22 maja 2024

Politechnika Warszawska

Wydział Elektroniki i Technik Informacyjnych

Algorytmy Ewolucyjne

Zadanie projektowe #2

Wykonawca zadania
Małkowski Mateusz

Prowadzący projekt
dr inż. Grzegorz Bogdan

Małkowski Mateusz

Numer indeksu: 321358

Numer konta USOS: 1172008

Adres email: 01172008@pw.edu.pl

Spis treści

Cel zadania

Celem zadania było znalezienie rozwiązania problemu plecakowego za pomocą algorytmu genetycznego. Problem plecakowy polega na wybraniu zbioru przedmiotów o określonych wagach i wartościach, tak aby maksymalizować sumaryczną wartość przedmiotów w plecaku, przy jednoczesnym nieprzekroczeniu jego maksymalnej pojemności wagowej.

Parametry Algorytmu

- Liczba przedmiotów: 32
- Wagi przedmiotów: Losowane z rozkładu równomiernego z przedziału [0.1, 1.0] z dokładnością do 0.1
- Wartości przedmiotów: Losowane z rozkładu równomiernego z przedziału [1, 100] z dokładnością do 1
- Maksymalna waga plecaka: 30% sumy wag wszystkich przedmiotów

Kryteria doboru optymalnych parametrów

Warunek zatrzymania

Za warunek zatrzymania algorytmu wybrano brak poprawy wartości funkcji celu przez 50 generacji z progiem poprawy 10^{-3}. Pozwala on na wyjście algorytmu z minimów lokalnych i daje więcej czasu na znalezienie lepszego rozwiązania. Jest bardzo uniwersalny i zapewnia dobre wyważenie między efektywnością czasową a jakością znalezionego rozwiązania. Jednakże, odpowiednie dostrojenie parametrów tych kryteriów jest kluczowe dla optymalnego działania algorytmu i może wymagać dostosowania w zależności od specyfiki problemu. Jego wadą jest potencjalna nieefektywność w sytuacjach, gdy poprawa jest bardzo powolna.

Selekcja turniejowa

Selekcja turniejowa pozwala na zachowanie różnorodności dzięki losowości jednocześnie będąc wydajna obliczeniowo. Pozwala na łatwe dostosowanie presji selekcyjnej poprzez zmianę rozmiaru turnieju i dobrze zachowuje różnorodność genetyczną w populacji. Jednakże, zbyt duża presja selekcyjna i losowość mogą prowadzić do problemów z przedwczesną konwergencją i niestabilnością wyników.

Funkcja celu z karą

$$fitness = wartość - kara$$

Zastosowano funkcję celu z karą co zapewnia, że rozwiązania przekraczające maksymalną wagę są penalizowane, co skutecznie prowadzi algorytm do znajdowania rozwiązań zgodnych z ograniczeniami problemu. Dzięki jej zastosowaniu algorytm genetyczny może lepiej różnicować pomiędzy rozwiązaniami, które nie spełniają warunków. Nawet jeśli rozwiązanie ma wysoką wartość, jeżeli przekracza ono maksymalną wagę, kara sprawia, że jego fitness jest niższy. Dzięki

temu algorytm może skoncentrować swoje wysiłki na bardziej obiecujących obszarach przestrzeni rozwiązań. Jej wadą jest konieczność obliczenia kary dla każdego rozwiązania co lekko zwiększa złożoność obliczeniową.

