Praca domowa 7
Porównanie algorytmu roju i genetycznego

Damian Jankowski s188597

1 czerwca 2023

1 Wstęp

Celem pracy domowej było porównanie algorytmu roju cząstek (PSO) oraz algorytmu genetycznego (GA) w wybranym problemie.

Zdecydowałem się na problem plecakowy, który polega na wybraniu zbioru przedmiotów o określonej wartości i wadze, tak aby maksymalizować wartość przedmiotów, a jednocześnie nie przekroczyć określonej wagi plecaka.

2 Opis algorytmów

2.1 Algorytm roju cząstek

Algorytm roju cząstek (Particle Swarm Optimization, PSO) jest metaheurystyką inspirowaną zachowaniem stad i rojów w naturze. Algorytm składa się z populacji cząstek, gdzie każda cząstka reprezentuje jedno potencjalne rozwiązanie problemu. Każda cząstka porusza się w przestrzeni rozwiązań, posiadając swoją pozycję oraz prędkość.

Algorytm PSO opiera się na iteracyjnym przemieszczaniu cząstek w poszukiwaniu optymalnego rozwiązania. Cząstki zmieniają swoje pozycje i prędkości w oparciu o najlepsze znalezione dotychczas rozwiązanie dla danej cząstki (lokalne najlepsze rozwiązanie) oraz najlepsze znalezione rozwiązanie wśród wszystkich cząstek (globalne najlepsze rozwiązanie).

Podczas iteracji, cząstki poruszają się w przestrzeni rozwiązań, aktualizując swoje pozycje i prędkości na podstawie zdefiniowanych równań.

W kontekście problemu plecakowego, algorytm PSO poszukuje kombinacji przedmiotów optymalnie umieszczonych w plecaku, tak aby maksymalizować wartość przedmiotów, jednocześnie nie przekraczając określonej wagi plecaka. Cząstki w tym przypadku reprezentują różne kombinacje produktów, a optymalne rozwiązanie polega na znalezieniu cząstki, której kombinacja przedmiotów ma najwyższą wartość.

2.2 Algorytm genetyczny

Algorytm genetyczny (Genetic Algorithm, GA) jest metaheurystyką inspirowaną procesem ewolucji biologicznej. Algorytm operuje na populacji rozwiązań, które są reprezentowane przez struktury zwane chromosomami. Chromosomy składają się z genów, które zawierają informacje o cechach rozwiązania.

Algorytm genetyczny operuje na populacji, gdzie każde rozwiązanie jest kodowane jako chromosom. Algorytm składa się z kilku kroków, takich jak selekcja, krzyżowanie, mutacja i ewaluacja.

Selekcja polega na wyborze najlepszych rozwiązań z populacji na podstawie funkcji celu. Następnie, wybrane rozwiązania są krzyżowane, co polega na wymianie części informacji genetycznej pomiędzy dwoma chromosomami, aby stworzyć potomstwo. Krzyżowanie ma na celu zwiększenie różnorodności populacji i wprowadzenie nowych kombinacji genetycznych.

Mutacja polega na losowej zmianie pewnych genów w chromosomie, co pomaga w eksploracji nowych obszarów przestrzeni poszukiwań. Po krzyżowaniu i mutacji, oceniana jest jakość nowo powstałego potomstwa za pomocą funkcji celu. Jeśli potomstwo ma lepszą jakość niż pewne osobniki w populacji, zastępuje je.

Algorytm podobnie jak algorytm roju cząstek kontynuuje iteracje, aż do osiągnięcia zadowalającego rozwiązania lub spełnienia warunku stopu.

3 Porównanie

By porównać działanie algorytmów zaimplementowałem najpierw generator instancji problemu plecakowego, który tworzy 20 przedmiotów o losowej wadzie i wartości. Następnie zaimplementowałem algorytm roju cząstek oraz algorytm genetyczny, które rozwiązują ten problem w czasie 100 iteracji.

Dla sprawdzenia działania algorytmów, zaimplementowałem również algorytm brute-force, który sprawdza wszystkie możliwe kombinacje przedmiotów i wybiera tą, która ma największą wartość i mieści się w plecaku.

