplikacijskom sloju i koristimo izraz korisnost kada govorimo o algoritamskom sloju. Također, koristimo izraz evaluacija samo kada govorimo o kvaliteti i testiranje kada govorimo o korisnosti. Sa tako posloženim izrazima, problem koji dizajner algoritma mora riješiti se može gledati kao optimizacijski problem u prostoru parametarskih vektora funkcije korisnosti. Stoga su rješenja problema dizajniranja evolucijskih algoritama zapravo parametarski vektori evolucijskih algoritama sa najvećom korisnošću. U tablici 14 je dan sažetak spomenute terminologije.

	Rješavanje problema	**Dizajn algoritma**
Metoda	evolucijski algoritam	procedura dizajna
Prostor pretraživanja	kvaliteta	parametarski vektori
Mjera za uspješnost	kvaliteta	korisnost
Provjera	evaluacija	testiranje

Tablica 14: Terminologija za rješavanje problema i dizajniranje algoritma

Sada možemo definirati prostor korisnosti u kojemu se nalaze parametarski vektori evolucijskog algoritma. Očito je da prostor kvalitete, koji se često koristi u evolucijskom računarstvu, ima puno toga zajedničkog sa prostorom korisnosti. Unatoč tome, postoji nekoliko razlika između ta dva prostora. Jedna od njih je da su vrijednosti kvalitete najčešće determinističke, dok su vrijednosti korisnosti uvijek stohastične jer one govore o izvedbi evolucijskog algoritma što je stohastična metoda pretraživanja. Obzirom na to zaključujemo da maksimalna korisnost mora biti definirana u nekom statističkom smislu. Posljedica toga je da usporedba parametarskih vektora evolucijskog algoritma može biti dosta teško ako podaci dobiveni evolucijskim algoritmom pokazuju veliku varijancu. Druga razlika je u tome da je kvaliteta jako povezana sa funkcijom cilja problema u aplikacijskom sloju i razlika između prikladne funkcije kvalitete je većinom u aritmetičkim detaljima. Nasuprot toga, kvaliteta ovisi o metrici izvedbe koja se koristi, a koja govori o zahtjevima korisnika i kontekstu u kojem se evolucijski algoritam koristi. Na primjer, rješavanje jednog tipa problema samo jednom ili uzastopno rješavanje problema istog tipa su dva različita slučaja sa različitim implikacijama za optimalni tip evolucijskog algoritma.

6.2 Metode dizajniranja algoritma

Svi dizajnirani algoritmi rade na prinicipu "generiraj i testiraj" tako da se generiraju parametarski vektori i testiraju se kako bi se dobila njihova korisnost. Gledajući korak generiranja, modeli mogu biti podijeljeni u dvije kategorije: ne-iterativni i iterativni modeli. Svi ne-iterativni modeli rade korak generiranja samo jednom, i to tijekom inicijalizacije, kreirajući fiksirani skup vektora. Tada je svaki od tih vektora testiran tijekom faze testiranja kako bi se pronašao najbolji vektor u danom skupu. Stoga bi se moglo reći

da ne-iterativni modeli koriste "incijaliziraj i testiraj" proceduru. Inicijalizacija može biti obavljena nasumičnim odabirom generirajući sustavsku mrežu u skupu parametara ili nekim vektorima koji bi popunili taj prostor. Najpoznatija metoda u ovoj kategoriji je često korišten parametar optimizacije kroz sustavsku usporedbu nekoliko kombinacija vrijednosti parametara, npr. četiri vrijednosti stope mutacije, četiri vrijednosti stope križanja, dvije vrijednosti veličine turnira i četiri vrijednosti veličine populacije. Za razliku od toga, iterativni modeli ne fiksiraju skup vektora tokom inicijalizacije, nego započinju malim skupom incijalizacije i kreiraju nove vektore svaki idući put. Česti primjeri takvih metoda su meta evolucijski algoritmi i iterativne metode odabira.

Gledajući korak testiranja, razlikujemo dva tipa modela: jednofazne i višefazne procedure. U oba slučaja, modeli rade veći broj testova kako bi dobili dovoljno dobru procjenu korisnosti. To je krajnje potrebno zbog stohastične prirode evolucijskih algoritama. Razlika između te dvije procedure je u tome da jednofazna procedura radi isti broj testova za svaki vektor, dok višefazna procedure koristi modificiraniju strategiju. U takvim strategijama se poslije koraka testiranja dolazi do koraka odabira u kojemu se samo odabrani vektori prosljeđuju dalje u završno testiranje.

