Poročilo za informatiko

- OPIS PROGRAMERSKEGA PROBLEMA

Programerski problem te naloge je bil implementacija preprostega genetskega algoritma za namen generiranja določenih besed – oz. nizov. Sam projekt ni namenjene za dejansko uporabe, lahko pa služi kot podpora drugim problemom, ki potrebujejo tovrstni algoritem. V tem primeru gre tako nekako za »Hello World« program genetskega algoritma.

Podrobnejši opis problema

Genetski algoritem generira izhode, zanje pa dobi le podatek o njihovi uspešnosti. S pomočjo teh podatkov mora v čim manj poizkusih priti do pravilnega odgovora. Uspešnost izhoda ovrednoti »trenirna« funkcija, v angleščini in žargonu genetskih algoritmov imenovana tudi »fitness function«[1]

Cilj algoritma je generacija niza, ki je enak ciljnemu nizu. Za to mora porabiti čim manj ciklov poizvedb z »trenirno« funkcijo.

Vhodni parametri

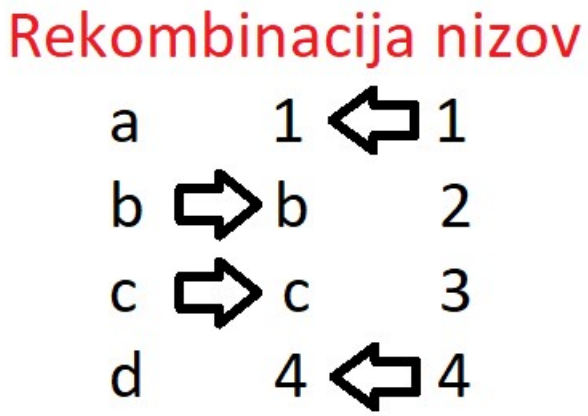
Uporabnik v program vnese želeni ciljni niz in nato še nekaj vrednosti, ki so pomembne za delovanje algoritma, a bi ji lahko imel program nastavljene v neki nastavitveni datoteki

Izhod

Algoritem izvozi najboljši niz in njegovo vrednost »trenirne« funkcije v posamezni generacji