3.1 Wyniki

Dla następującego plecaka:

```
Plecak o pojemnosci: 2000
Produkty:
Nazwa Wartosc Waga
Produkt 1 296 347
Produkt 2 962 210
Produkt 3 663 153
Produkt 4 676 179
Produkt 5 484 466
Produkt 6 706 146
Produkt 7 261 453
Produkt 8 163 486
Produkt 9 752 228
Produkt 10 878 394
Produkt 11 987 427
Produkt 12 828 394
Produkt 13 333 370
Produkt 14 797 446
Produkt 15 515 382
Produkt 16 669 146
Produkt 17 328 329
Produkt 18 335 112
Produkt 19 379 436
Produkt 20 210 327
```

3.2 Brute force

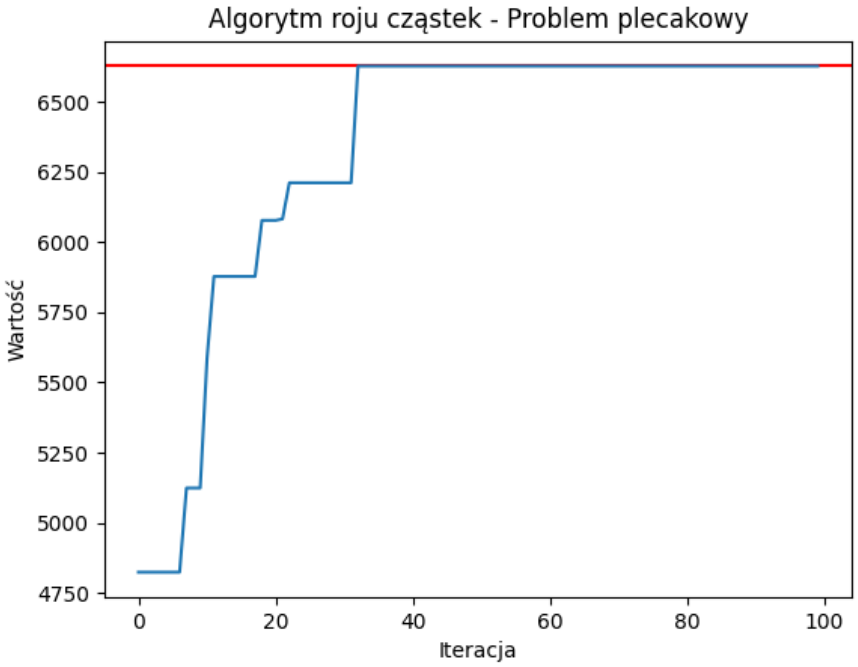
Algorytm brute-force znalazł następujące rozwiązanie:

```
Brute force:
Najlepsza wartosc: 6628
Najlepsza pozycja: [0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0]
```

3.3 Algorytm roju cząstek

Algorytm roju cząstek znalazł następujące rozwiązanie:

```
Algorytm roju czastek:
Najlepsza wartosc: 6628
Najlepsza pozycja: [0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0]
```

