Projekt nr 5 – Wykorzystanie algorytmów inspirowanych biologią do optymalizacji funkcji wielu zmiennych z wykorzystaniem biblioteki MealPy

Członkowie zespołu projektowego: Przemysław Skoczewski, Jakub Skoczewski, Paweł Malec

1. Hunger Games Search (HGO)

HGS został zaprojektowany zgodnie z działaniami powodowanymi głodem i zachowaniem behawioralnym zwierząt. Opiera się na prostej koncepcji "głodu" jako najważniejszej motywacji i przyczyny zachowań, decyzji i działań w życiu zwierząt. Te czynności są często adaptacyjne, ewolucyjne i zapewniają większe szanse na przeżycie i zdobycie pożywienia. Metoda ta posiada dynamiczny charakter, prostą konstrukcję oraz wysoką wydajność. Została porównana z obszernym zestawem 23 znanych funkcji optymalizacyjnych oraz zestawem testów porównawczych IEEE CEC 2014. Ponadto HGS został zastosowany do kilku problemów inżynieryjnych, aby zademonstrować jego przydatność. Głód jest cechą „niejedzenia przez długi czas, przy czym im silniejszy jest głód, tym silniejsze pragnienie jedzenia i tym bardziej aktywny będzie organizm w poszukiwaniu jedzenia w krótkim czasie, zanim zrobi się za późno i spowoduje głód lub śmierć. W przeciwnym razie szansa na przeżycie będzie zbyt mała, a zwierzę umrze. Stąd, gdy źródło pożywienia jest ograniczone, istnieje „gra" między głodnymi zwierzętami, aby znaleźć źródło pożywienia i wygrać. Gra opiera się więc na logicznych decyzjach i ruchach gatunków.

2. Model matematyczny

Zwierzęta towarzyskie często współpracują ze sobą podczas żerowania, ale istnieje możliwość, że nie każdy osobnik uczestniczy w takiej współpracy. Tak prezentuje się instrukcja gry:

$$\overrightarrow{X(t+1)} = \begin{cases} Game_1: \overrightarrow{X(t)} \cdot (1 + randn(1)), & r_1 < l \\ Game_2: \overrightarrow{W_1} \cdot \overrightarrow{X_b} + \vec{R} \cdot \overrightarrow{W_2} \cdot \left|\overrightarrow{X_b} - \overrightarrow{X(t)}\right|, & r_1 > l, r_2 > E \\ Game_3: \overrightarrow{W_1} \cdot \overrightarrow{X_b} - \vec{R} \cdot \overrightarrow{W_2} \cdot \left|\overrightarrow{X_b} - \overrightarrow{X(t)}\right|, & r_1 > l, r_2 < E \end{cases}$$

Gdzie R należy do zakresu [-a,a]. R1 oraz R2 reprezentują losową wartość z zakresu [0,1]. Randn(1) jest liczbą losową spełniającą rozkład normalny. T wskazuje na aktualną iterację. W1 oraz W2 przedstawiają stopień głodu. Xb reprezentuje lokalizację najlepszego osobnika w tej iteracji. X(t) reprezentuje lokalizację każdego osobnika. Współczynnik l ma na celu ulepszenie algorytmu.

Zasasy i prawa w równaniu powyżej pozwalają jednostkom na eksplorację możliwych lokalizacji w pobliżu rozwiązania optymalnego i lokalizacji oddalonych od rozwiązania optymalnego, co w pewnym stopniu gwarantuje przeszukanie wszystkich lokalizacji w granicach przestrzeni rozwiązania (dywersyfikacja).

3. Rola głodu

Model symuluje cechy glodu poszukiwanych osobników.

The formula of $\overrightarrow{W_1}$ in **Eq. (2.1)** is as follows:

$$\overrightarrow{W_1(i)} = \begin{cases} hungry(i) \cdot \dfrac{N}{SHungry} \times r_4, & r_3 < l \\ 1 & r_3 > l \end{cases}$$

The formula of $\overrightarrow{W_2}$ in **Eq. (2.1)** is shown as follows:

$$\overrightarrow{W_2(i)} = (1 - exp(-|hungry(i) - SHungry|)) \times r_5 \times 2$$

Gdzie hungry reprezentuje głód każdego osobnika. N reprezentuje ilość osobników. Shungry to suma głodu wszystkich osobników (suma z hungry). R3,4 i 5 to losowe liczby z zakresu [0,1].

