Primena genetskog algoritma u kompresiji slika

Projekat na kursu „Naučno izračunavanje“

Matematički fakultet, Univerzitet u Beogradu

Nemanja Antić, 1100/2017

Filip Lazić, 1101/2017

Profesor: dr. Mladen Nikolić

Asistent: dr. Stefan Mišković

Uvod

- Genetski algoritam kao algoritam pretrage inspirisan procesom prirodne selekcije koji svojim radom akumulira znanje o prostoru pretrage kako bi došao do opšteg optimalnog rešenja
- Inicijalizacija
- Fitness funkcija
- Selekcija
- Crossover
- Mutacija

```
START
Generate the initial population
Compute fitness
REPEAT
    Selection
    Crossover
    Mutation
    Compute fitness
UNTIL population has converged
STOP
```

SSGA - Steady–state Genetic Algorithm

Genetski algoritam za kompresiju slika.

1. Ulazni parametri

- Veličina populacije - 65
- Maksimalni broj generacije - 500
- Dužina hromozoma (veličina bloka) - 256
- Blokovi piksela (m x n) - 4 x 4
- Verovatnoća mutacije - 0.1
- Datoteka koju treba kompresovati

```
img = cv2.imread("img.bmp")
img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

n = 4
m = 4
codebook_size = 256
pop_size = 65
max_gen = 500
mutation_prob = 0.1
```

2. Deljenje slike u blokove

- Delimo sliku u blokove piksela (n x m)
- Povećavanjem blokova dobijamo veći stepen kompresije, ali gubimo na kvalitetu i obrnuto.

```
def divide_image(img, n, m):
    image_vectors = []
    for i in range(0, img.shape[0] - n + 1, n):
        for j in range(0, img.shape[1] -m + 1, m):
            image_vectors.append(img[i : i + m, j : j + m])
    image_vectors = np.asarray(image_vectors).astype(int)
    return image_vectors
```

3. Generisanje početnog „codebook“-a

- Generišemo „codebook“ tako što uzmemo **codebook_size** nasumično izabranih blokova piksela. Neka je to skup C.

```
def build_codebook(codebook_size, vector_dimension, image_vectors):
    codebook = np.zeros((codebook_size, vector_dimension)).astype(int)
    for i in range(0, codebook_size):
        m = np.random.randint(0, image_vectors.shape[0])
        codebook[i] = image_vectors[m].reshape(1, vector_dimension)

    return codebook
```

- Za svaki blok (n x m) piksela nalazimo najbliži element iz C i klasifikujemo ga na osnovu toga. Tada za svaki element iz C imamo klasu najbližih image blokova.

```
def image_block_classifier(image_vectors, codebook):
    index_vector = np.zeros((image_vectors.shape[0],1)).astype(int)
    for i in range(0, image_vectors.shape[0]):
        hpsnr = 0
        best_unit = 0
        for j in range(0, codebook.shape[0]):
            tpsnr = psnr(image_vectors[i].reshape(1, codebook.shape[1]), codebook[j])
            if (tpsnr > hpsnr):
                best_unit = j
                hpsnr = tpsnr
        index_vector[i] = best_unit
    return index_vector
```

- Funkcija koja odredjuje bliskost:

$$PSNR = 10 \log_{10}\left(\frac{255^2}{MSE}\right) \quad MSE = \frac{1}{k \times k}\sum_{i=1}^{p}\sum_{j=1}^{p}(X_{i,j} - Y_{i,j})^2$$

```
def psnr(x, y):
    mse = np.square(x - y).mean()
    if mse == 0:
        return -1
    return int (10 * np.log10(255 * 255 / mse))
```

4. Inicijalizacija populacije

- SSGA algoritam pokušava da nađe blok u svakoj klasi koji će najbolje prezentovati sve blokove iz klase.
- Za svaki blok se kreira populacija uzimajući nasumične blokove iz klase
- Hromozomi su blokovi (n x m)
- Geni su vrednosti piksela u bloku

```python
def initialize_population(codebook, pop_size, image_vectors, chromosom_size, index_vector, unit):
    population = np.zeros((pop_size, chromosom_size)).astype(int)
    all_blocks = np.asarray(index_vector == unit).nonzero()[0]
    if (all_blocks.shape[0] == 0):
        return  population
    for i in range(0, pop_size):
        for j in range(0, chromosom_size):
            m = np.random.randint(0,all_blocks.shape[0])
            n_r = all_blocks[m]
            vector = image_vectors[n_r].reshape(1, chromosom_size)
            population[i, j] = vector[0][j]

    return population
```

Funkcija prilagođenosti *(eng. Fitness function)*

- Izračunava se za svaki hromozom u populaciji. Hromozom sa najboljom vrednošću je ciljani blok koji najbolje prezentuje blokove iz klase

```
def fitness_calculation(pop_size, population, chromosom_size, diff_pixel):
    fitness = []
    for i in range(0, pop_size):
        counter = 0
        for j in range(0, pop_size):
            if (i == j):
                continue
            for k in range(0, chromosom_size):
                if diff_pixel[i][k] < diff_pixel[j][k]:
                    counter = counter + 1
        fitness.append(counter / chromosom_size)

    fitness =  np.asarray(fitness).astype(float)
    return fitness
```

Funkcija diff-pixel

- Za svaki hromozom nam je potreban i niz koji sadrži kulmulativne razlike piksela bloka u odnosu na druge blokove iz klase
- Ova funkcija nam služi za računanje fitness funkcije

```
def calculate_diff_pixels(image_vectors, index_vector, population, pop_size, chromosom_size, unit):
    diff_pixels = np.zeros((pop_size, chromosom_size))
    arr = np.asarray(index_vector == unit).nonzero()[0]
    for i in range(0, pop_size):
        for j in range(0, chromosom_size):
            value = 0
            for k in range(0, arr.shape[0]):
                ind = arr[k]
                vector = image_vectors[ind].reshape(1, chromosom_size)
                value = value + abs(population[i, j] - vector[0][j])
            diff_pixels[i, j] = value
    return diff_pixels
```