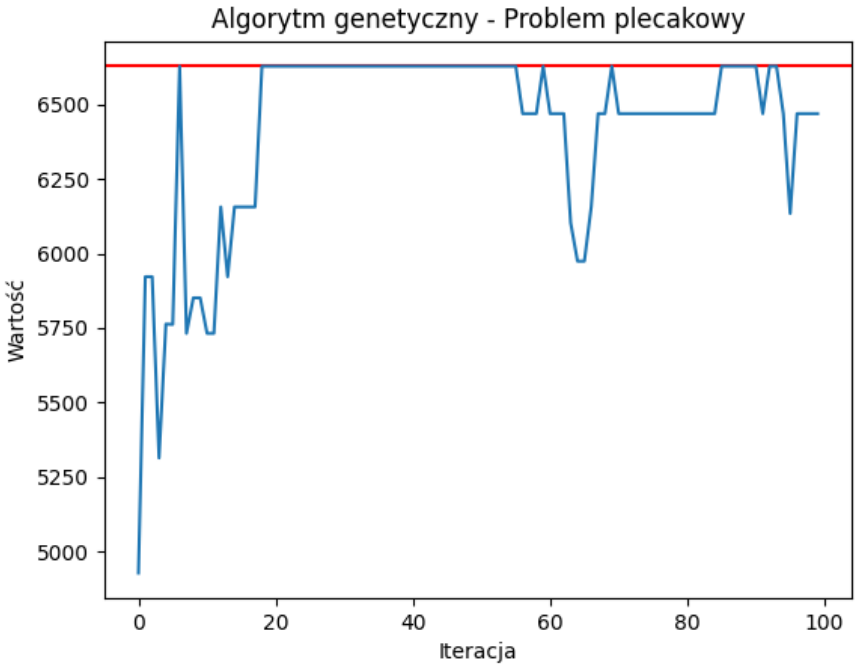


Rysunek 1: Wykres wartości plecaka w zależności od iteracji dla algorytmu roju cząstek

3.4 Algorytm genetyczny

Natomiast algorytm genetyczny znalazł następujące rozwiązanie:

```
Algorytm genetyczny:
Najlepsza wartosc: 6628
Najlepsza pozycja: [0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0]
```



Rysunek 2: Wykres wartości plecaka w zależności od iteracji dla algorytmu genetycznego

4 Wnioski

W przypadku algorytmu roju cząstek, wartość plecaka dąży do wartości optymalnej mniej chaotycznie niż w przypadku algorytmu genetycznego. Po osiągnięciu niej wartości plecaka nie zmieniają się już i pozostają na stałym poziomie.

Algorytm genetyczny osiąga wartość optymalną szybciej niż algorytm roju cząstek, jednakże nie jest w stanie jej utrzymać i wartość plecaka zaczyna oscylować.

Wybór algorytmu zależy od tego, czy zależy nam na szybkim, choć nie pewnym znalezieniu rozwiązania czy na stabilnym dążeniu do niego. W przypadku algorytmu roju (PSO) dzięki ciągłej aktualizacji pozycji cząstek, algorytm jest w stanie znaleźć globalne optimum. Wadą takiego rozwiązania jest możliwość wymogu większej liczby iteracji, aby znaleźć rozwiązanie.

Algorytm genetyczny (GA) jest w stanie znaleźć rozwiązanie szybciej niż PSO oraz dzięki mechanizmom selekcji, krzyżowania i mutacji, eksplorować różne kombinacje populacji. Natomiast może być podatny na utknięcie w lokalnym optimum oraz być skłonny do oscylacji wokół wartości optymalnej.