Szansa mutacji

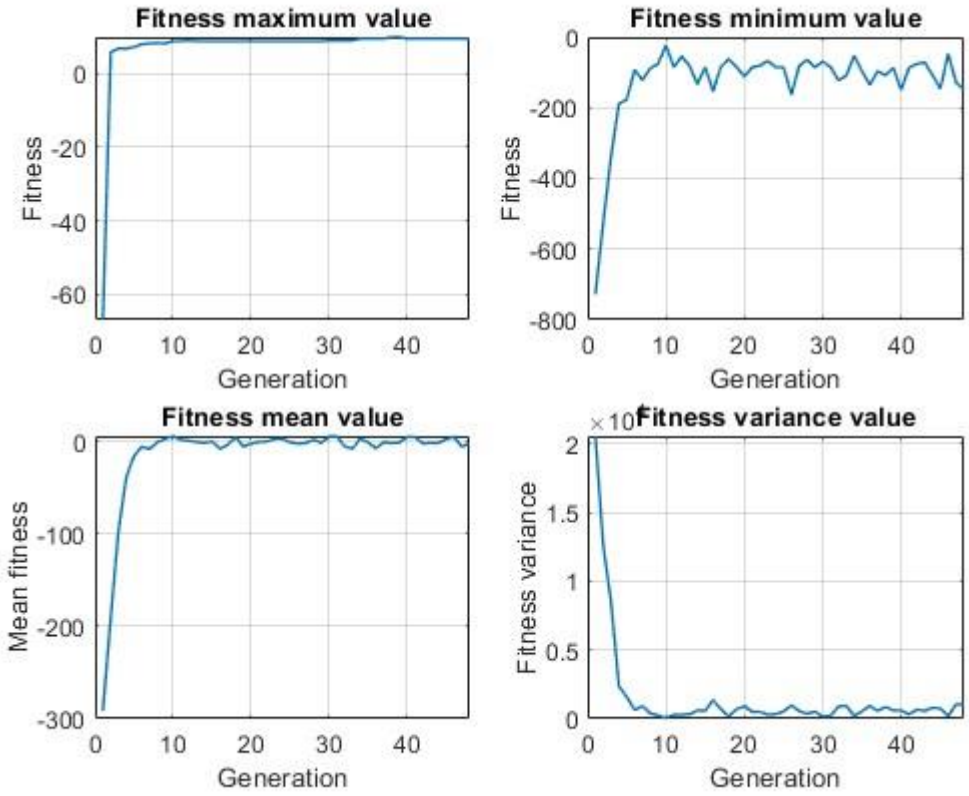
Najlepsza szansa mutacji dla stworzonego algorytmu wynosi 0.01. Przetestowano algorytm z różnymi jej wartościami i tylko dla niej algorytm nie zatrzymywał się w minimach lokalnych. Z wykresów przedstawionych na rysunkach 2, 3, 4 widać, że dla większego prawdopodobieństwa mutacji wartość średnia i wariancja nie zbliżały się do zera i miały większy rozstrzał.

Z tabel 2,3,4 widać, że dla szansy mutacji = 0.01 znaleziono największą wartość najlepszego rozwiązania. Dla większych jej wartości algorytm nie zachowywał wystarczająco dużo pozytywnych cech rodziców, by dojść do lepszych wyników. Przez zatrzymanie się algorytmu w minimum lokalnym dla większych szans mutacji algorytm kończył pracę po mniejszej ilości iteracji.

Krzyżowanie

Algorytm przetestowano z krzyżowaniem dwupunktowym i jednopunktowym. Każdorazowo algorytm miał 95% szansy na krzyżowanie i 5% na zostawienie potomków elitarnych, co pozwoliło na stworzenie zróżnicowanego potomstwa i zwiększenie znaczenia elity w populacji. Z tabeli 1 i 2 widać, że dla jednopunktowego algorytm znalazł większą wartość najlepszego rozwiązania. Jest też mniej wymagające obliczeniowo niż krzyżowanie dwupunktowe. Z rysunków 1 i 2 przedstawiających wykresy wartości funkcji dla obu krzyżowań widać, że rodzaj krzyżowania nie miał dużego wpływy na wartość średnią i wariancję.