Formuła dla hungry(i):

$$hungry(i) = \begin{cases} 0, & AllFitness(i) == BF \\ hungry(i) + H, & AllFitness(i)! = BF \end{cases}$$

Gdzie AllFitness(i) zachowuje przydatność każdego osobnika w bieżącej iteracji. W każdej iteracji głód najlepszego osobnika został ustawiony na 0. W przypadku innych jest dodawany na podstawie pierwotnego głodu. Stąd można zauważyć, że odpowiadające współczynnik 'H' różnych osobników będą się różniły między sobą.

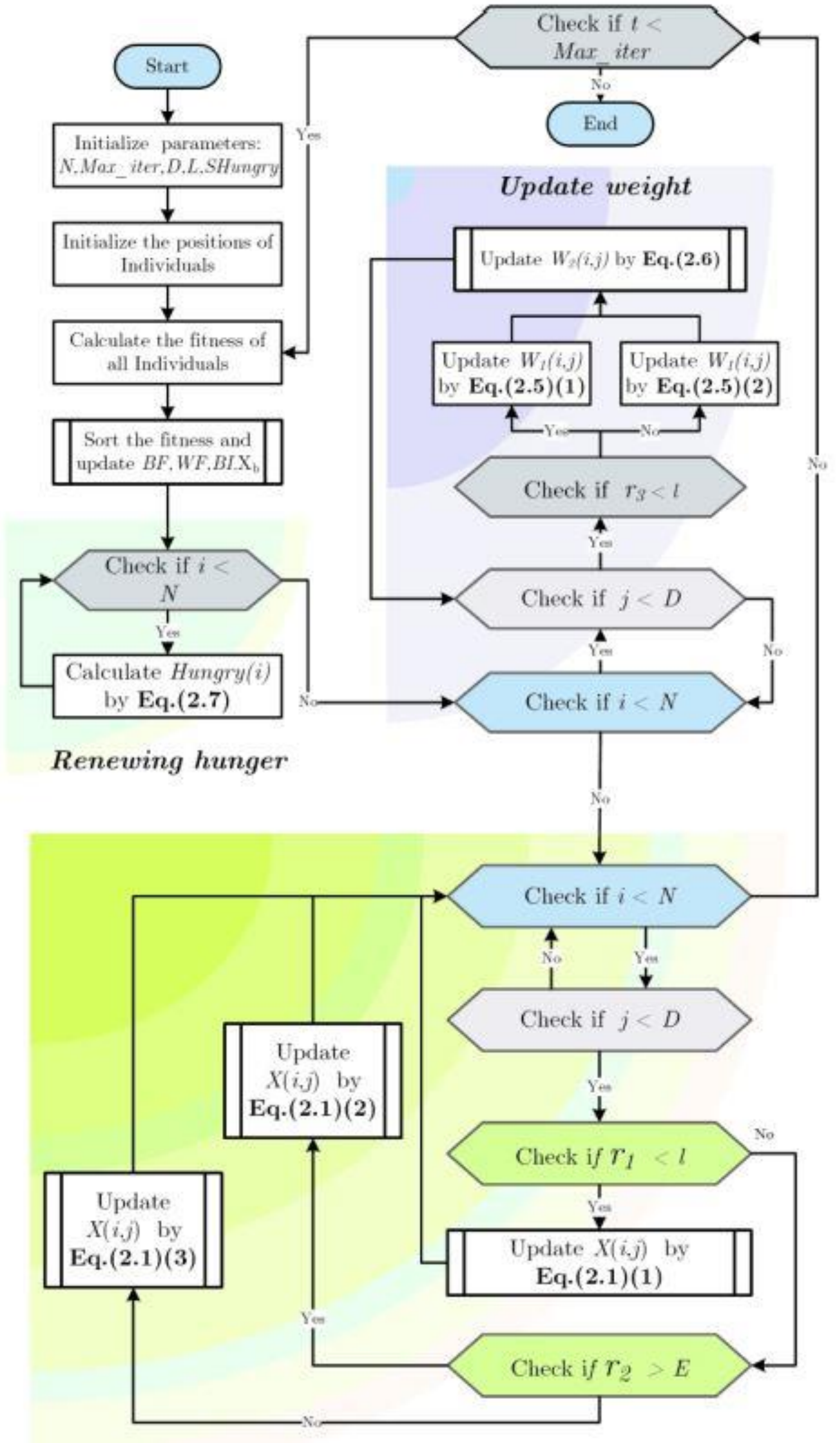
Wzór na H:

$$TH = \frac{F(i) - BF}{WF - BF} \times r_6 \times 2 \times (UB - LB)$$
$$H = \begin{cases} LH \times (1 + r), & TH < LH \\ TH, & TH \geq LH \end{cases}$$

Gdzie R6 jest liczbą losową z zakresu [0,1]. F(i) reprezentuje fitness każdego osobnika. BF to najlepsza sprawność w obecnym procesie iteracji. WF reprezentuje najgorszą sprawność w tej iteracji. UB i LB wskazuje górną i dolną granicę przestrzeni cech.

