KI LaTeX DOKUMENT
Materiały do przedmiotu "Rozwiązywanie zadań odwrotnych"

Metody ewolucyjne - zastosowanie metod ewolucyjnych do rozwiązania zadania wyprzedzającego

dr inż. Konrad M. Gruszka,*

Abstract. W tym dokuemncie przedstawiam ogólne informacje na temat działania metod ewolucyjnych w kontekście ich zastosowania w problemie transferu ciepła przez jednowymiarowy materiał jednorodny. Następnie omawiamy przykładowy kod, służący do generowania rozkładu temperatury w zadanym systemie z użyciem metody opartej o algorytm genetyczny.

1 Wprowadzenie

Algorytm Genetyczny
Algorytmy genetyczne to jedna z metod optymalizacji inspirowana procesem naturalnej selekcji. Działają one w oparciu o populację potencjalnych rozwiązań, która ewoluuje w wyniku selekcji, krzyżowania i mutacji. Celem jest znalezienie optymalnego rozwiązania problemu (np. optymalnego rozkładu temperatury).

Kluczowe elementy algorytmu genetycznego:

- **Populacja**: To zbiór rozwiązań (osobników), którymi manipuluje algorytm. Każde rozwiązanie reprezentowane jest przez wektor, w którym znajdują się różne zmienne problemu. W tym przypadku to różne rozkłady temperatury.
- **Funkcja oceny (fitness)**: Funkcja oceny określa jakość rozwiązań w populacji. Zwykle jest to funkcja celu, która dla danego rozwiązania oblicza, jak dobre jest ono w kontekście problemu (np. jak dobrze rozkład temperatury pasuje do zadanego rozkładu).
- **Selekcja**: Selekcja decyduje, które osobniki (rozwiązania) będą przekazywane do następnej generacji. W algorytmach genetycznych najczęściej stosuje się selekcję turniejową, gdzie losowo wybierane są dwa osobniki, a lepszy z nich przechodzi do następnej rundy.
- **Krzyżowanie (crossover)**: Krzyżowanie to operacja, która łączy cechy dwóch rodziców, tworząc dzieci. W kontekście algorytmu genetycznego może to być np. wymiana części wektorów rozwiązań między dwoma osobnikami (np. część rozkładu temperatury pierwszego rodzica zostaje wymieniona z częścią drugiego rodzica).
- **Mutacja**: Mutacja to proces, który wprowadza losowe zmiany w osobnikach w

* Katedra Informatyki, Wydział Informatyki i Sztucznej Inteligencji (kgruszka@icis.pcz.pl)

sposób losowy. Może to polegać na drobnych zmianach w rozkładzie temperatury. Mutacja pomaga uniknąć utknięcia w lokalnych minimach funkcji celu.

- **Generacje**: Algorytm genetyczny działa w cyklach, tzw. generacjach. W każdej generacji populacja jest aktualizowana przez selekcję, krzyżowanie i mutację. Z każdym pokoleniem populacja staje się coraz lepsza w kontekście optymalizacji.

1.1 Rozkład temperatury i funkcja celu

W problemie rozkładu temperatury, głównym celem jest znalezienie temperatur w różnych punktach pręta, które najlepiej pasują do zadanego rozkładu. Funkcja celu (np. suma kwadratów różnic między obliczonym rozkładem temperatury a oczekiwanym rozkładem) służy jako miernik jakości rozwiązania.

Funkcja celu: Funkcja celu jest matematycznym wyrażeniem, które ocenia, jak dobre dane rozwiązanie jest w kontekście problemu. W kontekście algorytmu genetycznego funkcja ta może wyglądać następująco:

$$J(T_{\text{model}}) = \sum_i (T_{\text{model},i} - T_{\text{target},i})^2 \tag{1}$$

Gdzie:

- $T_{\text{model},i}$ to temperatura w punkcie i rozkładu obliczonego przez algorytm,
- $T_{\text{target},i}$ to zadana (docelowa) temperatura w tym punkcie.

Celem algorytmu genetycznego jest minimalizacja tej funkcji, co oznacza, że algorytm stara się znaleźć rozkład temperatury, który jak najlepiej odwzorowuje rozkład docelowy.

1.2 Wszystkie parametry algorytmu

Różne parametry algorytmu genetycznego kontrolują jego działanie i wpływają na efektywność optymalizacji. Wartości tych parametrów powinny być odpowiednio dobrane, aby algorytm działał efektywnie.