5 Kod programu

```
import random
import math

from matplotlib import pyplot as plt

# Parametry problemu
plecak_waga = 0
produkty = []

najlepsze_rozwiazanie = 0


def wygeneruj_plecak():
    global plecak_waga
    plecak_waga = 2000

    global produkty
    for i in range(20):
        produkty.append({
            "nazwa": "Produkt " + str(i + 1),
            "wartosc": random.randint(100, 1000),
            "waga": random.randint(100, 500)
        })

    wypisz_produkty()


def wypisz_produkty():
    print("Plecak o pojemnosci: ", plecak_waga)
    print("Produkty:")
    print("Nazwa\t\tWartosc\tWaga")
    for produkt in produkty:
        print(produkt["nazwa"], "\t", produkt["wartosc"], "\t", produkt["waga"])


def wypisz_plecak(najlepsza_wartosc, najlepsza_pozycja):
    print("Najlepsza wartosc: ", najlepsza_wartosc)
    print("Najlepsza pozycja: ", najlepsza_pozycja)


def oblicz_wartosc_plecaka(pozycja):
    # Obliczanie wartości dla danej pozycji
    wartosc = sum([produkty[i]["wartosc"] for i, gen in enumerate(pozycja) if gen])
    waga = sum([produkty[i]["waga"] for i, gen in enumerate(pozycja) if gen])
```

```
    return wartosc if waga <= plecak_waga else 0


def sprawdz_brute_force():
    najlepsza_wartosc = 0
    najlepsza_pozycja = []
    for i in range(2 ** len(produkty)):
        pozycja = [int(x) for x in bin(i)[2:]]
        pozycja = [0] * (len(produkty) - len(pozycja)) + pozycja
        wartosc = oblicz_wartosc_plecaka(pozycja)
        if wartosc > najlepsza_wartosc:
            najlepsza_wartosc = wartosc
            najlepsza_pozycja = pozycja

    global najlepsze_rozwiazanie
    najlepsze_rozwiazanie = najlepsza_wartosc

    wypisz_plecak(najlepsza_wartosc, najlepsza_pozycja)


def sigmoid(x):
    # Funkcja sigmoidalna
    return 1 / (1 + math.exp(-x))


def algorytm_roju():
    # Parametry PSO
    liczba_czastek = 100
    liczba_iteracji = 100
    wsp_roju_1 = 0.5
    wsp_roju_2 = 0.5

    class Czastka:
        def __init__(self):
            # Inicjalizacja pozycji i predkości cząstki
            self.pozycja = [random.randint(0, 1) for _ in range(len(produkty))]
            self.predkosc = [random.uniform(0, 1) for _ in range(len(produkty))]
            self.najlepsza_pozycja = self.pozycja.copy()

        def oblicz_wartosc(self):
            # Obliczanie wartości dla aktualnej pozycji cząstki
            return oblicz_wartosc_plecaka(self.pozycja)

    class RojCzastek:
        def __init__(self):
            self.czastki = [Czastka() for _ in range(liczba_czastek)]
            self.najlepsza_pozycja_globalna = self.czastki[0].pozycja.copy()

        def aktualizuj_najlepsza_pozycje_globalna(self):
            # Aktualizacja najlepszej pozycji globalnej dla roju
            for czastka in self.czastki:
                if czastka.oblicz_wartosc() > oblicz_wartosc_plecaka(self.
                    ↪ najlepsza_pozycja_globalna):
                    self.najlepsza_pozycja_globalna = czastka.pozycja.copy()

        def aktualizuj_czastki(self):
            # Aktualizacja pozycji i predkości dla każdej cząstki
            for czastka in self.czastki:
                for i, gen in enumerate(czastka.pozycja):
                    predkosc_czastki = czastka.predkosc[i]
                    r1 = random.random()
                    r2 = random.random()
```

```
                # Aktualizacja predkości cząstki
                predkosc_czastki = predkosc_czastki + wsp_roju_1 * r1 * (
                        czastka.najlepsza_pozycja[i] - czastka.pozycja[i]) + wsp_roju_2 * r2 *
                            ↪ (
                                    self.najlepsza_pozycja_globalna[i] - czastka.pozycja[
                                        ↪ i])
                czastka.predkosc[i] = predkosc_czastki

                # Aktualizacja pozycji cząstki na podstawie predkości
                if random.random() < sigmoid(predkosc_czastki):
                    czastka.pozycja[i] = 1
                else:
                    czastka.pozycja[i] = 0

            if czastka.oblicz_wartosc() > oblicz_wartosc_plecaka(czastka.najlepsza_pozycja):
                czastka.najlepsza_pozycja = czastka.pozycja.copy()

    roj = RojCzastek()
    najlepsza_wartosc = 0
    najlepsza_pozycja = []
    wartosc_historia = []

    for _ in range(liczba_iteracji):