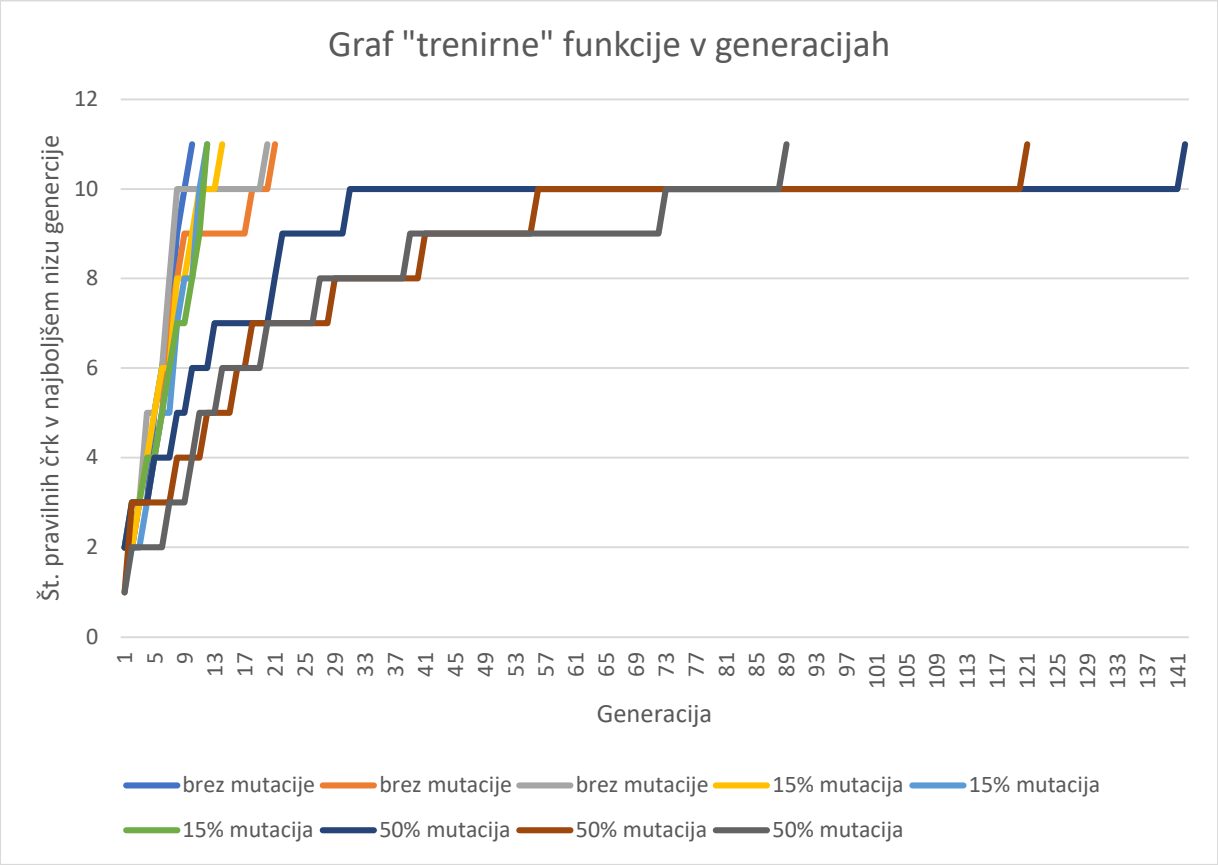
- NAČIN DELOVANJA ALGORITMA

Genetski algoritmi temeljijo na podlagi dedovanja. Posamezni niz deduje od enega ali več starševskih nizov; njihovo število je odvisno od implementacije algoritma. Navadno to deluje podobno kot v biologiji; otroški niz je sestavljen iz delčkov starševskih nizov, podobno kot deluje podvajanje kromosomov. Določene implementacije algoritma način rezanja in kombiniranja nizov med sabo določen s konstantami, a v mojem primeru gre za naključno mešanje.



[1] https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

Drugi pomemben del genetskih algoritmov so mutacije. Te rahlo spremenijo določene nize in na ta način povečajo raznovrstnost »dednega materiala« v algoritmu. Te v nobeni implementaciji algoritma niso konstantne, od algoritma do algoritma pa je odvisno, kaj je dovoljeno tem mutacijam spreminjati oziroma, kaj se deduje na splošno. V mojem primeru se deduje vsebina nizov, ne spreminja pa se npr. njihova dolžina. Odvisna je tudi količina mutacij, ki je optimalna za določen algoritem; večja količina mutacija lahko povzroči nestabilno premikanje »trenirne« funkcije, manjša pa navadno ustavlja hitrost razvoja dednega materiala in močno zmanjšuje raznovrstnost le tega. To je dobro vidno iz spodnjega grafa. Uporabnik nastavi količino mutacij na začetku delovanja programa.



Tretji pomembni element genetskih algoritmov pa je izbor. Glavni razlog, zakaj algoritem postaja čedalje boljši je, ker se lahko razmnožujejo le najboljši organizmi. Količina teh in način izbora je spet odvisen od implementacije, nekateri izmed algoritmov celo vzamejo nekaj zelo slabih nizov skupaj z dobri, a moja implementacija vzame najboljših N procentov nizov; število N vnese uporabnik.

Pomemben del je tudi na koliko nizov se razmnoži izbrani niz. V moji implementaciji algoritma se boljši nizi razmnožijo večkrat od slabših, vendar pa to po mojih opažanjih ni zelo pogosto v ostalih implementacijah zato to ni standardni del algoritma.

Česa še ne vem, ne znam, nisem še sprogramiral	*Kaj že vem, znam oziroma sem že izdelal*
Povezava genetskih algoritmov z splošno uporabnim namenom	*Osnovni genetski algoritem*
	Naključno generiranje nizov

	»Trenirna« funkcija
	Rekombinacija
	Mutacija

- DOKAZ O NAUČENEM

Koncept zaporednosti izvajanja ukazov	*Ko se program zažene, se ukazi izvajajo eden za drugim*
	Ko se progam zažene, začne delovati po naslednjih korakih:
	• *Naloži vse tri knjižnice*
	• *Nastavi tri spremenljivke na vhod uporabnika*
	• *Nastavi in izračuna spremenljivko za količino preživelih v populaciji*
	• *Naredi seznam z vsemi znaki, ki jih program potrebuje*
	`import random`
	`import string`
	`import math`
	`goal_string = input('Vpisite ciljni niz: ')`
	`population_size = int(input('Vpisite velikost populacije (minim`
	`survival_ration = int(input("Vpisite kolicino osebkov populacij`
	`v odstotkih "))`
	`mutation_rate = int(input('Vpisite moznost mutacije v procentih`
	`goal_string = "Jakob Kralj"`
	`population_size = 100`
	`survival_ration = 50`
	`half = int(population_size*survival_ration/100)`
	`alpha = string.ascii_letters + ' '`