Uczucie głodu jest ograniczone do dolnej granicy. Aby algorytm działał lepiej, kontrolujemy górną i dolną granicę głodu.

Schemat blokowy algorytmu HGS:



4. Cechy HGO

Jako pozbawiony gradientów, populacyjny optymalizator HGO wykazuje wydajność dzięki następującym zaletom:

- Adaptacyjne i zmienne w czasie mechanizmy pozwalają tej metodzie obsługiwać wiele modalności i lokalne problemy optima.
- Uwzględnienie wskaźnika głodu i wpływu głodu na zakres działania sprawia, że jest bardziej on elastyczny i zdolny do zmiany wydajności.
- Zastosowanie indywidualnych wartości sprawności umożliwia uwzględnienie informacji historycznych, jeśli jest to konieczne do zmiany zachowania.
- Jest to metoda populacyjna ze stochastycznyumi elementami przełączającymi, która wzbogaca jej główne zachowania eksploracyjne i ekploatacyjne oraz elastyczność w radzeniu sobie z trudnymi problemami.
- Kod HGS jest publicznie dostępny a użytkownicy mogą łatwo uzyskać do niego dostęp

5. Wymagania do uruchomienia algorytmu

- Python 3.9
- Biblioteka mealpy

6. Algorytm

- Ustawienie górnej i dolnej granicy

```
## A1. Kiedy masz różne dolne i górne granice dla każdego parametru
problem_dict1 = {
    "obj_func": F5,
    "lb": [-3, -5, 1, -10, ],
    "ub": [5, 10, 100, 30, ],
    "minmax": "min",
    "verbose": True,
}
problem_obj1 = Problem(problem_dict1)

## A2. Jeśli masz taką samą dolną i górną granicę dla każdego parametru, wtedy możesz użyć:
##       + int lub float: następnie musisz określić rozmiar problemu / liczbę wymiarów (n_dims)
problem_dict2 = {
    "obj_func": F5,
    "lb": -10,
    "ub": 30,
    "minmax": "min",
    "verbose": True,
    "n_dims": 30,   # Pamiętaj o "n_dims"
}
problem_obj2 = Problem(problem_dict2)
```

- Przypadki zakończenia programu

```
## Istnieją 4 przypadki zakończenia:
### 1. FE (Liczba ocen funkcji)
### 2. MG (Maximum Generacji / Epok): Domyślna wartość
### 3. ES (Wczesne zatrzymanie): Same idea in training neural network (Jeśli najlepsze rozwiązanie globalne nie jest lepsze niż epsilon
###     po K epokach, następnie zatrzymać program)
### 4. TB (Granice czasowe): Chcesz aby algorytm działał K sekund. Np przy porównaniu różnych algorytmów

termination_dict1 = {
    "mode": "FE",
    "quantity": 100000  # liczba ocen funkcji
}
termination_dict2 = {  # Podczas tworzenia obiektu, będzie on nadpisywał domyślną epokę zdefiniowaną w modelu
    "mode": "MG",
    "quantity": 1000  # liczba epok
}
termination_dict3 = {
    "mode": "ES",
    "quantity": 30  # po ilu epokach jeśli wynik nie ulegnie poprawy zatrzymany zostanie program
}
termination_dict4 = {
    "mode": "ES",
    "quantity": 60  # Czas uruchomienia (przypadek 4)
}
termination_obj1 = Termination(termination_dict1)
termination_obj2 = Termination(termination_dict2)
termination_obj3 = Termination(termination_dict3)
termination_obj4 = Termination(termination_dict4)
```

