Ime projekta

Seminarski rad u okviru kursa
Računarska inteligencija
Matematički fakultet

Autor1, Aleksa Voštić
email1, akile9v@gmail.com

24. avgust 2021.

Sažetak

Kratki sazetak.Kratki sazetak.Kratki sazetak.Kratki sazetak.Kratki sazetak.Kratki sazetak.Kratki sazetak.Kratki sazetak.Kratki sazetak.Kratki sazetak.

Sadržaj

1 Uvod

Ovde ide uvod i to je to!

2 Genetski Algoritam

Genetski algoritam je zasnovan na Darvinovoj teoriji evolucije tako da u okviru jedne populacije najčešće opstaju samo najbolje prilagođene jedinke. Nove generacije jedinki se dobijaju ukrštanjem jedinki iz prethodne generacije, uzimajući u obzir da se kod nekih jedinki mogu ponekad dešavati mutacije nad njenim genima. Osnovno očekivanje je da se smenom generacija dobijaju populacije jedinki koje su bolje prilagođene u zadatom okruženju. Svaka jedinka je definisana njenim kôdom koji predstavlja skup njenih gena i na osnovu kojih možemo izračunati pomoću odgovarajuće funkcije prilagođenosti koliko se ta jedinka prilagodila okruženju. Ona jedinka koja ima najbolju funkciju prilagođenosti možemo smatrati kao najbolji rezultat koji smo dobili primenom genetskog algoritma. Sam algoritam se svodi na nekoliko važnih koraka koji se ponavljaju odgovarajući broj iteracija:

- Generisanje početne populacije
- Određivanje funkcije prilagođenosti populacije
- Sve dok nije ispunjen kriterijum zaustavljanja:
  - Izvršiti selekciju jedinki
  - Ukrstiti izabrane parove jedinki
  - Potencijalna mutacija izabranih jedinki
  - Određivanje funkcije prilagođenosti populacije
- Ispisivanje najbolje nađenog rešenja.

Takođe se može iskoristiti i princip elitizma koji se zasniva na ideji da se najbolja ili više najboljih jedinki prenesu direktno u narednu generaciju i tako izbegnu slučajevi u kojima najbolja jedinka ne pređe u narednu generaciju ili potomci budu lošiji od roditelja. [1].

U narednim sekcijama ćemo ukratko opisati na koji način smo svaki od ovih koraka implementirali.

2.1 Generisanje počene populacije i veličina populacije

Nakon učitavanja svih fudbalskih igrača koji mogu učestvovati u pravljenju jednog tima, jednu jedinku populacije ćemo definisati kao jedno rešenje koje se sastoji od 15 igrača. Način na koji se vrši izbor igrača koji će činiti jedan tim je totalno proizvoljan, naravno uz poštovanje već navedenih uslova. Ovim načinom generisanja početnih jedinki smo hteli da obezbedimo da što više različitih igrača bude izabrano u okviru jedne populacije, jer je broj igrača veliki i na taj način potencijalno obezbedimo raznovrsnije ukrštanje jedinki. Neophodno je i popraviti nekorektnosti kao što su dva ista igrača u okviru jedne iste jedinke, broj igrača na odgovarajućoj poziciji je veći/manji od dozvoljenog i slično.

Veličinu same populacije je jako teško odrediti, jer nam samu jedinku čine diskretni podaci (skup od 15 različitih igrača od 600 mogućih), pa je bilo neophodno sprovoditi odgovarajuće testove.

Ispostavilo se da za ogromne populacije (>1000), algoritam u većini slučajeva nađe dobro rešenje, što ima i logike: veća veličina populacije daće veću šansu da se prilikom same inicijalizacije u okviru populacije nalazi neko prihvatljivo dobro rešenje, ali to povlači da onda moramo imati i veći broj generacija i samim tim algoritam će se duže izvršavati (dok je potencijalno možda već našao dovoljno dobro rešenje). Ovo se isključivo ispostavilo za naš konkretan problem. Nekada je bolje imati veću veličinu populacije nego broj iteracija. [4]
Za manje vrednosti (<100), algoritam je češće dolazio do lošijih i neprihvatljivijih rešenja.
Balans smo našli negde između. Za ne toliko puno vremena i za dovoljno veliku populaciju (oko 500), algoritam daje skroz dobra i prihvaljiva rešenja, a dosta često budu čak i optimalna. [3]

2.2 Određivanje funkcije prilagođenosti populacije

Svakoj jedinki računamo prilagođenost na osnovu sledeće formule:

$$fitness = \sum_{n=1}^{15^{1}} player.evaluation$$

$$\begin{aligned} player.evaluation = x * player.total\_points \\ +y * player.form \\ +z * player.selected\_by\_percent^{2} \end{aligned}$$

Kao što možemo primetiti, na vrlo prost način određujemo prilagođenost jedne jedinke tako što odredimo zbir vrednosti igračevih parametara evaluacije. U ovom slučaju, što je veća vrednost vratila funkcija prilagođenosti jedne jedinke, to nam je rešenje prihvatljivije i bolje. Parametri *total_points*, *player.form*, *player.selected_by_percent* su se pokazali kao najbolji za ocenjivanje pojedinačnih igrača koji će činiti jednu jedinku.

