Aproksimacija slika genetskim algoritmom koriscenjem n poligona

Seminarski rad u okviru kursa
Racunarska Inteligencija
Matematički fakultet

Matija Pejić
matija.pejic@yahoo.com

4. januar 2021.

Sadržaj

1 Uvod

U ovom radu pogledacemo neke od nacina implementacija genetskog algoritma za aproksimaciju slika pomocu n poligona u jeziku Python kao i njihove razlike, prednosti i mane.
Sve verzije algoritma kao i resenja mogu se naci na linku:
https://github.com/MatijaPejic/RI_project

2 Osnovni koncepti problema i njihova realizacija u kodu

Pre nego sto krenemo da se bavimo dizajnom algoritma moramo odgovoriti na neka osnovna pitanja:

1. Kako reprezentovati sliku?
2. Kako iscrtavati poligone?
3. Kako praviti nove aproksimacije slike od postojecih?
4. Kako uporedjivati slike (Fitness)?

U svim implementacijama koje ce se spominjati u ovom radu za reprezentaciju slika i crtanje poligona koriscena je Python biblioteka Pillow, koja predstavlja fork Python Imaging Library-a (PIL). PIL dodaje mogucnosti procesiranja slika nasem Python interpretatoru kao i podrsku velikom broju formata fajlova i efikasno interno skladistenje slika.
Kreiranje novih aproksimacija i uporedjivanje slika je usko vezano za jedne od najbitnih koncepata genetskih algoritama a to su *ukrstanje(crossover)/mutacija* i *fitness* funkcija tako da cemo taj deo izdvojiti u zasebno poglavlje.

3 Ukrstanje/mutacija i fitness funkcija

Prilikom prvih implementacija algoritma u kojim su jedinke bile citave aproksimacije sa njihovim poligonima ukrstanje smo vrsili na 2 nacina a to su : *uniformo* i *tacke prekida (breakpoints)*. Oba ova metoda na svoje jedinstvene nacine kombinuju poligone roditelja radi stvaranja dece. U kasnijim verzijama algoritma jedinke ce postati sami poligoni pa postupak stvaranja nove aproksimacije jeste samo mutiranje vec postojecih poligona. Ne postoji jedan pravi nacin da ovo uradimo, ali radi jednostavnosti, novu aproksimaciju kreiramo tako sto izaberemo jedan poligon sa trenutne aproksimacije, pomerimo mu jedno teme na nasumicnu lokaciju na slici i promenimo mu boju. Zasto smo presli sa tretiranja aproskimacije kao jedinke na to da tretiramo poligon kao jedinku bice jasnije u nastavku teksta.

Fitness fukcija je ono sto ce nam govoriti koliko su nase aproksimacije dobre tj. koliko su one blizu originalnoj slici. U resenjima koja su data u ovom radu koriscene su dve funckije:

1. Euklidsko rastojanje
2. SSIM (*eng.* Structural similarity index measure)

3.1 Euklidsko rastojanje

Ideja ove fitness funkcije je jednostavna, racunamo rastojanje boja izmedju svaka dva piksela na istim kordinatama originalne slike i nase aproksimacije. Ukupan fitness bice suma ovih rastojanja.

```
fitness = 0
for y = 0 to width
    for x = 0 to height
          c1 = GetPixel(sourceImage, x, y)
          c2 = GetPixel(generatedImage, x, y)
          deltaRed = c1.Red - c2.Red
          deltaGreen = c1.Green - c2.Green
          deltaBlue = c1.Blue - c2.Blue
          pixelFitness = deltaRed * deltaRed +
                               deltaGreen * deltaGreen +
                               deltaBlue * deltaBlue
          fitness += pixelFitness
    end
end
return fitness
```

Primer 1: Pseudo kod za euklidsko rastojanje

Sto je fitness manji to je nasa aproksimacija bolja.
Prednost ove metode jeste jednostavnost, a glavna mana je efikasnot. Sto su slike vecih dimenzija ova funckija ce raditi sve sporije i sporije jer mora preci preko sve veceg broja piksela. Jos jedna mana koju nije tako lako zapaziti jeste da ovaj metod ne mari za strukturu slike vec samo gleda da poklopi boje, ovo moze dovesti do toga da aproksimacije nemaju glavne karakteristike slike. Ipak, videcemo kasnije, ova funckija daje dobre rezultate.