Wyniki



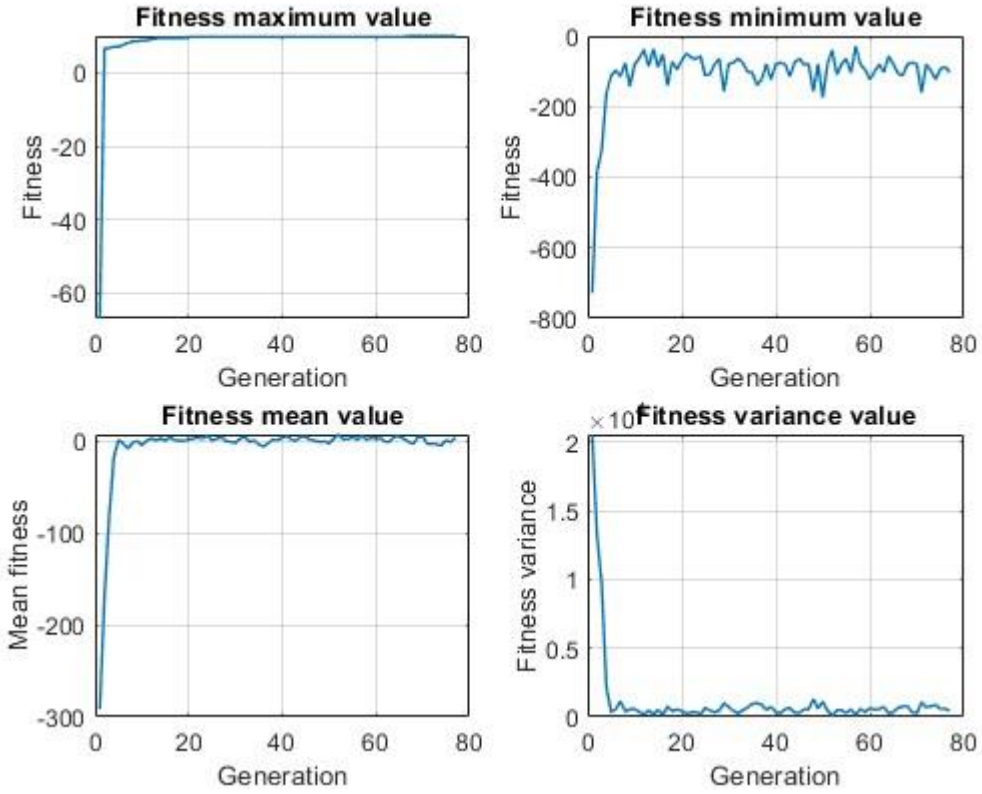
Rysunek 1:Wykresy wartości funkcji celu dla krzyżowania dwupunktowego z szansą mutacji 0.01

Szansa mutacji = 0.01	
Typ krzyzowania dwupunktowy	
Wartosc najlepszego rozwiazania	9.8
Waga najlepszego rozwiazania	446
Srednia liczba potomkow krzyzowania	47,6
Srednia liczba potomkow elitarnych	2,4
Srednia liczba potomkow zmutowanych	14
Liczba iteracji	88

Tabela 1:Wyniki dla krzyżowania dwupunktowego z szansą mutacji 0.01

1	0	1	1	0	1	0	0	0	0	1	1	1	0	1	1	1	0	0	0	1	0	1	1	0	1	0	0	0	0	0	1

Tabela 2:Najlepszy wynik dla krzyżowania dwupunktowego z szansą mutacji 0.01



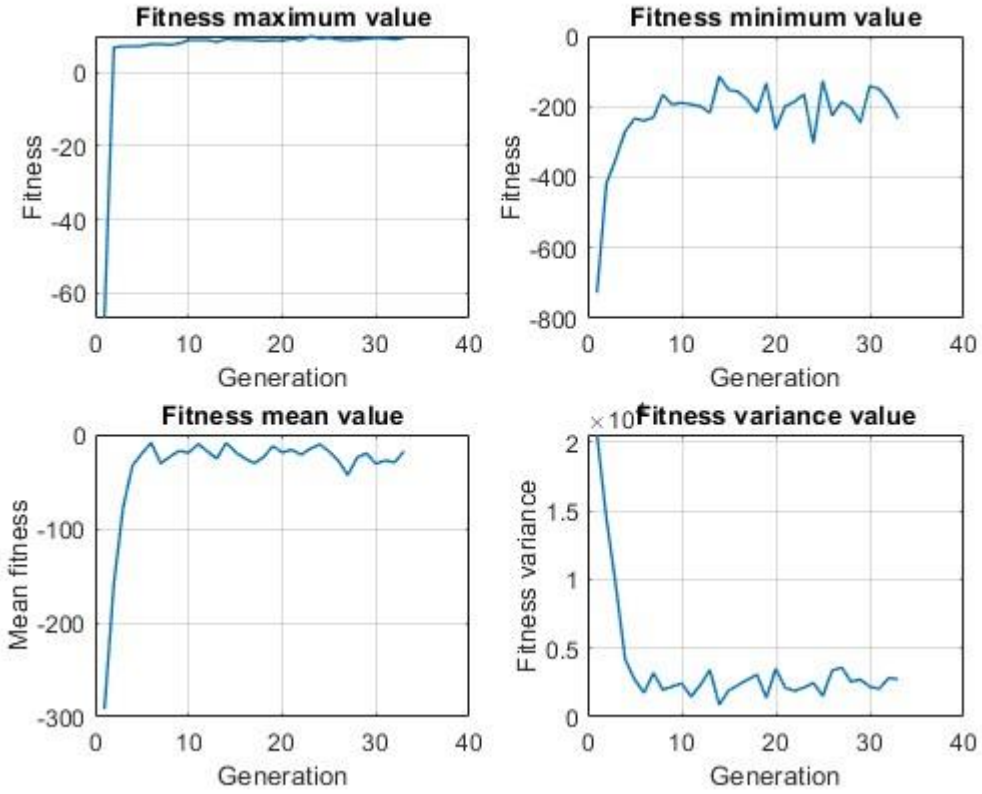
Rysunek 2:Wykresy wartości funkcji celu dla krzyżowania jednopunktowego z szansą mutacji 0.01