2.3 Kriterijum zaustavljanja

Algoritam se zaustavlja tek nakon što prođe kroz svaku iteraciju, odnosno dok ne formira i poslednju moguću generaciju. Postavlja se pitanje koji je to optimalan broj generacija koje trebamo proizvesti da bismo došli do optimalnog rešenja?
Veliki broj iteracija može nam se na prvi pogled činiti kao dobra ideja jer na taj način omogućavamo algoritmu da što više i duže pretražuje oblast pretrage i na taj način pronađe optimalno rešenje, međutim u našem slučaju se to nije ispostavilo kao i najbolja ideja. Veliki broj iteracija (>2000) je definitivno dovodio do boljih i prihvatljivijih rešenja ali ne toliko često i do onog najboljeg rešenja, a uz to se podvlačilo i duže vreme izvršavanja algoritma.
Za mali broj iteracija (<300) algoritam je dosta češće nalazio optimalno

[1] Jedna jedinka se sastoji od 15 igrača

[2] Parametri x,y,z su prilagodljivi i potencijalno će se menjati u zavisnosti od toga čemu želimo da damo više značaja. Važno je da njihov zbir bude jedank 1.0

rešenje za razliku od velikog broja iteracija i to u kratkom vremenskom roku, međutim dosta često je nalazio i lošija rešenja koja možda i nisu toliko prihvatljiva.
Zaključili smo da je bolje da imamo dobra i prihvatljivija rešenja, ali i da obezbedimo da imamo šansu da nađemo optimalna rešenja, dok ona lošija na neki način što više zaobiđemo. Zbog toga nismo hteli da uzmemo mali broj iteracija, jer su se češće javljala ne toliko dobra rešenja, od onih koja su prihvatljivija za razliku od velikog broja iteracija gde su se više pojavljivala dobra rešenja ali ne i najbolja. Kao i u određivanju same veličine populacije, balans je negde između. Algoritam nam se najbolje ponašao za izabrani broj iteracija između 500 i 1000. [2]

2.4 Selekcija

Proces selekcije se svodi da u okviru jedne populacije odaberemo neke 2 jedinke da bi kasnije reprodukovali neke nove dve jedinke. Odlučili smo se za turnirsku selekciju jer nam se čini da najbolje pristaje našem problemu. Ideja se svodi na to da iz skupa populacije odaberemo par jedinki proizvoljno i uvidimo koja jedinka je najbolja od tih nekoliko odabranih, odnosno koja jedinka ima najbolju funkciju prilagođenosti. Na taj način treba odabrati dve takve jedinke i proslediti ih dalje za reprodukciju novih jedinki.
Veličina turnira je postavljena na vrlo malu vrednost, iako imamo puno jedinki, jer je algoritam tako davao najbolje rezultate. To može biti zbog toga što ukoliko bismo zadali veći turnir, vrlo brzo bismo došli do već nekih ustaljenih lokalnih najboljih rešenja koja bi se stalno takmičila i stalno pobeđivala, pa na taj način ne bismo dozvoljavali možda nekim drugim jedinkama koje i nisu toliko trenutno prilagodljive, ali možda imaju potencijal da njihova deca budu dosta bolja čak i od tih lokalnih najboljih jedinki. Ispostavilo se da se algoritam najbolje ponašao za veličinu turnira između 3 i 15. Za velike turnire algoritam je vrlo retko nalazio optimalna rešenja, možda 1 u 100 pokretanja, što je vrlo loše.

2.5 Ukrštanje

Nakon što procesom selekcije odredimo koje to dve jedinke treba da reprodukuju decu, neophodno je da ih ukrstimo. Ukrštanje se dešava u 100% slučajeva, odnosno u svakoj iteraciji. Postupak se svodi da prvo sortiramo hromozome oba roditelja po poziciji igrača, tako da dobijemo nešto nalik fudbalskoj formaciji [gk, gk, df, df, df, df, df, mf, mf, mf, mf, mf, fw, fw, fw] a nakon toga odaberemo proizvoljni indeks u odnosu na veličinu jedinke (vrednost između 0 i 14) i praktično na tom mestu "presečemo"oba roditelja. Decu formiramo tako što prvi deo prvog roditelja i drugi deo drugog roditelja spojimo za prvo dete i drugi deo prvog roditelja i prvi deo drugog roditelja spojimo za drugo dete. Naravno ovim načinom formiranja novih jedinki često se mogu dešavati nekorektnosti u novo sastavljenom timu, jer prilikom razmene hromozoma, može se desiti da oba roditelja na istoj poziciji imaju istog igrača, pa samim time, jedinka dete koja je nastala njihovim ukrstanjem ima dva ista igrača (npr. dva Salaha u timu), što nije ispravno. Takve i slične probleme neophodno je korektovati. Zbog toga smo uveli i posebnu funkciju korekcije koja nakon ukrštanja, proveri da li je sve dobro prošlo, a ako nije, dodatno izmeni hromozome da bismo dobili korektne vrednosti.