- **Populacja** (*pop_size*): Określa liczbę osobników (rozwiązań) w każdej generacji. Większa populacja daje algorytmowi więcej rozwiązań do wyboru, ale zwiększa również czas obliczeń.
- **Liczba pokoleń** (*num_generations*): To liczba iteracji (generacji), przez które będzie przechodził algorytm. W każdej generacji populacja jest modyfikowana, a jej jakość powinna poprawiać się z pokolenia na pokolenie.
- **Współczynnik krzyżowania** (*crossover_rate*): Określa prawdopodobieństwo, z jakim operacja krzyżowania zostanie zastosowana do pary rodziców. Wyższy współczynnik krzyżowania zwiększa różnorodność w populacji.
- **Współczynnik mutacji** (*mutation_rate*): Określa prawdopodobieństwo, z jakim zostanie przeprowadzona mutacja w rozwiązaniu. Mutacje wprowadzają losowe zmiany w rozwiązaniu, co pozwala algorytmowi uniknąć utknięcia w lokalnym minimum.

1.3 Selekcja turniejowa

Selekcja turniejowa jest prostą i skuteczną metodą doboru najlepszych osobników do kolejnej generacji. Polega na losowym wyborze dwóch osobników z populacji i porównaniu ich wyników. Osobnik, który ma lepszy wynik funkcji celu (czyli jest bardziej "dopasowany"), przechodzi do kolejnej generacji.

Dzięki tej metodzie, algorytm nie marnuje zasobów na osobników o słabej jakości, skupiając się na najlepszych rozwiązaniach.

1.4 Krzyżowanie

Krzyżowanie polega na wymianie informacji między dwoma osobnikami, aby stworzyć nowe rozwiązania (dzieci). Dzięki temu możliwe jest "połączenie" dobrych cech dwóch rozwiązań w jedno.

W klasycznym algorytmie genetycznym najczęściej stosuje się **krzyżowanie jednopunktowe**, gdzie część wektora jednego rodzica jest wymieniana z częścią wektora drugiego rodzica.

1.5 Mutacja

Mutacja jest procesem, który wprowadza losowe zmiany w jednym lub kilku elementach rozwiązania. Dzięki temu algorytm nie utknie w miejscach lokalnych minimów i będzie mógł eksplorować nowe, potencjalnie lepsze rozwiązania. Mutacja może być stosunkowo rzadko stosowaną operacją w porównaniu do krzyżowania.

1.6 Generacje

Algorytm genetyczny działa w cyklach, tzw. generacjach. W każdej generacji:

- Selekcja wybiera najlepszych osobników,
- Krzyżowanie tworzy nowe osobniki,
- Mutacja wprowadza losowe zmiany.

Każda generacja powinna prowadzić do poprawy jakości rozwiązania (niższa wartość funkcji celu).

2 Algorytm

Poniżej przedstawiam realizację prostego algorytmu genetycznego wraz z wyjaśnieniem funkcjonowania poszczególnych bloków kodu.

```
import numpy as np
import matplotlib.pyplot as plt

##
## DEFINICJA PARAMETRÓW POCZĄTKOWYCH MODELU
##

# Parametry algorytmu genetycznego
pop_size = 100  # Rozmiar populacji
num_generations = 100  # Liczba pokoleń
```

```
mutation_rate = 0.1  # Prawdopodobieństwo mutacji
crossover_rate = 0.7  # Prawdopodobieństwo krzyżowania

# Warunki brzegowe
T_start = 100  # Temperatura na początku listy
T_end = 50  # Temperatura na końcu listy

# Funkcja celu - przykładowo błąd kwadratowy
def fitness(temp_array):
    # Przykładowa funkcja celu: suma kwadratów różnic od wartości docelowej (np. równomierny
    ↪  rozkład temperatur)
    target = np.linspace(T_start, T_end, 20)
    return np.sum((temp_array - target)**2)

#INICJALIZACJA POPULACJI
def initialize_population():
    return [np.concatenate([[T_start], np.random.uniform(T_start, T_end, 18), [T_end]]) for
    ↪  _ in range(pop_size)]

##
## SELEKCJA METODĄ TURNIEJOWĄ
##
def tournament_selection(pop):
    new_pop = []
    pop_size = len(pop)
    for _ in range(pop_size):
        # Losowanie indeksów dla dwóch osobników
        idx1, idx2 = np.random.choice(pop_size, 2, replace=False)
        indiv1, indiv2 = pop[idx1], pop[idx2]
        # Wybór osobnika z lepszym dostosowaniem
        indiv = indiv1 if fitness(indiv1) < fitness(indiv2) else indiv2
        new_pop.append(indiv)
    return new_pop

def crossover(parent1, parent2):
    if np.random.rand() < crossover_rate:
        point = np.random.randint(1, len(parent1) - 1)
        child1 = np.concatenate([parent1[:point], parent2[point:]])
        child2 = np.concatenate([parent2[:point], parent1[point:]])
        return child1, child2
    else:
        return parent1, parent2

##
## MUTACJA
##

def mutate(indiv):
    if np.random.rand() < mutation_rate:
        point = np.random.randint(1, len(indiv) - 1)
        indiv[point] = np.random.uniform(T_start, T_end)
    return indiv
```

```
##  ---------------------------------------------
##  -                GŁÓWNA PĘTLA                -
##  -   -   -   -   -   -   -   -   -   -   -   - -

population = initialize_population()
best_solution = None
best_fitness = float('inf')

for generation in range(num_generations):
    population = tournament_selection(population)
    next_generation = []
    for i in range(0, len(population), 2):
        parent1, parent2 = population[i], population[i+1]
        child1, child2 = crossover(parent1, parent2)
        next_generation.extend([mutate(child1), mutate(child2)])
    population = next_generation

    # Znajdowanie i zapisywanie najlepszego rozwiązania
    current_best = min(population, key=fitness)
    current_best_fitness = fitness(current_best)
    if current_best_fitness < best_fitness:
        best_fitness = current_best_fitness
        best_solution = current_best

    print(f"Generation {generation}: Best Fitness = {best_fitness}")

# Wykres najlepszego rozwiązania
plt.plot(best_solution)
plt.title("Best Solution")
plt.xlabel("Position")
plt.ylabel("Temperature")
plt.show()
```