Szansa mutacji = 0.01	
krzyżowanie jednopunktowe	
Wartosc najlepszego rozwiazania	10.1
Waga najlepszego rozwiazania	446
Srednia liczba potomkow krzyzowania	47,7
Srednia liczba potomkow elitarnych	2,3
Srednia liczba potomkow zmutowanych	13,5
Liczba iteracji	117

Tabela 3:Wyniki dla krzyżowania jednopunktowego z szansą mutacji 0.01

1	1	1	1	0	1	0	0	0	0	0	1	1	0	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1

Tabela 4:Najlepszy wynik dla krzyżowania jednopunktowego z szansą mutacji 0.01



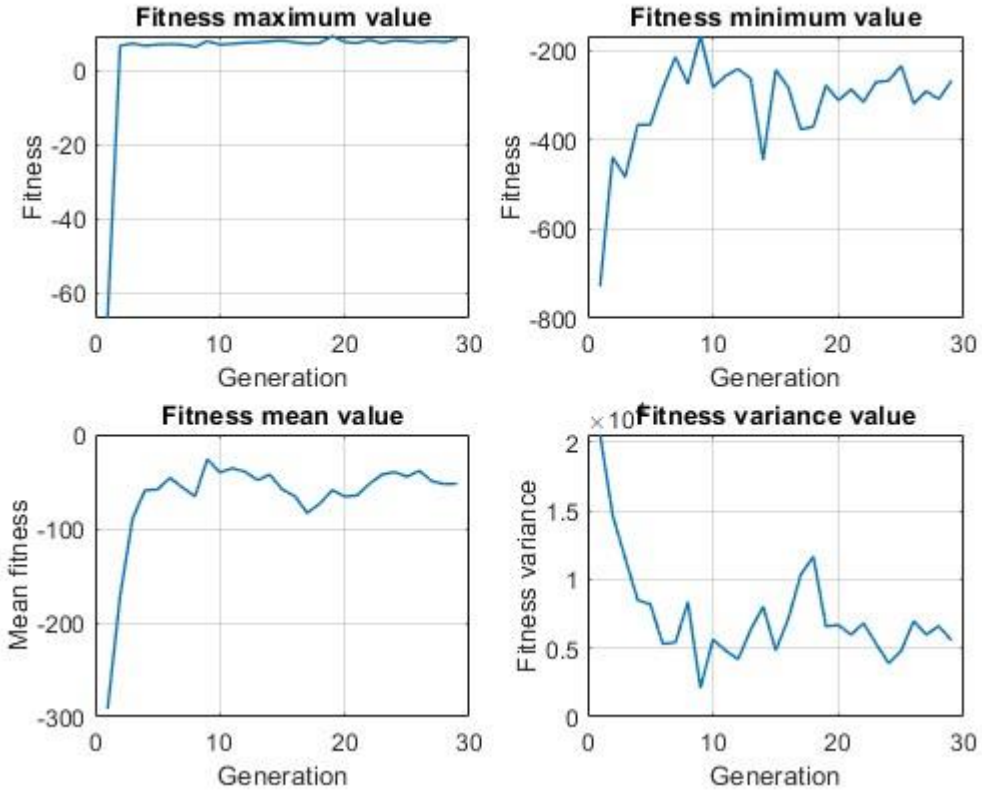
Rysunek 3:Wykresy wartości funkcji celu dla krzyżowania jednopunktowego z szansą mutacji 0.05

