Sprawozdanie

Algorytmy genetyczne
Problem 3SAT

Matsvei Pilkou

1. Opis kodu:
   W tym rozdziale będą opisane główne funkcje i ich działanie

```
def read_and_create_dv_and_set(path):
    temp_df = pd.read_csv(path)

    M = {"A":[],"B":[],"C":[]}
    mySet = set()

    for i in range(7):
        if (temp_df.iloc[0][0][0] == 'c' or temp_df.iloc[0][0][0] == 'p'):
            temp_df = temp_df.drop(temp_df.index[0])

    for i in range(2):
        if (temp_df.iloc[-1][0][0] == '0' or temp_df.iloc[-1][0][0] == '%'):
            temp_df = temp_df.drop(temp_df.index[-1])

    temp_df = temp_df.rename(index=str, columns={temp_df.columns[0]: "col"})
    temp_df = temp_df['col'].str.split(' ')
    if(temp_df[0][0] == ''):
        temp_df[0].remove('')

    def devideData(x):
        M["A"].append(int(x[0]))
        mySet.add(abs(int(x[0])))
        M["B"].append(int(x[1]))
        mySet.add(abs(int(x[1])))
        M["C"].append(int(x[2]))
        mySet.add(abs(int(x[2])))
        return True

    sum(temp_df.apply(lambda x : devideData(x)))

    return pd.DataFrame(M),(len(mySet))
```

funkcja read_and_create_dv_and_set

Ta funkcja jest przeznaczona głównie dla stworzenia „data frejmu„ z danych tekstowych i usunięciu niepotrzebnych znaków i obliczeniu zmiennych w podanym pliku.

```python
def crossover(parent_1, parent_2):
    index = random.randrange(1, len(parent_1))
    child_1 = parent_1[:index] + parent_2[index:]
    child_2 = parent_2[:index] + parent_1[index:]
    return child_1, child_2

def mutate(individual):
    mutate_index = random.randrange(len(individual))
    if individual[mutate_index] == 0:
        individual[mutate_index] == 1
    else:
        individual[mutate_index] == 0

def selection(population):
    return random.choice(population)
```

funkcje pomocnicze

Na powyższym rysunku zostali przedstawione funkcje odpowiadające za krzyżowanie, mutacje i wybieranie chromosomów.

```python
def evaluation(individual, x):

    temp_individual = individual.copy()
    V = 0
    for i in range(3):
        j = (abs(x[x.index[i]]) - 1)
        if x[x.index[i]] < 0:
            temp_individual[j] = abs(temp_individual[j] - 1)
        V += temp_individual[j]

    return V>0

def fitness (individual, data):
    return sum(data.apply(lambda x: evaluation(individual,x),axis=1))
```

funkcje fitness i evaluation

Funkcja fitness sumuje ile jest zrobionych poprawnie wyrażeń logicznych (z pliku) i wystawia ocenę. Wspomagająca funkcja *evaluation* wystawia ocenę (prawda lub fałsz) za każdy poprawnie zrobiony przykład (wiersz).

```
def start_task(path,population_size=50,
               generations = 100,mutation_probability=0.05):

    df,n = read_and_create_dv_and_set(path)

    def create_individual(data):
            return [random.randint(0, 1) for _ in range(n)]

    genAlg = GeneticAlgorithm(
        df,
        population_size=population_size,generations=generations,
        mutation_probability=mutation_probability,
        elitism=True,maximise_fitness=True)
    genAlg.create_individual = create_individual
    genAlg.crossover_function = crossover
    genAlg.mutate_function = mutate
    genAlg.selection_function = selection
    genAlg.fitness_function = fitness
    start = time.time()
    genAlg.run()
    end = time.time()
    return end - start
```