- Różne tryby szkolenia

```
## + sequential: (sekwencyjny) domyślny (jeden rdzeń)
## + thread: wiele wątków w zależności od używanego CPU
## + process: wiele rdzeni do uruchomienia algorytmu

model5 = HGS.OriginalHGS(problem_dict1, epoch=100, pop_size=50, PUP=0.08, LH=10000)
model5.solve(mode='sequential')  # Default

model6 = HGS.OriginalHGS(problem_dict1, epoch=100, pop_size=50, PUP=0.08, LH=10000)
model6.solve(mode='thread')

if __name__ == "__main__":
    model7 = HGS.OriginalHGS(problem_dict1, epoch=100, pop_size=50, PUP=0.08, LH=10000)
    model7.solve(mode='process')
```

- Tworzenie wykresów

```
## Istnieje 8 różnych wykresów:
## D.1: Wartość fitness:
##       1. Global best fitness
##       2. Local best fitness
## D.2: Wartośc obiektywna:
##       3. Global objective
##       4. Local objective
## D.3: Czas pracy (dla każdej epoki)
##       5. Runtime
## D.4: Eksploracja a eksploatacja
##       6. Exploration vs Exploitation
## D.5: Różnorodność populacji
##       7. Diversity
## D.6: Wartość trajektorii (tylko 1D i 2D!)
##       8. Trajectory

model8 = HGS.OriginalHGS(problem_dict1, epoch=100, pop_size=50, PUP=0.08, LH=10000)
model8.solve()

## Dostęp do każdego można uzyskać obiektem "history":
model8.history.save_global_objectives_chart(filename="HGS/goc")
model8.history.save_local_objectives_chart(filename="HGS/loc")
model8.history.save_global_best_fitness_chart(filename="HGS/gbfc")
model8.history.save_local_best_fitness_chart(filename="HGS/lbfc")
model8.history.save_runtime_chart(filename="HGS/rtc")
model8.history.save_exploration_exploitation_chart(filename="HGS/eec")
model8.history.save_diversity_chart(filename="HGS/dc")
model8.history.save_trajectory_chart(list_agent_idx=[3, 5], list_dimensions=[3], filename="HGS/tc")
```

- Zdefiniowanie funkcji celu, ograniczeń

```
## Do obsługi wielu celów mealpy używa metody ważenia, przekształca wiele celów w jeden cel (wartość fitness)
## Zdefiniowanie funkcji celu, ograniczeń
def obj_function(solution):
    f1 = solution[0] ** 2
    f2 = ((2 * solution[1]) / 5) ** 2
    f3 = 0
    for i in range(3, len(solution)):
        f3 += (1 + solution[i] ** 2) ** 0.5
    return [f1, f2, f3]
```

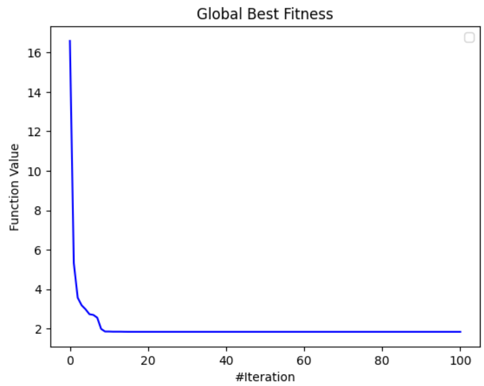
- Zdefiniowanie wag

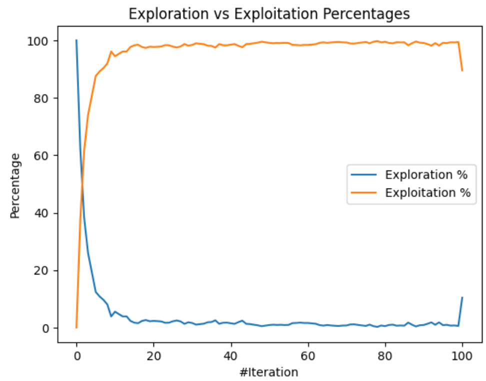
```
### f1=50%,f2=20%,f3=30% -> [0.5, 0.2, 0.3] -> wartość fitness = 0.5*f1 + 0.2*f2 + 0.3*f3
### Domyślna waga to [1, 1, 1]
problem_dict9 = {
    "obj_func": obj_function,
    "lb": [-3, -5, 1, -10, ],
    "ub": [5, 10, 100, 30, ],
    "minmax": "min",
    "verbose": True,
    "obj_weight": [0.5, 0.2, 0.3]  # pamiętaj o "obj_weight"
}
problem_obj9 = Problem(problem_dict9)
model9 = HGS.OriginalHGS(problem_obj9, epoch=100, pop_size=50, PUP=0.08, LH=10000)
model9.solve()
```