3.2 SSIM

SSIM (Structural Similarity Index) je perceptivna metrika koja kvantifikuje degradaciju kvaliteta slike koja moze nastati kao posledica procesiranja slike (npr. kompresija) ili transmisije slike. Sam indeks se nalazi u intervalu [-1, 1] gde 1 predstavlja potpuno poklapanje slika sto je samo dostigljivo ako racunamo indeks na dve identicne slike. SSIM je po svemu superiorniji od Euklidskog rastojanja za potrebe naseg algoritma jer eliminise sve njegove nedostatke. SSIM poseduje brojne varijante i ima siroku upotrebu u TV i grafickoj industriji. Sama formula je jako komplikovana te je necemo ovde prikazivati.
Za nase potrebe koristicemo SSIM koji je implementiran u Python biblioteci scikit-image.

4 Razvoj algoritma

U ovom poglavlju videcemo razlicite, uspesne i neuspesne, implementacije genetskog algoritma kao i njihove prednosti i mane.
Podsecam, svi kodovi su dostupni na adresi *https://github.com/MatijaPejic/RI_project*

4.1 Klasican genetski algoritam

Ovo podglavlje se odnosi na fajlove *v1.py, v2.py i v3.py* sa github linka.
Prve verzije pratile su klasicnu definiciju genetskih alogoritama. Posedovale su veliku populaciju, metode za kreiranje inicijalnih i novih populacija, metode za

selekciju (turnirsku i ruletsku), razlicite metode ukrstanja jedinki kao i metod za mutaciju. Medjutim sam pristup problemu bio je los. U ovim verzijama jedinku u populaciji predstavljala je jedna aproksimacija zajedno sa svim svojim poligonima, ovo je cinilo sve navedene metode izuzetno sporim i jos dodatno na to za fitness korisceno je Euklidsko rastojanje. Ove verzije algoritma su potpuno ne upotrebljive i nisu dale nikakve rezultate vredne spominjanja.
U verziji 3 (*v3.py*) mozemo primetiti jedan novi metod a to je *Colorscan* (koji ce se kasnije zvati *Dominant_ colors*). Ovaj metod je jako bitan za efikasnost i navodjenje algoritma ka resenju. Njegova uloga je da izdvoji dominante boje slike i obezbedjuje da se poligoni uvek boje tim bojama, ideja ovoga je da nema smisla bojati poligone bojama koje se ne pojavljuju na slici.

4.2 Modifikacije klasicnog algoritma

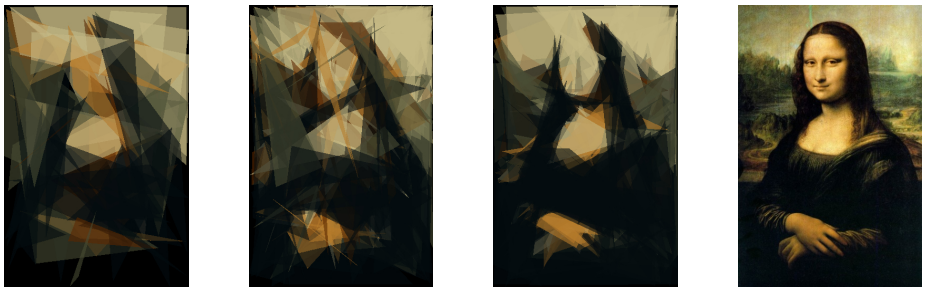
Ovo podglavlje se odnosi na fajlove *v4.py i v5.py* sa github linka.
Jasno je iz predhodnih implementacija da nase shvatanje jedinke nije bilo dobro i da je ono cinilo da efikasnost algoritma bude jako losa. U verzijama 4 i 5 (*v4.py, v5.py*) imamo skroz drugi pristup problemu. Umesto da populaciju cini veliki broj aproksimacija sada cemo imati samo 2 aproskimacije. Roditelj i dete koji se neprestano takmice izmedju sebe. Onaj koji izgubi se modifikuje sve dok ne pobedi. Pored ove izmene nase jedinke vise nisu aproksimacije vec poligoni na njima. Kada mutiramo mi menjamo poligone na jednoj aproksimaciji (onoj koja je izgubila). Najbolje resenja cuvamo sa strane kao globalno najbolje resenje, dok *simuliranim kaljanjem* pokusavamo da izbegnemo lokalne minimume. Zarad efikasnosti vise ne vrsimo selekciju jer sada imamo samo 2 aproksimacije, ukrstanje aproksimacija je nepotrebno jer uvek mutiraju poligoni jedne aproksimacije. Uz male izmene funkcije za izvlacenje boja koju smo spomenuli u predhodnom poglavlju dolazimo do verzije 5 (*v5.py*) koja je prva verzija koja je dala dobre rezultate.