Također se može napraviti podjela metoda dizajniranja algoritama obzirom na korištenje meta modela prostora korisnosti. Obzirom na to, imamo podjelu na dvije glavne klase: pristupi neovisni o modelu i pristupi ovisni o modelu.

7 Kontrola parametara

Sada ćemo objasniti kako raditi kontrolu parametara za vrijeme rada evolucijskog algoritma. U prethodnom poglavlju je pokazano kako podešavanje parametara može uvelike poboljšati rad evolucijskog algoritma. No podešavanje vrijednosti parametara ima jednu manu, a to je da se ono radi prije nego je algoritam počeo raditi i te vrijednosti ostaju fiksirane za vrijeme njegovog rada. Ciklus rada evolucijskog algoritma je dinamičan i adaptivan proces. Stoga je fiksiranje parametara suprotno tom duhu. Također, ponekad su različite vrijednosti parametara optimalne u drugačijim fazama evolucijskog procesa. Na primjer, veliki korak mutacije može biti dobar u ranijim generacijama dok se istražuje prostor pretraživanja, a manji korak mutacije će možda biti kasnije prikladniji za podešavanje kandidata za rješenje. Obzirom na to, statički parametar (vrsta parametra koji je fiksiran) koraka mutacije bi doveo do inferiornijeg tipa evolucijskog algoritma.
Kako bi zaobišli ograničenja statičkih parametara, umjesto parametara p koristit ćemo funkciju $p(t)$, gdje t označava broj generacije ili bilo koju drugu mjeru za vrijeme koje je prošlo. Problem pronalaska optimalnih statičkih parametara za pojedini problem je dovoljno teško. Stoga je dizajniranje optimalnih dinamičkih parametara još teži problem. Mehanizmi modificiranja parametara tokom rada evolucijskog algoritma se javljaju prilično rano. Na primjer, evolucijske strategije su promijenile mutacijske parametre tokom rada na Rechenbergovo "$\frac{1}{5}$ pravilo uspjeha" koristeći informacije o omjeru uspješnosti mutacija. Davis je eksperimentirao sa promjenom stope križanja u genetskim algoritmima obzirom na napredak pojedinih operatora križanja. Često zajedničko svojstvo takvih metoda je mehanizam ljudske procjene koje uzima u obzir informacije o uspješnosti pojedinih parametarskih vrijednosti. Jedan od često korištenih pristupa je baziran na opažanju da pronalazak dobrih parametarskih vrijednosti je loše strukturiran i kompleksan problem. Takvi problemi su točno onakvi za kakve se i koriste evolucijski algoritmi. Stoga je očita ideja da se koristi evolucijski algoritam za podešavanje parametara evolucijskog algoritma za dani problem. To se može učiniti koristeći meta evolucijski algoritam ili korištenjem samo jednog evolucijskog algoritma koji podešava sam sebe za dani problem. Kao što je spomenuto u Poglavlju 4.2, samoadaptacija za mutacijske parametre spada u tu katego-

riju.

7.1 Primjer promjene parametara

Neka je optmizacijski problem minimizirati

$$f(x) = f(x_1, ..., x_n)$$

i neka za ograničenja vrijede dane jednakosti i nejednakosti

$$g(x) \leq 0, i = 1, ..., q$$

$$h(x) = 0, i = 1, ..., m$$

gdje je domena varijabli dana donjom i gornjom granicom $l_i \leq x_i \leq u$, za $1 \leq i \leq n$. Za takav problem možemo dizajnirati evolucijski algoritam koji ima reprezentaciju u kojoj je svaki individualac reprezentiran vektorom koji sadrži brojeve tipa $x = (x_1, ..., x_n)$.

