Metaheurystyki — zadanie 2

Algorytm symulowanego wyżarzania

GRUPA 3 — piątek 10:15

Bartosz Kołaciński
251554

Nikodem Nowak
251598

6 listopada 2025

Użyte technologie	Python 3.13
Użyte biblioteki	random, collections.abc, time, math, numpy, matplotlib.pyplot

Spis treści

1 Opis zasad działania algorytmu

1.1 Opis algorytmu symulowanego wyżarzania

Algorytm symulowanego wyżarzania to metaheurystyka inspirowana procesem fizycznym wyżarzania metali. Celem algorytmu jest znalezienie przybliżonego ekstremum danej funkcji. Algorytm rozpoczyna się od losowego rozwiązania początkowego i iteracyjnie poszukuje lepszych rozwiązań poprzez generowanie nowych kandydatów w sąsiedztwie aktualnego rozwiązania. W każdej iteracji, jeśli nowe rozwiązanie jest lepsze od aktualnego, zostaje zaakceptowane jako nowe rozwiązanie. Jeśli jest gorsze, może zostać zaakceptowane z pewnym prawdopodobieństwem, które zależy od różnicy jakości rozwiązań oraz bieżącej temperatury i współczynnika k. Temperatura jest stopniowo obniżana w trakcie działania algorytmu, co zmniejsza prawdopodobieństwo akceptacji gorszych rozwiązań oraz zmniejsza obszar poszukiwań.

1.2 Opis implementacji rozwiązania

Cała logika algorytmu symulowanego wyżarzania została zaimplementowana w klasie `SimulatedAnnealing`. `self.func` to funkcja której maksimum chcemy znaleźć, `self.domain` to obszar w którym szukamy maksimum, a `self.dimensions` to liczba wymiarów problemu optymalizacyjnego.

```
    class SimulatedAnnealing:
        def __init__(self,
                 func: Callable,
                 domain: tuple[float, float] | list[tuple[float, float]])

```

Kod 1: Inicjalizacja klasy SimulatedAnnealing

Metoda `run_epochs` klasy `SimulatedAnnealing` odpowiada za przeprowadzenie określonej liczby epok algorytmu symulowanego wyżarzania. W każdej epoce algorytm wykonuje określoną liczbę iteracji, w trakcie których poszukuje lepszego rozwiązania.
`epochs` to liczba epok do wykonania, `attempts_per_epoch` to liczba prób w każdej epoce, `init_temp` to początkowa temperatura, `alpha` to funkcja schładzająca temperaturę, a `k` to współczynnik wpływający na prawdopodobieństwo akceptacji gorszych rozwiązań.

```
    def run_epochs(self,
                   epochs: int,
                   attempts_per_epoch: int,
                   init_temp: float,
                   alpha: Callable[[float], float],
                   k: float) -> tuple[list[float], float]:

```

Kod 2: Metoda run_epochs klasy SimulatedAnnealing

Na początku metody `run_epochs` zajmujemy się inicjalizacją wartości. Początkowy punkt (nieograniczony do jednego wymiaru) jest wybierany losowo z podanego przedziału lub obszaru określonego przez `self.domain`. Wartość funkcji w tym punkcie jest obliczana i przechowywana jako `best_f_value`. Temperatura jest inicjalizowana wartością `init_temp`.

```
    best_point = [uniform(self.domain[i][0], self.domain[i][1])
                  for i in range(self.dimensions)]
    best_f_value = self.func(*best_point)
    temp = init_temp

```

Kod 3: Wybór losowego punktu oraz inicjalizacja temperatury

Fragment tworzy nowy punkt będący kandydatem do bycia nowym najlepszym punktem. Pętla losuje współrzędną z sąsiedztwa najlepszego rozwiązania, ogranicza ją do wcześniej podanej dziedziny lub obszaru i dodaje do `new_point`. Dzięki zależności zakresu losowania od temperatury algorytm przy wysokiej temperaturze eksploruje dalej, a przy niskiej koncentruje się wokół bieżącego najlepszego rozwiązania.

```
    new_point = []
    for i in range(self.dimensions):
        left, right = self.domain[i]
        new_coord = uniform(best_point[i] - 2 * temp,
                            best_point[i] + 2 * temp)
        new_coord = max(left, min(new_coord, right))
        new_point.append(new_coord)