krzyżowanie jednopunktowe	
Szansa mutacji	0,05
Wartosc najlepszego rozwiazania	8.4
Waga najlepszego rozwiazania	432
Srednia liczba potomkow krzyzowania	48
Srednia liczba potomkow elitarnych	2
Srednia liczba potomkow zmutowanych	39,8
Liczba iteracji	73

Tabela 5:Wyniki dla krzyżowania jednopunktowego z szansą mutacji 0.05

1	0	0	1	0	1	0	1	0	0	1	1	1	0	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1

Tabela 6:Najlepszy wynik dla krzyżowania jednopunktowego z szansą mutacji 0.05



Rysunek 4:Wykresy wartości funkcji celu dla krzyżowania jednopunktowego z szansą mutacji 0.1

krzyżowanie jednopunktowe	
Szansa mutacji	0,1
Wartosc najlepszego rozwiazania	8
Waga najlepszego rozwiazania	418
Srednia liczba potomkow krzyzowania	48,1
Srednia liczba potomkow elitarnych	1,9
Srednia liczba potomkow zmutowanych	47,7
Liczba iteracji	69

Tabela 7:Wyniki dla krzyżowania jednopunktowego z szansą mutacji 0.1

0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	1

Tabela 8:Najlepszy wynik dla krzyżowania jednopunktowego z szansą mutacji 0.1

Listing programów

```
close all;

numer_albumu=321358; %% Wpisz swój
numer albumu
rng(numer_albumu);
N=32;
items(:,1)=round(0.1+0.9*rand(N,1),1);
items(:,2)=round(1+99*rand(N,1));

max_weight = 0.3*sum(items(:,2));

test_algorythm(items, max_weight, 'one
point', 0.01);


% function used to test the algotythm
function test_algorythm(items,
max_weight, crossover_type,
mutation_chance)
    [best_solution, fitness_history,
offspring_count]  =
genetic_knapsack(items, max_weight ,
mutation_chance, crossover_type);
    disp(['Mutation chance: ',
num2str(mutation_chance)])
    disp(['Crossover type: ',
crossover_type])
    disp(['Best solution:',
num2str(best_solution)]);
    disp(['Best solution value: ',
num2str(best_solution*items(:,1))]);
    disp(['Best solutin weight: ',
num2str(best_solution*items(:,2))]);
    disp(['mean crossover offspring:
',
num2str(mean(offspring_count(:,1)))])
    disp(['mean elite offspring: ',
num2str(mean(offspring_count(:,2)))])
    disp(['mean mutated offspring: ',
num2str(mean(offspring_count(:,3)))])

    plot_fitness(fitness_history)
end

% genetic algorythm
function [best_solution,
fitness_history, offspring_count] =
genetic_knapsack(items, max_weight,
mutation_chance, crossover_type)
    % parameters
    population_size = 50;
    num_generations = 10000;
    crossover_rate = 0.95;
    patience = 50;
    improvement_threshold = 1e-3;

    % initialization
    population = randi([0, 1],
population_size, size(items, 1));
    fitness_history =
zeros(num_generations, 5);
    offspring_count =
zeros(num_generations, 3);
    best_fitness = -Inf;
    generations_without_improvement =
0;

    for generation = 1:num_generations
        fitness = evaluate(population,
items, max_weight);

        [current_best_fitness, ~] =
max(fitness);
        fitness_history(generation, 1)
= current_best_fitness;
        fitness_history(generation, 2)
= min(fitness);
        fitness_history(generation, 3)
= mean(fitness);
        fitness_history(generation, 4)
= var(fitness);

        % stopping
        if current_best_fitness >
best_fitness + improvement_threshold
            best_fitness =
current_best_fitness;

generations_without_improvement = 0;
        else

generations_without_improvement =
generations_without_improvement + 1;
        end

        if
generations_without_improvement >=
patience
            fitness_history =
fitness_history(1:(generation-
patience+10), 1:4);
            offspring_count =
offspring_count(1:(generation-
patience+10), 1:3);
```

```
            break;
        end

        offspring_count(generation, 1)
= 0;
        offspring_count(generation, 2)
= 0;
        offspring_count(generation, 3)
= 0;

        % Selection
        selected_indices =
tournament_selection(fitness,
population_size);
        selected_population =
population(selected_indices, :);

        % crossover