7.1.1 Promjena veličine koraka mutacije

Pretpostavimo da je potomstvo producirano aritmetičkim križanjem te zatim Gaussian mutacijom koja zamjenjuje komponente vektora x sa

$$x_i^{'} = x_i + N(0, \sigma).$$

Kako bi prilagodili σ koristit ćemo funkciju $\sigma(t)$ koja je defnirana određenim heurističkim pravilom i mjerom za vrijeme t. Na primjer, veličina mutacijskog koraka može biti definirana trenutnim generacijskim brojem t kao:

$$\sigma(t) = 1 - 0.9 * \frac{t}{T},$$

gdje t varira od 0 do T, maksimalnog generacijskog broja. Ovdje veličina koraka mutacije $\sigma(t)$ se sporo smanjuje od 1 na početku ($t = 0$) do 0.1 što je generacijski broj t bliži T. U ovakvom pristupu se vrijednosti parametara mijenjaju po deterministčkoj shemi. Korisnici stoga imaju punu kontrolu parametara i njihove vrijednosti u danom vremenu

t je kompletno determinirana i predvidljiva.

Tokom promjene parametara bilo bi dobro kada bi imali neku mjeru pomoću koje bi mogli odrediti je li određena vrijednost parametra bolja ili lošija za određeni problem. To se može napraviti koristeći isti σ za sve vektore u populaciji i sve varijable svakog vektora. Na primjer, Rechenbergovo "$\frac{1}{5}$ pravilo uspjeha" govori da bi omjer uspješnih mutacija prema svim mutacijama trebao biti $\frac{1}{5}$. Stoga ako je omjer veći od $\frac{1}{5}$ onda bi veličina koraka trebala bit povećana, a ako je omjer manji od $\frac{1}{5}$, tada bi se veličina koraka trebala smanjiti. Pravilo se provodi u periodičkim intervalima, recimo nakon k iteracija svaki σ je resetiran na $\sigma = \frac{\sigma}{c}$ ako je $p_s > \frac{1}{5}$, $\sigma = \sigma * c$ ako je $p_s < \frac{1}{5}$, $\sigma = \sigma$ ako je $p_s = \frac{1}{5}$, gdje je p_s relativna frekvencija uspješnih mutacija. Koristeći ovaj mehanizam, promjene vrijednosti parametara su bazirane na povratnoj informaciji. Utjecaj korisnika je puno manje direktniji nego u gore opisanoj determinističkoj shemi.

Također možemo pridružiti određenu veličinu koraka svakom rješenju i ko-evoluirati sa vrijednostima koje odgovaraju kandidatima rješenja. Stoga ćemo povećati reprezentaciju individualaca da budu duljine $n+1$ oblika $(x_1, ..., x_n, \sigma)$ i primjeniti određene varijacijske operatore (npr. Gaussian mutaciju i aritmetičko križanje). Stoga će i veličina koraka mutacije također podlegnuti evoluciji.

7.1.2 Promjena koeficijenata kazne

Pokazat ćemo da funkcija evaluacije (posljedično tome i funkcija kvalitete) može biti parametrizirana i promjenjiva kroz vrijeme. Iako je ovo manje korištena opcija nego promjena varijacijskih operatora, svejedno može pridonijeti neke korisne mehanizme za poboljšanje izvedbe evolucijskog algoritma.

Kada se bavimo ograničenim optimizacijskim problemima, često se koristi kaznena funkcija. Zajednička tehnika je metoda statičke kazne koja zahtjeva kaznene parametre unutar evaluacijske funkcije:

$$eval(x) = f(x) + W * penalty(x),$$

gdje je f funkcija cilja, $penalty(x)$ je nula ako se prekršaj ograničenja nije dogodio i pozitivna u suprotnom, te W je korisničko definirana težina. Skup funkcija f_j $(1 \leq j \leq m)$ može biti korišten za stvaranje kazne gdje funkcija f_j mjeri prekršaj na $j-tom$ ograničenju,

te su članovi tog skupa jednaki: $f_j(x) = max(0, g_j(x))$, ako je $1 \leq j \leq m$ ili $f_j(x) = h_j(x)$, ako je $q + 1 \leq j \leq m$. Kako bi tokom vremena prilagodili evaluacijsku funkciju, možemo zamijeniti statički parametar W sa funkcijom $W(t)$. Na primjer,

$$W(t) = (C * t)^{\alpha},$$

gdje su C i α konstante. Uočimo da pritisak kazne raste sa evolucijskim vremenom sa parametrima $1 \leq C$ i $1 \leq \alpha$.