```

Kod 4: Główna pętla algorytmu symulowanego wyżarzania

Gorsze rozwiązanie zostaje zaakceptowane gdy zajdzie warunek $e^{\frac{f(x')-f(x)}{kT}} > p$, gdzie p to losowa liczba z przedziału $[0, 1]$, $f(x')$ to wartość funkcji w nowym punkcie, $f(x)$ to wartość funkcji w najlepszym dotychczasowym punkcie, k to współczynnik wpływający na prawdopodobieństwo akceptacji gorszych rozwiązań, a T to bieżąca temperatura.

```
    p = uniform(0, 1)
    if exp((f_value - best_f_value) / (k * temp)) > p:
        best_point, best_f_value = new_point, f_value

```

Kod 5: Główna pętla algorytmu symulowanego wyżarzania

1.3 Opis uruchomienia programu

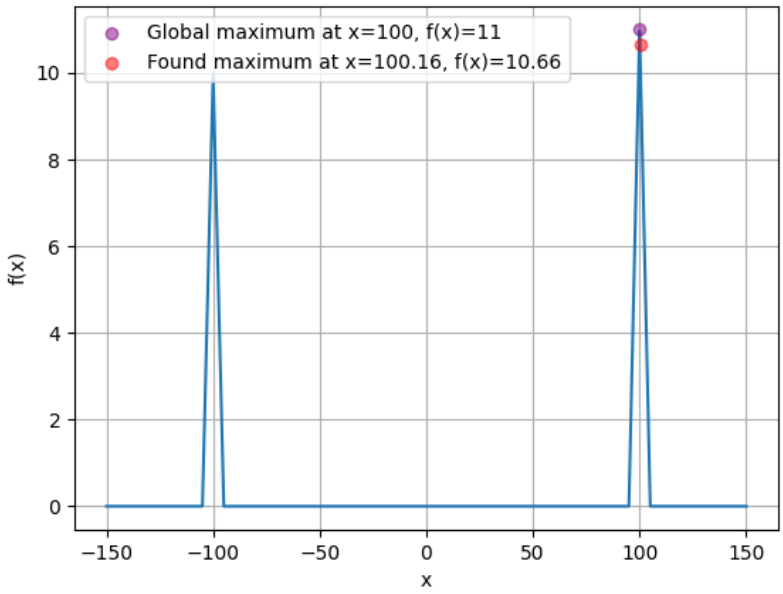
Program uruchamia się poprzez uruchomienie pliku main.py np. komendą `python main.py` w terminalu. Program umożliwia wybór jednej z czterech funkcji do optymalizacji: funkcji jednowymiarowej z sekcji 3 (przedział $[-150, 150]$), funkcji dwuwymiarowej z sekcji 4 (obszar $[-3, 12] \times [4.1, 5.8]$), lub własnej funkcji jedno- lub dwuwymiarowej. Przy definiowaniu własnej funkcji dostępne są standardowe funkcje matematyczne (sinus, cosinus, tangens, eksponens, logarytm, pierwiastek kwadratowy) oraz stałe (pi i e). Parametry algorytmu obejmują: liczbę epok, liczbę iteracji w epoce, temperaturę początkową, współczynnik alpha z przedziału $(0, 1)$ oraz stałą k. Po zakończeniu obliczeń program wyświetla znaleziony punkt (współrzędną x dla funkcji jednowymiarowej lub współrzędne x i y dla funkcji dwuwymiarowej), wartość funkcji w tym punkcie, numer iteracji oraz czas wykonania w milisekundach.

2 Odtworzenie eksperymentów z artykułu

2.1 Wybrana funkcja z rozdziału 3

Z sekcji 3 wybraliśmy funkcję $f(x)$ z przykładu 1, określoną w przedziale $[-150, 150]$, wyrażoną wzorem:

$$f(x) = \begin{cases} -2|x + 100| + 10 & \text{dla } x \in (-105, -95) \\ -2.2|x - 100| + 11 & \text{dla } x \in (95, 105) \\ 0 & \text{dla } x \notin (-105, -95) \cup (95, 105) \end{cases}$$



Rysunek 1: Wykres funkcji $f(x)$ w przedziale $[-150, 150]$ z oznaczonym znalezionym maksimum

