White Shark Optimizer & EPANET – pierwsze uruchomienie

Igor Swat, Rafał Piwowar

Niniejszy dokument opisuje postęp prac w zakresie implementacji algorytmu WSO (White Shark Optimizer) oraz integracji tegoż algorytmu ze środowiskiem symulacyjnym EPANET.

1. White Shark Optimizer - implementacja (Python)

Pierwszym punktem naszych prac była implementacja algorytmu White Shark Optimizer. Zgodnie z planem prac, implementacja miała zostać wykonana zarówno w Pythonie, jak i Elixirze (biblioteka ***nx***) dla celów porównawczych.

Przed podjęciem się implementacji, dokonaliśmy przeglądu istniejących już i dostępnych w sieci implementacji algorytmu, co doprowadziło do następujących rezultatów:

- Autorzy artykułu *White Shark Optimizer: A novel bio-inspired meta-heuristic algorithm for global optimization problems* **nie dostarczają** gotowej implementacji algorytmu, a jedynie jego pseudokod
- Odnaleziona jedna istniejąca implementacja w środowisku MATLAB (https://www.mathworks.com/matlabcentral/fileexchange/107365-white-shark-optimizer-wso/)
- Odnaleziona implementacja w pewnych aspektach (np. implementacji aktualizacji pozycji rekina względem gromady rekinów) odbiega od pseudokodu opisu zaproponowanego w artykule *White Shark Optimizer: A novel bio-inspired meta-heuristic algorithm for global optimization problems*

Pełny kod implementacji algorytmu WSO w języku Python znajduje się w pliku ***src/wso.py***. Poniżej opisane zostały najważniejsze fragmenty implementacji.

1.1 Hiperparametry

Wartości hiperparametrów używanych w algorytmie zostały zaczerpnięte ze wspomnianego artykułu jako wartości proponowane przez autorów:

```
self.p_min = 0.5
self.p_max = 1.5
self.tau = 4.125
self.mu = 2 / abs(2 - self.tau - np.sqrt(self.tau ** 2 - 4 * self.tau))
self.f_min = 0.07
self.f_max = 0.75
self.a0 = 6.25
self.a1 = 100.0
self.a2 = 0.0005
```

Należy jednak przy tym wspomnieć, iż nie muszą to być wartości optymalne dla konkretnego problemu jakim jest optymalizacja parametrów modelu sieci wodociągowej i możliwe są pewne eksperymenty w tym zakresie.

Warto również nadmienić istnienie dodatkowych parametrów – **rozmiaru populacji** i **liczby kroków (iteracji),** które ustalane są arbitralnie i przyjmowane są na wejściu algorytmu.

1.2 Inicjalizacja populacji rekinów

Algorytm WSO jest algorytmem optymalizacyjnym operującym na ciągłej przestrzeni parametrów. Omawiana implementacja zakłada, iż optymalizowane parametry są reprezentowane jako d – wymiarowy wektor liczb zmiennoprzecinkowych. Pozycje rekinów są reprezentowane jako wspomniane wektory.

Pozycje rekinów inicjalizowane są liczbami z rozkładu jednostajnego, spełniającymi następujący warunek:

$$w = [w_1, w_2, \dots, w_d],$$

$$w_{i_{lb}} \leq w_i \leq w_{i_{ub}} \; dla \; i = 1, 2, \dots, d$$

Gdzie $w_{i_{lb}}$ wyznacza dolną granicę wartości i-tego parametru, a $w_{i_{ub}}$ – górną granicę.

```
# Step 1 - Generate initial population with respect dimensionality
# - W for shark positions
# - v for shark velocities
W = np.random.uniform(problem.lb, problem.ub, (no_sharks, problem.dim))
v = np.zeros_like(W)      # zeros_like() automatically copies dimensionality of an array
```

1.3 Ewaluacja pozycji

Poszczególne pozycje rekinów wartościowane są funkcją straty opisującą dany problem. Algorytm ma na celu minimalizowanie wartości tejże funkcji. W tym celu na wejściu algorytmu przekazywana jest abstrakcyjna reprezentacja problemu, gdzie metoda evaluate() implementuje funkcję straty:

```python
class Problem(ABC):

    def __init__(self, dim: int, lb: np.ndarray, ub: np.ndarray):
        super().__init__()

        self.dim = dim
        self.lb = lb
        self.ub = ub

    @abstractmethod
    def evaluate(self, solution: np.ndarray) -> float:
        pass
```

1.4 Iteracyjna ewolucja populacji

Główna część algorytmu polegającą na iteracyjnym przemieszczaniu rekinów w przestrzeni rozwiązań składa się z 4 głównych kroków:

1. Aktualizacja prędkości poszczególnych rekinów
2. Przemieszczanie rekinów zgodnie z zadaną prędkością lub losowe rozmieszczanie

```python
# Step 4 - update positions with wavy motion or random allocation
f = self.f_min + (self.f_max - self.f_min) / (self.f_max + self.f_min)
for i in range(no_sharks):
    a = W[i, :] > problem.ub
    b = W[i, :] < problem.lb
    w0 = np.logical_xor(a, b)
    if random.random() < mv:
        W[i][w0] = problem.ub[w0] * a[w0] + problem.lb[w0] * b[w0]
    else:
        W[i, :] += v[i, :] / f
```

3. Przemieszczanie rekinów względem gromady
4. Ewaluacja uzyskanych pozycji z uwzględnieniem poprawności rozwiązań (nie wykraczania poza ustalony obszar)
   Ponieważ docelowo fragment ten jest najbardziej czasochłonnym etapem algorytmu w kontekście uruchamiania symulacji w środowisku EPANET, podjęliśmy decyzję o **ignorowaniu** rozwiązań (rekinów) wykraczających poza dopuszczalny obszar, co zmniejszy liczbę potencjalnych wywołań symulacji.

```
# Step 6 - evaluate and update best positions
for i in range(no_sharks):
    if np.all((W[i, :] >= problem.lb) & (W[i, :] <= problem.ub)):
        fit = problem.evaluate(W[i, :])
        if fit < fitness[i]:
            W_best[i, :] = W[i, :]
            fitness[i] = fit
        if fitness[i] < fitness_min:
            fitness_min = fitness[i]
            W_gbest = W_best[i].copy()
```

1.5 Testy

W celu przetestowania poprawności algorytmu, wykorzystaliśmy go do optymalizacja znanych funkcji benchmark'owych: ***Rosenbrock*** i ***Rastrigin.*** Poniżej przedstawiono rezultaty testów dla różnych wymiarowości problemu, przy następujących parametrach (wartości w tabeli to średnie straty na przestrzeni t testów):

- **Liczba testów:** 100
- **Rozmiar populacji:** 30
- **Liczba kroków:** 100

	dim d = 2	**dim d = 5**	**dim d = 10**
Rosenbrock	0.00000273	3.00738413	28.442888
Rastrigin	0.11955835	5.30953528	20.071228

Pondato, zaimplementowany algorytm został porównany z algorytmem ***PSO (Particle Swarm Algorithm)*** w minimalizacji funkcji Rastrigin w wersji dla dwóch wymiarów i liczby testów 1000:

- **PSO:** 0.13900127907431584
- **WSO:** 0.11650775432472957

2. White Shark Optimizer - implementacja (Elixir)

To be continued…

3. Integracja z EPANET

To be continued…