Druga opcija je da koristimo povratnu informaciju iz procesa pretraživanja. U jednom primjeru metoda smanjuje komponentu kazne $W(t + 1)$ za generaciju $t + 1$ ako su svi individualci u zadnjih k generacija bili mogući i povećava kaznu ako su svi najbolji individualci u zadnjih k generacija nisu bili mogući. $W(t)$ se ažurira u svakoj generaciji na idući način:

$$W(t+1) = \begin{cases} \frac{1}{\beta_1} * W(t), & b \in \kappa, t - k + 1 \leq i \leq t \\ \beta_2 * W(t), & b \in S \backslash \kappa, t - k + 1 \leq i \leq t \\ W(t), & inače \end{cases}$$

S je skup svih točaka pretraživanja (rješenja), $\kappa \subset S$ je skup svih mogućih rješenja., b označava najboljeg individualca obzirom na funkciju evalucije u generaciji i, $\beta_1, \beta_2 \geq 1$ i $\beta_1 \neq \beta_2$.

Treća mogućnost je da dopustimo samoadaptaciju parametara težine, slično kao što smo radili sa veličinom koraka mutacije u prethodnom poglavlju. Na primjer, moguće je proširiti reprezentaciju individualaca na $(x_1, ..., x_n, W)$ gdje je W težina koja prolazi mutaciju i rekombinaciju kao i ostale komponente. Možemo uvesti pojedinačnu kaznu za svako ograničenje. Stoga dobivamo vektor težina i možemo još proširiti reprezentaciju na $(x_1, ..., x_n, w_1, ..., w_m)$. Sada definiramo

$$eval(x) = f(x) + \sum_{j=1}^{m} w_j f_j(x)$$

kao funkciju koju treba minimizirati. Varijacijski operatori mogu biti korišteni i na x i na w dio kromosoma.

Važno je istaknuti bitnu razliku između samoadaptacije veličine mutacijskog koraka i

granične težine. Čak i ako se veličina koraka mutacije nalazi u kromosomu, evaluacija kromosoma je neovisna o pravoj vrijednosti σ. Odnosno,

$$eval((x, \sigma)) = f(x)$$

za svaki kromosom (x, σ). U suprotnom, ako se granične težine nalaze kromosomu, tada imamo

$$eval((x, w)) = f_w(x),$$

za svaki kromosom (x, w). To bi moglo omogućiti evoluciji da vara u smislu da se poboljšava tako da minimizira težine umjesto da optimizira f i zadovoljava ograničenja.

7.2 Klasifikacija kontrolnih tehnika

Kod klasificiranja kontrolnih tehnika parametara mnogi aspekti mogu biti uzeti u obzir. Na primjer:

1. Što je promijenjeno? (npr. reprezentacija, evaluacijska funkcija, stopa mutacije itd.)
2. Kako su promjene napravljene? (npr. deterministički heuristički, samoadaptivno ili heuristika obzirom na povratnu informaciju)
3. Dokazi obzirom na koje su promjene napravljene (praćenje efikasnosti rada algoritma, raznolikost populacije itd.)
4. Područje/razina promjene (npr. populacijska razina, individualna razina itd.)

7.2.1 Što je promijenjeno?

Metode promjene vrijednosti parametara mogu se podijeliti u tri kategorije.

- **Deterministička kontrola parametara**

  Vrijednosti parametara mijenjane su nekim determinističkim pravilom koje je u pravilu predeterminirano (najčešće od strane korisnika) bez ikakve povratne informacije iz pretrage. Najčešće je pravilo aktivirano u određenim intervalima.

- **Adaptivna kontrola parametara**

  Postoji oblik povratne informacije iz pretrage koji služi kao input mehanizmu koji određuje promjenu. Ažuriranjem vrijednosti parametara može sadržavati određeni određenu dodjelu vrijednosti obzirom na kvalitetu rješenja koje je otkriveno različitim operatorima ili parametrima tako da mehanizmi ažuriranja mogu razlikovati zasluge natjecateljskih akcija. Bitna stvar za istaknuti je da mehanizmi ažuriranja nisu dio uobičajenog evolucijskog kruga.