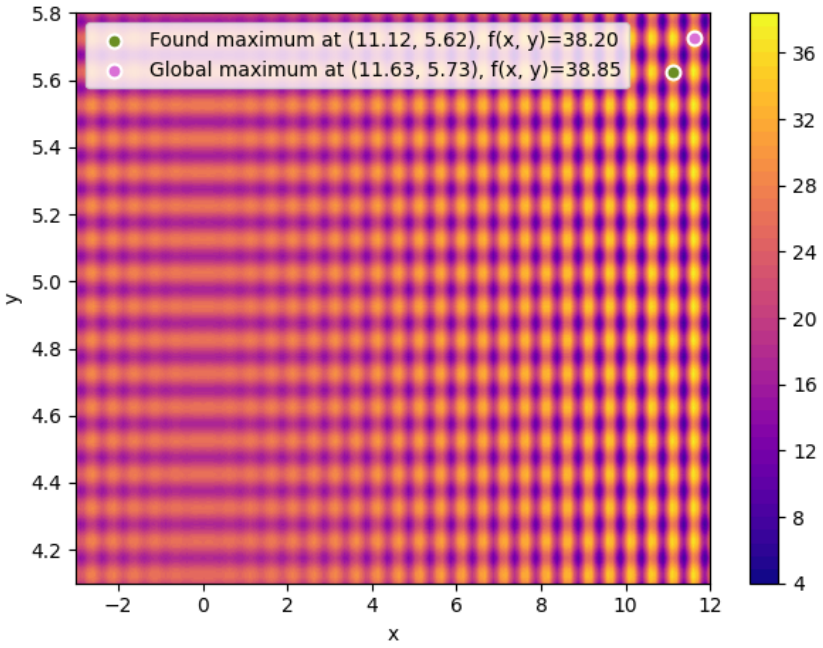
Wybrana funkcja posiada dwa ekstrema lokalne - $f(-100) = 10$ oraz $f(100) = 11$. Funkcja poza okolicami tych ekstremów przyjmuje wartość 0. Jej maksimum globalne to $f(100) = 11$.

Znalezione przez nas maksimum funkcji $f(x)$ to $f(100.16) = 10.66$ (rysunek 1), co różni się od maksimum globalnego o $\delta = |10.66 - 11| = 0.34$.

2.2 Wybrana funkcja z rozdziału 4

Z sekcji 4 wybraliśmy funkcję $f(x)$ z przykładu 5, określoną w obszarze $[-3, 12] \times [4.1, 5.8]$, wyrażoną wzorem:

$$f(x, y) = 21.5 + x \cdot \sin(4 \cdot \pi \cdot x) + y \cdot \sin(20 \cdot \pi \cdot y)$$



Rysunek 2: Wykres funkcji $f(x, y)$ w obszarze $[-3, 12] \times [4.1, 5.8]$ z oznaczonym znalezionym maksimum oraz maksimum globalnym

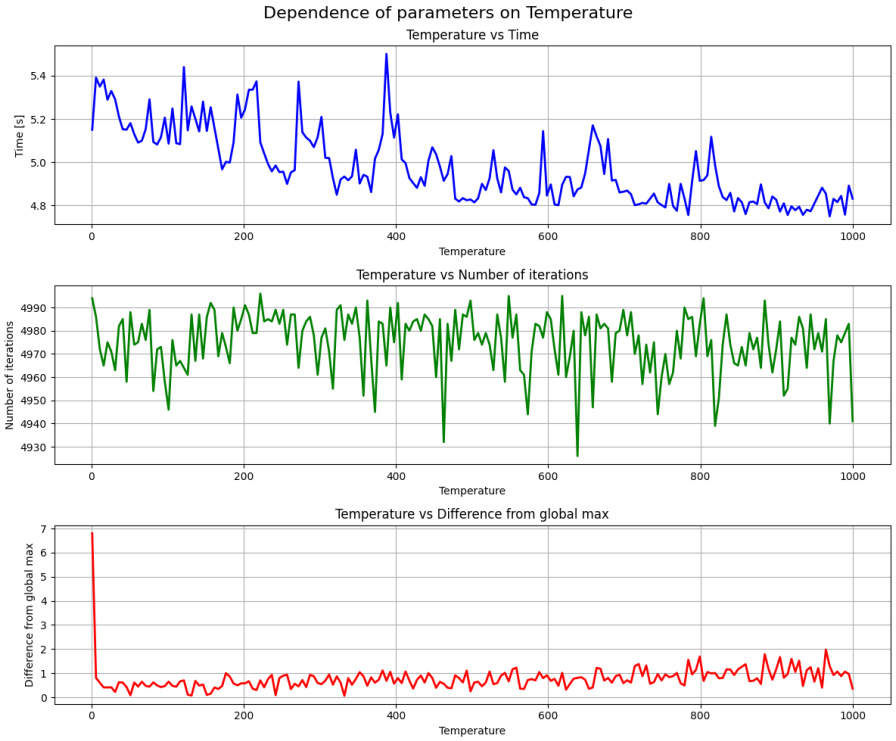
Wybrana funkcja posiada bardzo dużo ekstremów lokalnych w określonym obszarze. Jej maksimum globalne to $f(12, 5.7) = 38.85$.