Funkcja startująca

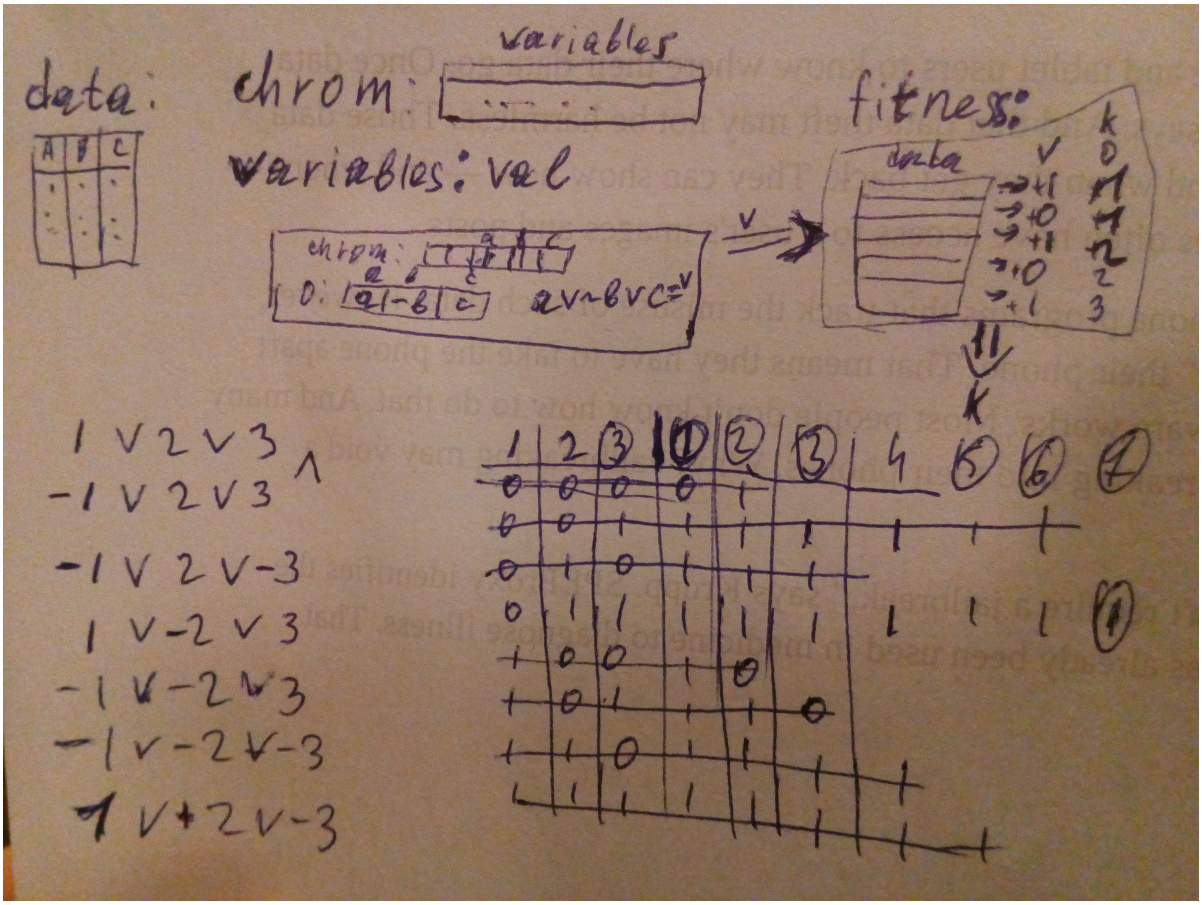
Funkcja *start_task* odpowiada za działanie algorytmu i zwraca czas działania algorytmu, dla podanych wartości.

```
fig, ax = plt.subplots(figsize=(12,6))
ax.plot(x, y, color="red", lw=2,ls='--',marker='o',
        label="population_size=50 & generations = 100 & mutation_probability=0.05")
ax.plot(x, z, color="red", lw=1,ls='--',marker='s',
        label="population_size=50 & generations = 100 & mutation_probability=0.1")
ax.plot(x, y1, color="black", lw=2,ls=':',marker='o',
        label="population_size=100 & generations = 150 & mutation_probability=0.05")
ax.plot(x, z1, color="black", lw=1,ls=':',marker='s',
        label="population_size=100 & generations = 150 & mutation_probability=0.1")
ax.plot(x, y2, color="green", lw=2,ls='-.',marker='o',
        label="population_size=50 & generations = 150 & mutation_probability=0.05")
ax.plot(x, y3, color="blue", lw=2,ls='--',marker='o',
        label="population_size=100 & generations = 100 & mutation_probability=0.05")
ax.legend(loc=0)
ax.set_xlabel('wielkosc problemu')
ax.set_ylabel('czas dzilania')
ax.set_title('Wykres')
```

Funkcja rysująca

Funkcja rysująca rysuje wykres z danych odliczonych przez algorytm.