Algorytm genetyczny zastosowany w tym kodzie ma na celu optymalizację rozkładu temperatury w jednorodnym pręcie 1D w stanie stacjonarnym. Jest to klasyczna metoda ewolucyjna, której celem jest znalezienie najlepszego rozwiązania poprzez symulowanie procesu doboru naturalnego (selekcja), krzyżowanie i mutację. Oto ogólna struktura działania:

- Inicjalizacja populacji: Populacja składa się z losowych rozkładów temperatury (z wyjątkiem pierwszego i ostatniego elementu, które są ustalone jako T_{start} i T_{end}).
- Selekcja: Selekcja polega na wyborze najlepszych osobników do kolejnej generacji. W tym przypadku wykorzystana jest metoda turniejowa, w której z dwóch losowo wybranych osobników wybierany jest ten o lepszym dopasowaniu (mniejszym błędzie w porównaniu do wartości docelowej).
- Krzyżowanie: Dla dwóch wybranych rodziców, tworzona jest para dzieci za pomocą krzyżowania jednopunktowego. Część genotypu jednego rodzica jest łączona z częścią genotypu drugiego rodzica, tworząc nowe osobniki.
- Mutacja: Co pewien czas (z określoną prawdopodobieństwem), wprowadzana jest mutacja — zmiana jednej z temperatur w rozkładzie.
- Wybór najlepszego rozwiązania: W każdej generacji algorytm sprawdza, który z

osobników ma najlepszy wynik (minimalizuje funkcję celu). Wynikiem tej funkcji jest suma kwadratów różnic między obecnym rozkładem temperatury a oczekiwanym (proporcjonalnym do linii prostej).

- Powtarzanie procesu: Proces selekcji, krzyżowania i mutacji jest powtarzany przez zdefiniowaną liczbę pokoleń. W każdej iteracji populacja "ewoluuje", a najlepsze rozwiązanie jest zapisywane.

2.1 Szczegółowe wyjaśnienie algorytmu

(1) Inicjalizacja populacji:

```
def initialize_population():
    return [np.concatenate([[T_start], np.random.uniform(T_start, T_end, 18),
    ↪  [T_end]]) for _ in range(pop_size)]
```

Funkcja ta tworzy początkową populację rozkładów temperatury. Każdy osobnik w populacji (rozwiązanie) jest tablicą z N=20 (dla 18 węzłów + 2 warunki brzegowe) temperaturami, gdzie:
- temperatura na początku i końcu pręta są ustalone na T_{start} i T_{end},
- pozostałe 18 temperatur jest losowo dobieranych z zakresu między T_{start} i T_{end}.