5. Selekcija

- Koristimo ruletsku selekciju

```
def selection(fitnes_values):
    fitness_sum = np.sum(fitnes_values)
    if fitness_sum == 0:
        return [0, 0]
    fitness_prob = fitnes_values / fitness_sum
    return np.random.choice(fitness_values.shape[0], 2,  p=fitness_prob)
```

6. Ukrštanje *(eng. „crossover“)*

- Odabere se slučajan index n unutar 2 roditeljska hromozoma
- Jedno dete dobija prvih n gena od prvog roditelja i poslednjih n od drugog roditelja
- Drugo dete dobija prvih n gena od drugog roditelja i poslednjih n od prvog roditelja

```
def crossover(parent1, parent2, codebook_size):
    n = np.random.randint(0, codebook_size)
    children1 = parent1
    children2 = parent2
    children1[ : n] = parent1[ : n]
    children1[n : ] = parent2[n :]
    children2[ : n] = parent2[ : n]
    children2[n : ] = parent1[n : ]
    return children1, children2
```

7. Mutacija

- Mutacija koju koristimo izvršava se tako što nasumični gen zamenimo uprosečenom vrednošću celog hromozoma

```
def mutation(p, chromosom):
    p_p = np.random.rand()
    if p_p > p:
        m = np.random.randint(0, 16)
        chromosom[m] = np.average(chromosom)

    return chromosom
```

8. Kriterijum zaustavljanja

- **Dva slučaja kada se algoritam zaustavlja:**

1. Maksimalan broj generacija
2. Ako vrednost fitness funkcije ne menja značajno (razlika izmedju stare i nove je manja od neke zadate vrednosti)

```
def check_termination_criterion(generation, max_gen, sum_fit, old_fitness, eps):
    if generation >= max_gen:
        return True
    new_average_fit = sum_fit / generation
    old_average_fit = old_fitness / (generation - 1)
    if abs(new_average_fit - old_average_fit) < eps:
        return True
    return False
```

9. Generisanje slike iz „codebook“-a

```
def decode_codebook(codebook, codebook_size, index_vector,  n, m):
    img = np.zeros((512, 512))
    i = 0
    j = 0
    for k in range (0, index_vector.shape[0]):
        index = index_vector[k]
        img[i : i + n, j : j + m] = codebook[index].reshape(4, 4)
        j = j + m
        if (j == 512):
            j = 0
            i = i + n
    return img
```

Implementacija celog algoritma

```python
def check_termination_criterion(generation, max_gen, sum_fit, old_fitness, eps):
    if generation >= max_gen:
        return True
    new_average_fit = sum_fit / generation
    old_average_fit = old_fitness / (generation - 1)
    if abs(new_average_fit - old_average_fit) < eps:
        return True
    return False

image_vectors = divide_image(img, n, m)
codebook = build_codebook(codebook_size, n * m, image_vectors)
index_vector = image_block_classifier(image_vectors, codebook)
for i in range(0, codebook_size):
    generation = 0
    termination_criteria = False
    old_fitness = 0
    sumFit = 0
    best_chromosom = np.zeros((1, n * m), dtype=int)
    population = initialize_population(codebook, pop_size, image_vectors, n * m, index_vector, i)
    diff_pixels = calculate_diff_pixels(image_vectors, index_vector,population, pop_size, n * m, i)
    fitness_values = fitness_calculation(pop_size, population, n * m, diff_pixels)

    while termination_criteria == False:
        parent1, parent2 = selection(fitness_values)
        children1, children2 = crossover(population[parent1], population[parent2],n * m)
        chromosom1 = mutation(mutation_prob, children1)
        chromosom2 = mutation(mutation_prob, children2)
        diff_pixels_offspring = calculate_diff_pixels(image_vectors, index_vector, np.vstack((chromosom1, chromosom2)), 2, n * m, i)
        fitness_values_offspring = fitness_calculation(2, np.vstack((chromosom1, chromosom2)), n * m, diff_pixels)
        first = np.random.randint(0, pop_size)
        second = np.random.randint(0, pop_size)
        third = np.random.randint(0, pop_size)
```

```
            min_index = first
            if fitness_values[first] > fitness_values[second]:
                min_index = second
            if fitness_values[min_index] > fitness_values[third]:
                min_index = third
            if fitness_values[min_index] < np.amax(fitness_values_offspring):
                arg = np.argmax(fitness_values_offspring)
                if arg == 0:
                    population[min_index] = chromosom1
                else :
                    population[min_index] = chromosom2

            sumFit = sumFit + np.amax(fitness_values_offspring)
            generation = generation + 1
            if generation == 1:
                old_fitness = sumFit
                continue
            termination_criteria = check_termination_criterion(generation, max_gen, sumFit, old_fitness, 0.01)
            old_fitness = sumFit

        diff_pixels = calculate_diff_pixels(image_vectors, index_vector,population, pop_size, n * m, i)
        fitness_values = fitness_calculation(pop_size, population, n * m, diff_pixels)
        best_chromosom_index = np.argmax(fitness_values)
        best_chromosom = population[best_chromosom_index]
        codebook[i] = best_chromosom

    img = decode_codebook(codebook, codebook_size, index_vector, n, m)
    cv2.imwrite("compressed.bmp", img)
```

Rezultat