2. Działanie algorytmu



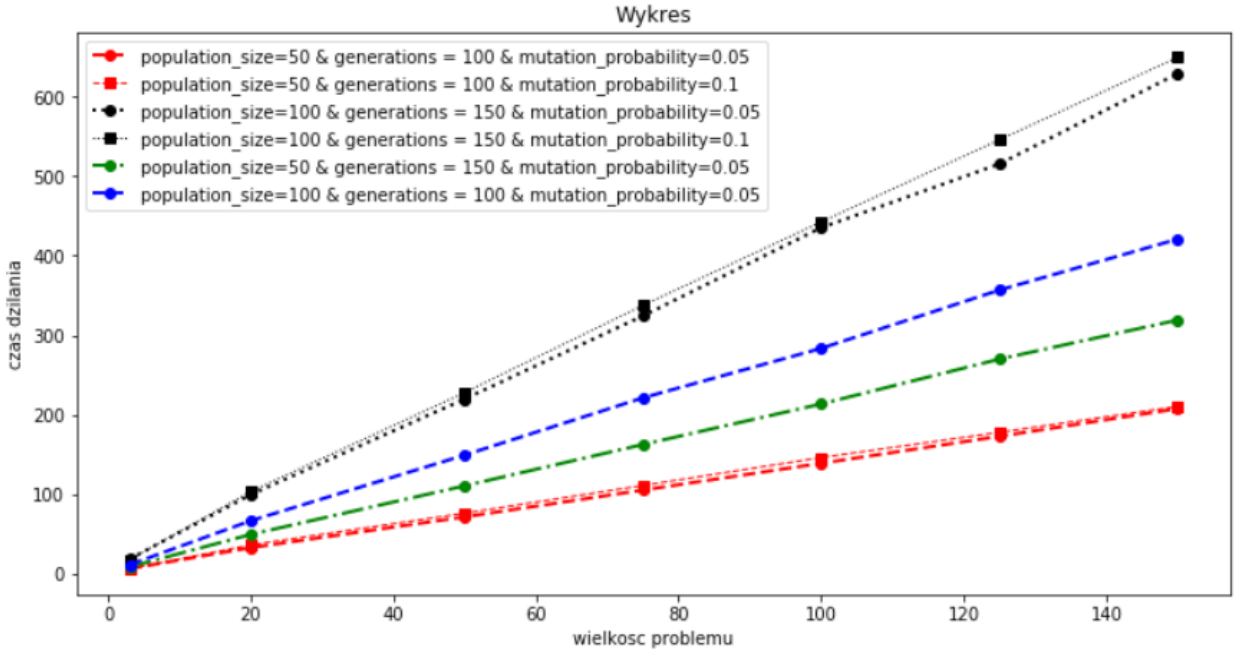
przykładowy opis działania algorytmu

Na obrazku jest pokazany opis działania algorytmu na przykładzie zawartym w pliku „uf_test.cnf”

3. Przykład działania (dla przykładu na obrazku powyżej)

Krok	Chromosom (len = ilość zmiennych)	Ocena (max 7)
0	[0,0,0]	6 (0+1+1+1+1+1+1)
	[0,0,0] ^ | | [1,0,0]	
1	[1,0,0]	6(1+0+1+1+1+1+1)
	[1,0,0] | ^ ^ [1,0,1]	
2	[1,0,1]	6(1+1+0+1+1+1+1)
	[1,0,1] | M ^ [0,0,1] M - mutacja	
3	[0,1,1]	7(1+1+1+1+1+1+1)

4. Wykres



Wykres algorytmu

Wykres ilustruje zależność czasu działania (w sekundach) i wielkości problemu (ilości zmiennych). Także różnymi kolorami są zaznaczone wykresy algorytmu przy różnych danych parametrycznych algorytmu.

5.