4.3 Konacna verzija

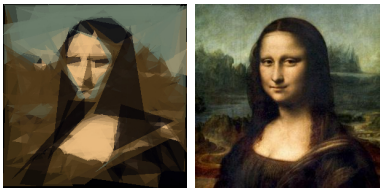
Verzija 5 je koristila Euklidsko rastojanje prilikom racunanja fitness-a i sasvim je prirodno da pokusamo da je modifikujemo da koristi SSIM za koji smo rekli da je za nas problem idealan nacin racunanja fitness-a. Tu modifikovanu verziju predstavlja verzija 8 (*8.py*)

5 Rezultati verzija 5 i 8

Iako rezultati prvog skupa slika deluju slicno bitno je napomenuti da je verzija algoritma sa SSIM-om radila znatno krace i po broju iteracija i vremenski. Jedan od problema algoritma jeste ako razvijamo sliku sa malim brojem poligona ovo moze dovesti do toga da jednostavno nema dovoljno poligona da se izrade fini detalji na slici kao na primer detalji lica.

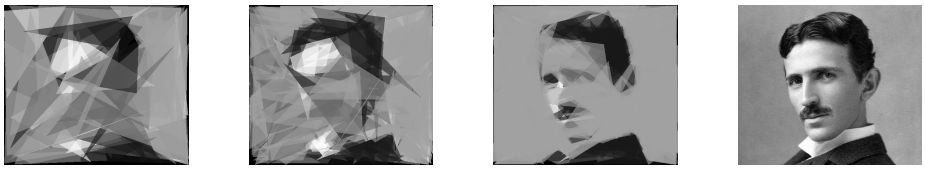


Slika 1: v5 50 poligona, v5 100 poligona, v8 100 poligona, cilj



Slika 2: v8 sa 100 poligona, cilj

Zato na sledecoj slici gde je citava slika zapravo lice, algoritam se lako fokusira na detelja i daje dobre rezultate.



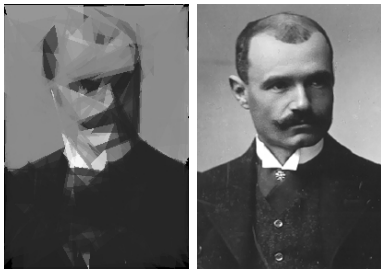
Slika 3: v5 50 poligona, v5 100 poligona, v8 100 poligona, cilj

Skup slika na kome se mozda najvise vidi razlika izmedju Euklidskog rastojanja i SSIM-a je skup slika Nikole Tesle. Slika je znatno kvalitetnija, jasnija i brze se razvila koriscenjem SSIM-a.

Slika 4: v8 100 poligona, v8 500 poligona, cilj

Ovde smo pokazali da povecavanjem poligona mozemo dobiti finije detalje. Sa samo 100 poligona veliki deo poligona se potrosi na bojenje pozadine i obuce

koje ce najvise doprineti fitness-u dok za finije (sitnije) detalje kao sto je lice, nema poligona.



Slika 5: v8 100 poligona, cilj

Algoritam se jako dobro ponasa kod crno belih slika jer one generalno imaju manje detalja na sebi i radi se sa malim brojem dominantih boja.

Slika 6: v8 100 poligona, cilj

6 Moguce modifikacije i uticaj ulaznih parametara na rad algoritma

Ukoliko zelimo da unapredimo verziju algoritma sa Euklidskim rastojanjem jedan od mogucih nacina jeste racunanje fitness-a koriscenjem niti. Verzija 6 (*v6.py*) je pokusaj ove modifikacije ali ne uspesan.
Paralelizacija citavog algoritma je takodje moguca modifikacija koja ce sasvim sigurno doprineti efikasnosti izvrsavanja.
Parametri koji najvise uticu na efikasnost izvrsavanja jesu dimenzije slike, kao i sama slika (njena kompleksnost) i izbor metode racunanja fitness-a. Broj poligona sa kojim vrsimo iscrtavanja nema veliki uticaj na efikasnost.
Povecanjem broja aproksimacija sa 2 na 10 algoritam postaje drasticno sporiji, sto mozemo videti u verziji 7 (*v7.py*). Ovo je posledica nacina implementacije resenja.

7 Zakljucak

Ne postoji jedan pravi nacin na koji mozemo da resimo ovaj problem koji predstavlja odlican i zanimljiv izazov za bilo koga ko zeli da testira svoje

programerske vestine u oblasti racunarske inteligencije.

8 Literatura i reference

1. https://rogerjohansson.blog/2008/12/07/genetic-programming-evolution-of-mona-lisa/
2. https://stackoverflow.com/questions/3241929/python-find-dominant-most-common-color-in-an-image
3. https://github.com/GiriB/EvoLisa
4. https://scikit-image.org/docs/dev/auto_examples/transform/plot_ssim.html