        roj.aktualizuj_najlepsza_pozycje_globalna()
        roj.aktualizuj_czastki()

        # Zapisz najlepszą pozycję globalną w każdej iteracji oraz jej wartość
        wartosc_historia.append(oblicz_wartosc_plecaka(roj.najlepsza_pozycja_globalna))

        # Zapisz najlepszą pozycję
        pozycja = [int(x) for x in roj.najlepsza_pozycja_globalna]
        pozycja = [0] * (len(produkty) - len(pozycja)) + pozycja

        wartosc = oblicz_wartosc_plecaka(roj.najlepsza_pozycja_globalna)

        if wartosc > najlepsza_wartosc:
            najlepsza_wartosc = wartosc
            najlepsza_pozycja = pozycja

    # Wykres
    plt.axhline(y=najlepsze_rozwiazanie, color='r', linestyle='-')
    plt.plot(range(liczba_iteracji), wartosc_historia)
    plt.xlabel('Iteracja')
    plt.ylabel('Wartość')
    plt.title('Algorytm␣roju␣cząstek␣-␣Problem␣plecakowy')
    plt.savefig('pso.png')
    plt.show()

    wypisz_plecak(najlepsza_wartosc, najlepsza_pozycja)


def algorytm_genetyczny():
    # Parametry algorytmu genetycznego
    liczba_osobnikow = 100
    liczba_iteracji = 100
    prawdopodobienstwo_mutacji = 0.2

    class Osobnik:
        def __init__(self):
            # Inicjalizacja pozycji osobnika
            self.pozycja = [random.randint(0, 1) for _ in range(len(produkty))]

        def oblicz_wartosc(self):
```

```
        # Obliczanie wartości dla aktualnej pozycji osobnika
        return oblicz_wartosc_plecaka(self.pozycja)

class Populacja:
    def __init__(self):
        self.osobniki = [Osobnik() for _ in range(liczba_osobnikow)]

    def selekcja(self):
        # Selekcja osobników
        suma_wartosci = sum([osobnik.oblicz_wartosc() for osobnik in self.osobniki])

        if suma_wartosci == 0:
            return

        prawdopodobienstwo = [osobnik.oblicz_wartosc() / suma_wartosci for osobnik in self.
            ↪ osobniki]

        # Wybór osobników do krzyżowania
        wybrani_osobnicy = random.choices(self.osobniki, prawdopodobienstwo, k=
            ↪ liczba_osobnikow)
        self.osobniki = wybrani_osobnicy

    def krzyzowanie(self):
        # Krzyżowanie osobników
        nowe_osobniki = []
        for _ in range(liczba_osobnikow // 2):
            rodzice = random.sample(self.osobniki, 2)
            punkt_podzialu = random.randint(1, len(produkty) - 1)
            nowe_osobniki.append(Osobnik())
            nowe_osobniki.append(Osobnik())
            nowe_osobniki[-2].pozycja = rodzice[0].pozycja[:punkt_podzialu] + rodzice[1].
                ↪ pozycja[punkt_podzialu:]
            nowe_osobniki[-1].pozycja = rodzice[1].pozycja[:punkt_podzialu] + rodzice[0].
                ↪ pozycja[punkt_podzialu:]

        self.osobniki += nowe_osobniki

    def mutacja(self):
        # Mutacja osobników
        for osobnik in self.osobniki:
            random_index = random.randint(0, len(osobnik.pozycja) - 1)
            if random.random() < prawdopodobienstwo_mutacji:
                osobnik.pozycja[random_index] = not osobnik.pozycja[random_index]

populacja = Populacja()
najlepsza_wartosc = 0
najlepsza_pozycja = []
wartosc_historia = []

for _ in range(liczba_iteracji):
    populacja.selekcja()
    populacja.krzyzowanie()
    populacja.mutacja()

    # Zapisz najlepszego osobnika w każdej iteracji oraz jego wartość
    najlepszy_osobnik = max(populacja.osobniki, key=lambda osobnik: osobnik.oblicz_wartosc()
        ↪ )
    wartosc_historia.append(najlepszy_osobnik.oblicz_wartosc())

    # Zapisz najlepszą pozycję
    pozycja = [int(x) for x in najlepszy_osobnik.pozycja]
    pozycja = [0] * (len(produkty) - len(pozycja)) + pozycja
```

```
        if najlepszy_osobnik.oblicz_wartosc() > najlepsza_wartosc:
            najlepsza_wartosc = najlepszy_osobnik.oblicz_wartosc()
            najlepsza_pozycja = pozycja

    # Wykres
    plt.axhline(y=najlepsze_rozwiazanie, color='r', linestyle='-')
    plt.plot(range(liczba_iteracji), wartosc_historia)
    plt.xlabel('Iteracja')
    plt.ylabel('Wartość')
    plt.title('Algorytm genetyczny - Problem plecakowy')
    plt.savefig('ga.png')
    plt.show()

    wypisz_plecak(najlepsza_wartosc, najlepsza_pozycja)


if __name__ == "__main__":
    wygeneruj_plecak()

    print("\nBrute force: ")
    sprawdz_brute_force()

    print("\nAlgorytm roju czastek: ")
    algorytm_roju()

    print("\nAlgorytm genetyczny: ")
    algorytm_genetyczny()
```