Znalezione przez nas maksimum funkcji $f(x, y)$ to $f(11.12, 5.62) = 38.20$ (rysunek 2), co różni się od maksimum globalnego o $\delta = |38.20 - 38.85| = 0.65$.

2.3 Analiza wpływu parametrów algorytmu na jakość rozwiązania

2.3.1 Analiza wpływu temperatury na jakość rozwiązania

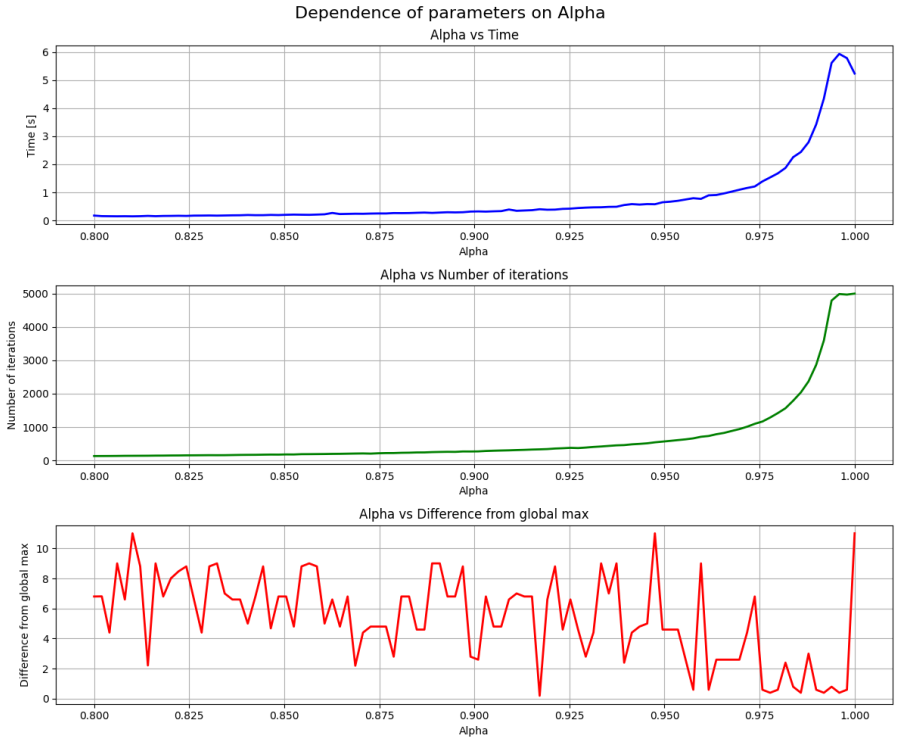
Wartość temperatury początkowej ma istotny wpływ na dokładność rozwiązania (co widać na rysunku 3), ale nie ma wpływu na czas trwania algorytmu ani na liczbę iteracji.



Rysunek 3: Zależności pomiędzy wartością temperatury początkowej a czasem, ilością iteracji oraz jakością rozwiązania