- **Samoadaptivna kontrola parametara**

  Za implementaciju samoadaptacije parametara je korištena evolucija evolucije. Parametri koji će biti adaptirani se nalaze u kromosomu i prolaze mutaciju i rekombinaciju. Bolje vrijednosti vode boljim individualcima koji stoga imaju veće šanse za preživljavanje i mogu producirati bolje potomostvo koje dovodi do boljih parametarskih vrijednosti.

7.2.2 Dokazi obzirom na koje su promjene napravljene

Treći kriterij za klasifikaciju tiče se dokaza koji se koriste za promjenu vrijednosti parametara. Najčešće se progres pretrage prati gledajući efikasnost operatora, raznolikosti populacije itd. te se informacije skupljaju i koriste kao povratna informacija za podešavanje parametara. Iz te perspektive možemo uočiti razliku između dva slučaja:

- **Apsolutni dokazi**

  Koristi se pravilo u kojem se mijenja vrijednost parametra ako se neki već unaprijed definirani događaj (uvjet) dogodio (ispunio). Na primjer, može se povećati stopa mutacije kada raznolikost populacije bude manja od određene vrijednosti ili se može povećati populacija obzirom na kvalitetu i varijancu. Suprotno od determinističke kontrole parametara gdje se pravilo aktivira nekim determinističkim okidačem (npr. određeno vrijeme koje je prošlo), ovdje se koristi povratna informacija iz pretrage. Takvi mehanizmi zahtijevaju da korisnik ima jasnu intuiciju u kojem smjeru bi htio upravljati parametrima u slučajevima koji mogu biti unaprijed određeni.

- **Relativni dokazi**

  Vrijednosti parametara se tokom jedne iteracije uspoređuju obzirom na to imaju li pozitivan ili negativan utjecaj te se tada nagrađuju bolje vrijednosti. Smjer promjene parametara nije unaprijed definiran, nego relativno obzirom na efikasnost rada različitih vrijednosti, stoga je potrebno imati više od jedne vrijednosti u svakom trenutku.

7.2.3 Područje/razina promjene

Kao što je već prije spomenuto, svaka promjena unutar bilo koje komponente evolucijskog algoritma može utjecati na gen (parametar), cijeli kromosom (individualca), cijelu populaciju, drugu komponentu (npr. selekciju) ili čak na evaluacijsku funkciju. To sve zovemo područje ili razina promjene. Područje ili razina promjene nije neovisna, nego ovisi o drugim komponentama i o tome gdje se promjena dogodila. Na primjer, promjena veličine koraka mutacije može uzrokovati promjene gena, kromosoma ili čak i cijele populacije, ovisno o implementaciji te promjene, ali promjene u koeficijentu kazne obično zahvaća cijelu populaciju. Obzirom na sve to, svojstvo područje promjene je sekundarno, odnosno ono ovisi o danim komponentama i njihovoj implementaciji.

8 Zaključak

Optimizacijski problemi su jedni od najčešćih i najbitnijih zadataka u današnjem svijetu primjenjene matematike. Upravo za takve probleme mogu se koristiti evolucijski algoritmi. Njihova prednost leži u tome što su inspirirani evolucijom koja je sama po sebi optimalna, odnosno uvijek se optimizira. Prednost evolucijskih algoritama je da su vrlo prilagodljivi i može ih se koristiti u raznim problemima. U radu su prezentirani principi evolucijskog algoritma, njegove komponente i najpoznatije vrste.

Može se zaključiti da su evolucijski algoritmi neizbježni u današnjem svijetu u kojem dominiraju optimizacijski problemi koji se pojavljuju čak i u najobičnijim sferama života. Vrlo su jednostavni za implementaciju i skraćuju vrijeme potrebno za rješavanje takvih problema.

Literatura

[1] A. E. Eiben, J. E. Smith, Introduction to Evolutionary Computing, Springer, 2015.

[2] Introduction to Evolutionary Computing, URL: https://www.evolutionarycomputation.org/ [pristupljeno svibanj, 2022.]

[3] Pregled evolucijskih algoritama, Ivan Rep, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu

[4] Towards data science, URL: https://towardsdatascience.com/introduction-to-evolutionary-algorithms-a8594b484ac [pristupljeno travanj, 2022.]