Koncept vejitve (if stavek)	*Program izvede ukaze glede na vrednost nekega izraza.*
	V mojem primeru koda preveri, ali je naključna vrednost manjša od želene količine mutacij, v primeru da je nastavi nek znak v nizu na drug naključen znak.
	`if(random.randint(0, 100) <= mutation_rate):`
	`dnk[i] = random.choice(alpha)`

Koncept zanke	*Zanka nek ukaz ponovi večkrat.* *V mojem primeru gre zanka čez vse znake v nizu in prešteje koliko jih je enaki in na isti poziciji kot v ciljnem nizu.* `for i in range(len(dnk)): score += dnk[i] == goal_string[i]`
Dogodkovno programiranje	*Dogotkovnega programiranja v tem programu žal ni. Gre za koncept dogodkov, neka funkcija se zažene takrat, ko se nekaj zgodi. Najbližje temu v moji kodi je verjetno podnožje funkcije input() kjer koda zapiše znak ko uporabnik pritisne tipko.*
Branje in izpisovanje (IO ukazi	*Uporabnik v program vnaša podatke, program pa jih izpisuje na zaslonu. To pri meni delata funkciji input() in print()*

	`goal_string = input('Vpisite ciljni niz: ')`
	`population_size = int(input('Vpisite velikost populacije (minimaln`
	`survival_ration = int(input("Vpisite kolicino osebkov populacije, `
	`v odstotkih "))`
	`mutation_rate = int(input('Vpisite moznost mutacije v procentih: '`
	`for i in steps:`
	`    print(i[1])`

Koncept spremenljivke	*Spremenljivka je prostor v pomnilniku, kjer je shranjena številka. Je splošno uporabna v programiranju do te mere, da primerov njene uporabe mrgoli že v kodi, namenjeni za prikazovanje drugih konceptov. Npr. goal_string ima shranjen ciljni niz, prek katerega se vrednosti rezultate algoritma.*
Koncept podprograma oziroma funkcije	*Podprogram ali funkcija podobno kot v matematiki vzame nek vhod in proizvede nek izhod. V moje primeru je to najbolje vidno na tihi funkciji naključna beseda, ki vzame kot vhod dolžino besede proizvede pa naključni niz dolžine tega vhoda.* `random_word = lambda n : ''.join([random.choice(alpha) for i in ra`
Tabelarična spremenljivka (list)	*Liste lahko v sebi hranijo več spremenljivk. Prek njih je možno tudi iterirati – iti čez listo. Vsi nizi se v mojem programi obnašajo kot tabele, najbolj očitno pa je to pri tabeli vseh znakov, definirani tukaj:* `alpha = string.ascii_letters + ' '`

ZAKLJUČEK

Nadgradnja bi vsebovala isti algoritem, a bi ga uporabil za razvijanje AI-ja kakšne igrice – torej računalniškega programa, ki bi bil zelo dober v neki igrici. Na ta način bi lahko nadgradil »trenirno« funkcijo in se tudi začel posvečati mrežam nevronov. Pri delu te naloge najbolj obžalujem svojo lenobo; če bi se naloge lotil prej, bi že prej ugotovil da ne znam še dovolj matematike za prave nevronske mreže. V vmesnem času bi se to matematiko mogoče že celo naučil. Sam algoritem in grafi, ki sem ji lahko narisal iz izhodnih podatkov se mi še kar dobro uspeli, dobro je bilo vidno tudi kako spremembe na parametre algoritma vplivajo na delovanje le tega. Zelo zanimivo bi bilo tudi uporabiti ta princip za program »Game of Life«… Najtežja pri tem projektu je bila gotovo teorija, ko je bila ta končana je bil program testiran in napisan v 1 uri in pol.

NAVODILA ZA UPORABO

V program vpišite vse, kar program od vas zahteva. Za ciljni niz si ne izberite predolgega niza.

Program ni odporen na nepravilne vhode.

Viri:

https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

https://stackoverflow.com – za vse težave s kode.

Za pravopisne napake se izkreno opravičujem. Jakob Kralj