2.3.2 Analiza wpływu alpha na jakość rozwiązania

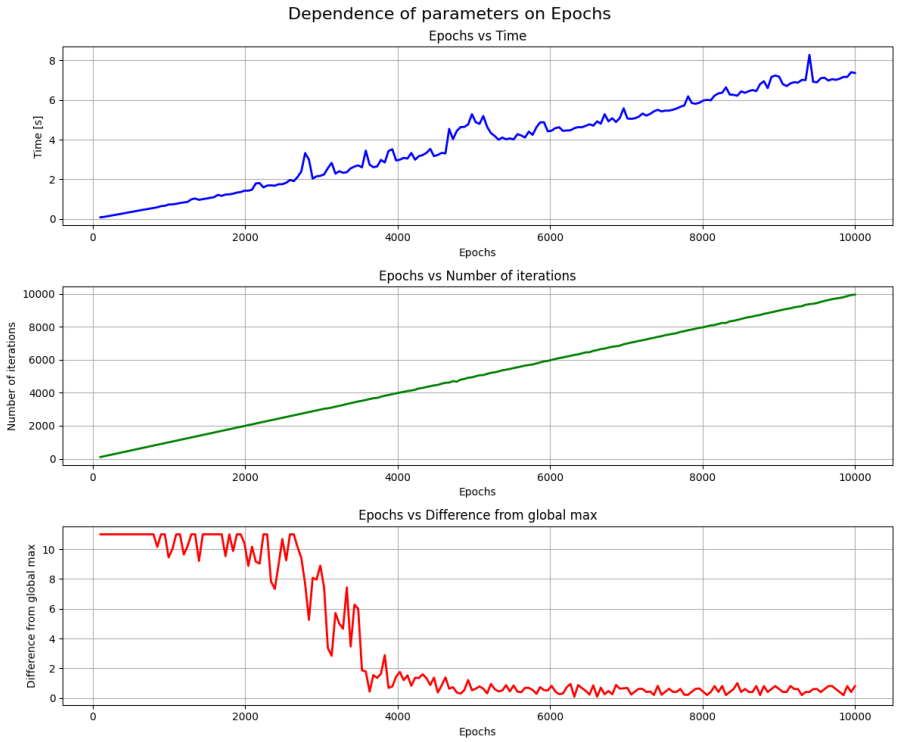
Wartość współczynnika α ma istotny wpływ na wszystkie trzy badane aspekty: czas trwania algorytmu, liczbę iteracji oraz dokładność rozwiązania (co widać na rysunku 4). Wykres zależności pomiędzy wartością α a czasem oraz liczbą iteracji przyjmują charakterystyczny kształt ze względu na drugi warunek stopu, którym jest osiągnięcie niskiej wartości temperatury.



Rysunek 4: Zależności pomiędzy wartością współczynnika α a czasem, ilością iteracji oraz jakością rozwiązania

2.3.3 Analiza wpływu liczby epok na jakość rozwiązania

Liczba epok wpływa na wszystkie trzy badane aspekty: czas trwania algorytmu, liczbę iteracji oraz dokładność rozwiązania. Pomiędzy ilością epok a czasem wykonania oraz pomiędzy ilością epok a ilością iteracji potrzebnych do znalezienia najlepszego rozwiązania łatwo zauważyć zależność liniową (co widać na rysunku 5). Zbyt niska wartość liczby epok skutkuje słabą dokładnością rozwiązania.



Rysunek 5: Zależności pomiędzy liczbą epok a czasem, ilością iteracji oraz jakością rozwiązania

3 Wnioski

Zaimplementowany algorytm symulowanego wyżarzania jest skuteczną metodą znajdowania rozwiązań bliskich optymalnym, co potwierdzają przeprowadzone eksperymenty. Nie gwarantuje on jednak znalezienia globalnego maksimum funkcji.

Kluczowe dla efektywności algorytmu jest odpowiednie dobranie parametrów:

- **Temperatura początkowa** ma istotny wpływ na jakość rozwiązania, nie wpływając na czas obliczeń. Wyższa wartość pozwala na szerszą eksplorację przestrzeni rozwiązań. Zbyt mała wartość może prowadzić do utknięcia w lokalnym ekstremum.
- **Współczynnik chłodzenia** (α) decyduje o tempie obniżania temperatury, co wpływa zarówno na czas działania algorytmu, jak i na dokładność znalezionego rozwiązania. Jego wartość pozwala znaleźć równowagę między szybkością zbieżności a ryzykiem przedwczesnego utknięcia w lokalnym ekstremum.
- **Liczba epok** wprost przekłada się na czas wykonania i liczbę iteracji. Zwiększenie tej wartości zazwyczaj poprawia jakość wyniku, lecz wiąże się z większym kosztem obliczeniowym.

Podsumowując, skuteczność algorytmu symulowanego wyżarzania jest silnie uzależniona od starannego dostrojenia parametrów w celu zbalansowania jakości rozwiązania i czasu jego poszukiwania.