        if crossover_type == 'two
point'
            % two point crossover
            new_population =
zeros(size(population));
            for i =
1:2:population_size
                parent1 =
selected_population(randi([1,
population_size]), :);
                parent2 =
selected_population(randi([1,
population_size]), :);
                if rand <
crossover_rate
                    points =
sort(randperm(size(items, 1), 2));
                    new_population(i,
:) = [parent1(1:points(1)),
parent2(points(1)+1:points(2)),
parent1(points(2)+1:end)];
                    new_population(i +
1, :) = [parent2(1:points(1)),
parent1(points(1)+1:points(2)),
parent2(points(2)+1:end)];

offspring_count(generation, 1) =
offspring_count(generation, 1) + 2;
                else % 20% szansa na
brak krzyżowania (kopiowanie rodziców)
                    new_population(i,
:) = parent1;
                    new_population(i +
1, :) = parent2;

offspring_count(generation, 2) =
offspring_count(generation, 2) + 2;
                end
            end
        else
            % one point crossover
            new_population =
zeros(size(population));
            for i =
1:2:population_size
                parent1 =
selected_population(randi([1,
population_size]), :);
                parent2 =
selected_population(randi([1,
population_size]), :);
                if rand <
crossover_rate
                    point = randi([1,
size(items, 1)-1]);
                    new_population(i,
:) = [parent1(1:point)
parent2(point+1:end)];

new_population(i+1, :) =
[parent2(1:point)
parent1(point+1:end)];

offspring_count(generation, 1) =
offspring_count(generation, 1) + 2;
                else
                    new_population(i,
:) = parent1;

new_population(i+1, :) = parent2;

offspring_count(generation, 2) =
offspring_count(generation, 2) + 2;
                end
            end
        end

        % Mutation
        for i = 1:population_size
            mutated = false;
            for j = 1:size(items, 1)
                if rand <
mutation_chance
                    new_population(i,
j) = ~new_population(i, j);
                    mutated = true;
                end
            end
            if mutated

offspring_count(generation, 3) =
offspring_count(generation, 3) + 1;
            end
        end
        population = new_population;
    end

    fitness = evaluate(population,
items, max_weight);
    [~, best_index] = max(fitness);
```

```
    best_solution =
population(best_index, :);
end

% function generating plots
function plot_fitness(fitness_history)
    figure;
    subplot(2,2,1)
    plot(fitness_history(:,1),
'Linewidth', 1)
    xlabel('Generation');
    ylabel('Fitness');
    title('Fitness maximum value')
    grid on;
    subplot(2,2,2)
    plot(fitness_history(:,2),
'Linewidth', 1)
    xlabel('Generation');
    ylabel('Fitness');
    title('Fitness minimum value')
    grid on;
    subplot(2,2,3)
    plot(fitness_history(:,3),
'Linewidth', 1)
    xlabel('Generation');
    ylabel('Mean fitness');
    title('Fitness mean value')
    grid on;
    subplot(2,2,4)
    plot(fitness_history(:,4),
'Linewidth', 1)
    xlabel('Generation');
    ylabel('Fitness variance');
    title('Fitness variance value')
    grid on;
end

% fitness function
function fitness =
evaluate(population, items,
max_weight)
    values = population * items(:, 1);
    weights = population * items(:,
2);
    penalty = max(0, weights -
max_weight);
    fitness = values - penalty;

end

% tournament selection function
function selected_indices =
tournament_selection(fitness,
population_size)
    tournament_size = 3;
    selected_indices =
zeros(population_size, 1);
    for i = 1:population_size
        competitors = randi([1,
population_size], tournament_size, 1);
        [~, best] =
max(fitness(competitors));
        selected_indices(i) =
competitors(best);
    end
end
```