(2) Selekcja metodą turniejową

```
def tournament_selection(pop):
    new_pop = []
    pop_size = len(pop)
    for _ in range(pop_size): # Pętla dla każdego osobnika w populacji
        idx1, idx2 = np.random.choice(pop_size, 2, replace=False)
        indiv1, indiv2 = pop[idx1], pop[idx2]
        indiv = indiv1 if fitness(indiv1) < fitness(indiv2) else indiv2
        new_pop.append(indiv)
    return new_pop
```

Metoda ta polega na losowym wyborze dwóch osobników (zwykle w ramach "turnieju"), porównaniu ich wyników i wybraniu tego, który daje lepszy wynik (mniejsza funkcja celu). Najpierw tworzymy pustą listę, w której przechowujemy nową populację. Następnie iterujemy po każdym elemencie (osobniku) populacji, losujemy dwa różne osobniki, porównujemy ich dopasowanie (fitness) i wybieramy tego lepszego. Wybrany osobnik jest dodawany do nowej populacji. Proces powtarzamy dla całej populacji, a na koniec zwracamy nową populację z wybranymi osobnikami.

(3) Krzyżowanie jednopunktowe:

```
def crossover(parent1, parent2):
    if np.random.rand() < crossover_rate:
        point = np.random.randint(1, len(parent1) - 1)
        child1 = np.concatenate([parent1[:point], parent2[point:]])
        child2 = np.concatenate([parent2[:point], parent1[point:]])
        return child1, child2
    else:
        return parent1, parent2
```

Krzyżowanie odbywa się poprzez losowy wybór punktu w obrębie rozwiązania (np. w tablicy temperatur). Następnie, część jednego rodzica jest łączona z częścią drugiego, tworząc dwoje dzieci. Jeśli krzyżowanie nie zachodzi, rodzice pozostają bez zmian. Najpierw sprawdzamy, czy zostanie wykonane krzyżowanie, porównując losową wartość z prawdopodobieństwem krzyżowania (crossover_rate). Jeśli warunek jest spełniony, losujemy punkt, w którym nastąpi podział rodziców (parent1, parent2). Następnie tworzymy dwoje dzieci: pierwsze dziecko powstaje przez połączenie części z pierwszego rodzica i reszty z drugiego, a drugie dziecko odwrotnie. Jeśli krzyżowanie nie zachodzi, zwracamy niezmienionych rodziców.

(4) Mutacja:

```
def mutate(indiv):
    if np.random.rand() < mutation_rate:
        point = np.random.randint(1, len(indiv) - 1)
        indiv[point] = np.random.uniform(T_start, T_end)
    return indiv
```

Mutacja polega na losowej zmianie jednej z temperatur w rozkładzie, co umożliwia algorytmowi eksplorację nowych rozwiązań. Prawdopodobieństwo mutacji jest określone przez parametr mutation_rate. Najpierw sprawdzamy, czy zostanie wykonana mutacja, porównując losową wartość z prawdopodobieństwem mutacji (mutation_rate). Jeśli warunek jest spełniony, losujemy punkt w osobniku, który zostanie zmieniony, a następnie przypisujemy temu punktowi losową wartość temperatury z zakresu od T_{start} i T_{end}. Na końcu zwracamy zmodyfikowanego osobnika.

(5) Oblicznaie funkcji celu:

```
def fitness(temp_array):
    target = np.linspace(T_start, T_end, 20)
    return np.sum((temp_array - target)**2)
```

Funkcja celu ocenia, jak dobrze dany rozkład temperatury pasuje do "celowego" rozkładu (który jest liniowy, z T_{start} na początku i T_{end} na końcu pręta). Błąd kwadratowy między bieżącym rozwiązaniem a docelowym jest sumowany i traktowany jako miara "nieoptymalności" danego rozwiązania. Najpierw tworzymy wektor target, który zawiera 20 równomiernie rozłożonych wartości temperatur w przedziale od T_{start} do T_{end}. Można tutaj zadać inny rozkład, np otrzymany z MRS. Następnie funkcja oblicza sumę kwadratów różnic między temperaturą w danym osobniku (temp_array) a wartościami w target. Im mniejszą wartość zwraca funkcja, tym lepszy jest osobnik.

(6) Główna pętla algorytmu: W głównej pętli algorytm ewoluuje populację przez kolejne pokolenia, wprowadzając selekcję, krzyżowanie i mutację. Po każdej generacji zapisuje najlepsze rozwiązanie, a po zakończeniu wszystkich pokoleń wykreśla wykres najlepszego rozwiązania (najmniejszy błąd).

```
population = initialize_population()
best_solution = None
best_fitness = float('inf')
```

```
for generation in range(num_generations):
    population = tournament_selection(population)
    next_generation = []
    for i in range(0, len(population), 2):
        parent1, parent2 = population[i], population[i+1]
        child1, child2 = crossover(parent1, parent2)
        next_generation.extend([mutate(child1), mutate(child2)])
    population = next_generation

    # Znajdowanie i zapisywanie najlepszego rozwiązania
    current_best = min(population, key=fitness)
    current_best_fitness = fitness(current_best)
    if current_best_fitness < best_fitness:
        best_fitness = current_best_fitness
        best_solution = current_best

    print(f"Generation {generation}: Best Fitness = {best_fitness}")
```