2.5.1 Korekcija jedinki

Pošto je problem koji rešavamo poprilično specifičan i ima dosta ograničenja i pravila koje moramo poštovati, morali smo i da napravimo odgovarajuću funkciju koje bi mogla da detektuje nekorektnosti i šta više da ih i ispravlja na odgovarajući način.
Ideja se svodi da proverimo tri glavna uslova:

- Jedinstveni igrači u timu
- Ograničenje po broju igrača iz jednog istog tima
- Poštovanje budžeta

Ukoliko se dogodi bilo koje prekoračenja, algoritam će pomoću funkcije korekcije to detektovati i probati da ustanovi na koji način da izbaci nekog igrača i zameni ga sa nekim novim igračem sa iste te pozicije. Ispostavilo se da je najbolji pristup tako što ćemo prvo detektovati problem i na osnovu njega odabrati igrača kojeg želimo da izbacimo (jer ga smatramo da narušava konzistentnost) i na osnovu njegove pozicije ubacimo totalno prozivoljnog igrača u tim. Nakon toga je naravno opet neophodno da proverimo spomenute uslove, jer mogu biti opet narušeni ubacivanjem novog igrača.

2.6 Mutacija

Ideja koja se krije iza mutacije je da se povremeno mogu dešavati neke promene u okviru jedne jedinke koje će promeniti takvu jedinku na jednom samo hromozomu, za razliku od ukrštanja, i na taj način potencijalno podstaći u pronalasku rešenja.
Ukoliko je ispunjen uslov da se dogodi mutacija, odabraće se jedan proizvoljan igrač i zameniti sa nekim drugim igračem sa te iste pozicije u timu. Takođe kao i kod ukrštanja, pozvaće funkcija koje će nakon mutacije korektovati nepravilnosti koje mogu uslediti ubacivanjem novog igrača u tim. Za naš problem je ustanovljeno da velika stopa mutacije (>0.1) dovodi do eksremno loših rešenja, odnosno ukoliko se mutacija događa često, to će i narušiti već dobro formirane jedinke potencijalnim izbacivanjem i ubacivanjem nekog proizvoljnog igrača.
Međutim i niska stopa mutacije (<0.01) se ispostavila poprilično beskorisnim jer algoritam je nalazio solidna rešenja, iako se mutacija događala retko, pa verovatno možda nije ni podsticala u nalaženju problema.
Treba naglasiti da je mutacija ipak bila možda i presudni faktor kako bi se algoritam ponašao zadovoljavajuće. Ustanovili smo da je stopa mutacije od 0.09 podigla algoritam na dosta viši nivo. Sa ovakvom stopom, algoritam je praktično uspevao da za bilo koje veličine generacije i populacije nalazi u nekoliko pokušaja pokretanja algoritma optimalna rešenja, što za neke manje ili veće vrednosti mutacije nije bio slučaj.

2.7 Elitizam

Prilikom pravljenja nove generacije pospukom ukrštanja ili mutacijom, pa kasnije i korektovanjem jedinki, mogu se dešavati situacije u kojima od roditelja koji su dobro prilagođeni dobijamo decu koja su lošije prilagođena od samih roditelja, pa samim tim potencijalno ni nema prostora u evoluciji ka najboljem rešenju. Ovo se u našem slučaju može često događati zbog načina na koji je implementirana korekcija jedinki, pa se zbog toga javila ideja da možda nekoliko najboljih jedinki iz cele populacije

direktno prenesemo u narednu generaciju. Iako nam je veličina populacije velika, ispostavilo se da ne trebamo ni previše najboljih jedinki prenositi u narednu generaciju, već samo nekoliko. Ukoliko sačuvamo 1-3 jedinke, algoritam bi se ponašao najbolje.

Literatura

[1] Denis Aličić. Genetski algoritam. Dostupno na http://poincare.matf.bg.ac.rs/~denis_alicic/ri/6.html, 2020.

[2] M.S. Gibbs, H.R. Maier, G.C. Dandy, and J.B. Nixon. *Minimum Number of Generations Required for Convergence of Genetic Algorithms*. IEEE, 2006.

[3] Stanley Gotshall and Bart Rylander. Optimal population size and the genetic algorithm. Dostupno na https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.2431&rep=rep1&type=pdf.

[4] Dana Vrajitoru. Large population or many generations for genetic algorithms? Dostupno na https://www.cs.iusb.edu/~danav/papers/LargePop.pdf.