Na początku generujemy początkową populację (rozwiązania) przy pomocy funkcji initialize_population(), a następnie rozpoczynamy iterację przez zdefiniowaną liczbę pokoleń. W każdej generacji przeprowadzamy selekcję turniejową, aby wybrać najlepszych osobników do kolejnej rundy. Następnie, dla każdej pary rodziców, wykonujemy krzyżowanie, a potem mutację, tworząc nową generację. Po każdej iteracji algorytm sprawdza, które rozwiązanie ma najlepsze dopasowanie (najmniejszy błąd) i zapisuje je jako najlepsze rozwiązanie. Na koniec, po zakończeniu wszystkich generacji, algorytm wypisuje wynik w postaci najlepszego rozwiązania w każdej generacji.

3 Wariacje na temat...

3.1 Selekcja ruletkowa:

Selekcja ruletkowa to probabilistyczna metoda selekcji, gdzie prawdopodobieństwo wyboru osobnika zależy od jego przystosowania (fitness). Jak działa selekcja ruletkowa?:

(1) Oblicz sumę przystosowania (S) wszystkich osobników w populacji.
(2) Oblicz prawdopodobieństwo wyboru dla każdego osobnika (i):

$$P(i) = \frac{fitness(i)}{S}$$

(3) Stwórz "koło ruletki", gdzie każdy osobnik zajmuje fragment proporcjonalny do swojego fitness.
(4) Losuj wartość z zakresu [0, S] i wybierz osobnika, którego segment obejmuje tę wartość.
(5) Powtórz kroki 3–4 aż do uzyskania wymaganej liczby osobników.

3.2 Selekcja rangowa (Rank Selection):

Tutaj nie liczy się bezpośrednia wartość fitness, ale pozycja osobnika w populacji uszeregowanej według fitness.

Jak działa selekcja rangowa?

(1) Uszereguj osobniki według fitness (od najlepszego do najgorszego).
(2) Przypisz im rangi: najlepszy dostaje rangę 1, drugi 2, itd.
(3) Oblicz prawdopodobieństwo wyboru na podstawie rangi, np.:

$$P(i) = \frac{ranga(i)}{\sum rang}$$

(4) Użyj selekcji ruletkowej z nowymi wartościami rang.

3.3 Krzyżowanie dwupunktowe

Krzyżowanie dwupunktowe jest podobne do jednopunktowego, ale zamiast jednego punktu wybieramy dwa punkty, a fragment między nimi podlega wymianie.

(1) Wybierz dwa losowe punkty podziału chromosomu.
(2) Podziel chromosomy w tych miejscach.
(3) Wymień środkowe fragmenty między rodzicami.
(4) Utwórz dwa nowe potomki.

Przykład:

```
Rodzic 1:  101|100|10
Rodzic 2:  011|011|01
```

Po krzyżowaniu:

```
Potomek 1: 101|011|10
Potomek 2: 011|100|01
```

3.4 Krzyżowanie jednorodne (Uniform Crossover)

W tym podejściu każdy gen potomka jest wybierany losowo od jednego z rodziców, bez stałych punktów podziału.
Jak to działa?

(1) Dla każdego genu rzucamy monetą.
(2) Jeśli wynik to "orzeł", gen pochodzi od Rodzica 1.
(3) Jeśli wynik to "reszka", gen pochodzi od Rodzica 2.
(4) Powtarzamy dla każdego genu, aż powstanie nowy chromosom.

Przykład:

```
Rodzic 1: 10110010
Rodzic 2: 01101101
```

Po krzyżowaniu:

```
Potomek 1: 10101100
Potomek 2: 01110011
```

4 Podsumowanie

Aby zrozumieć działanie algorytmu genetycznego w kontekście rozkładu temperatury, należy zrozumieć, jak algorytm manipuluje populacją rozwiązań (temperatur) przez selekcję, krzyżowanie i mutację, aby znaleźć najlepszy możliwy rozkład temperatury, który minimalizuje różnicę między obliczonym i docelowym rozkładem temperatury. Parametry algorytmu, takie jak rozmiar populacji, liczba pokoleń, współczynniki krzyżowania i mutacji, mają kluczowy wpływ na efektywność tego procesu.