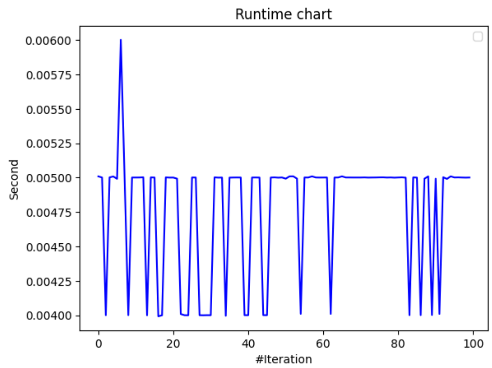
7. Przeprowadzone eksperymenty

- Przypadek 1:

Wykonanie sekwencyjne (jeden rdzeń), różne dolne i górne granice, 100 epok, populacja 50, wagi [1,1,1]

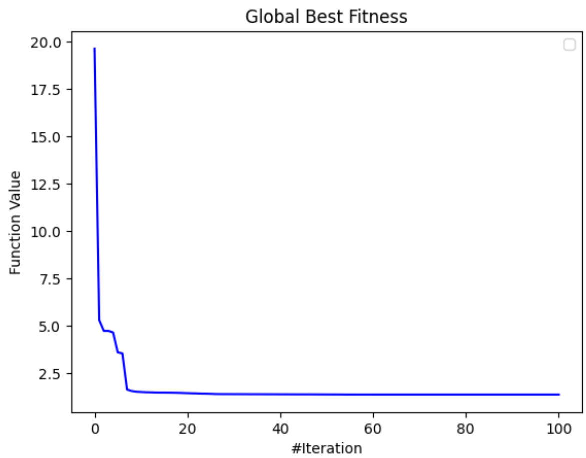


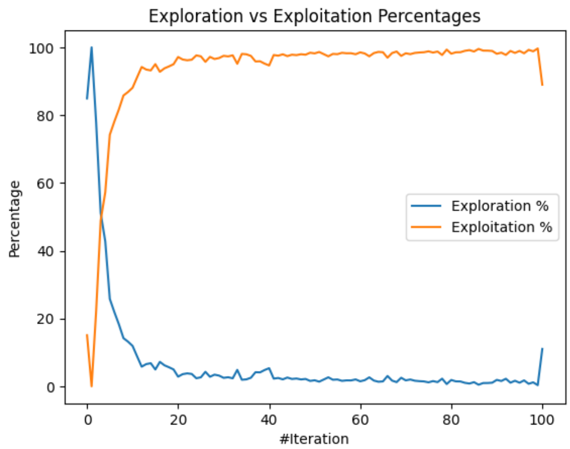


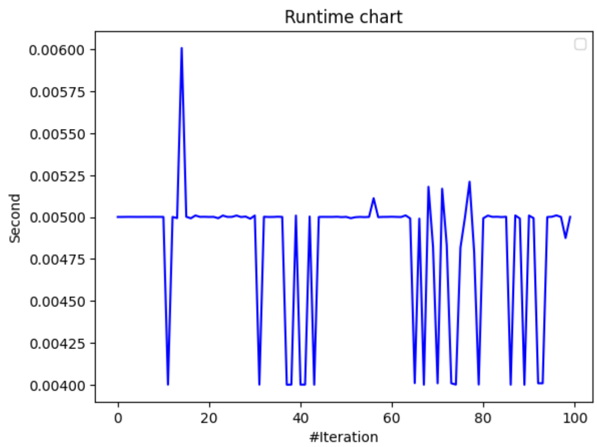


- Przypadek 2

Wykonanie sekwencyjne (jeden rdzeń), taka sama dolna i górna granica, 100 epok, populacja 50, wagi [1,1,1]

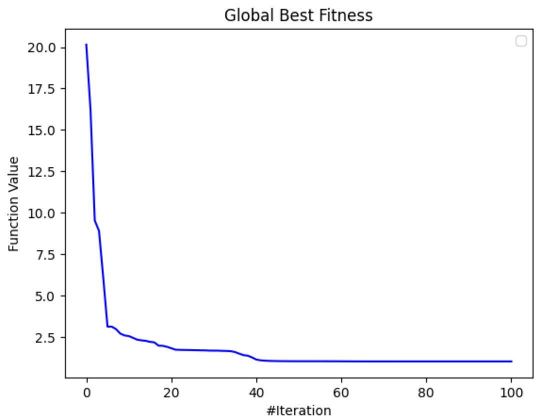


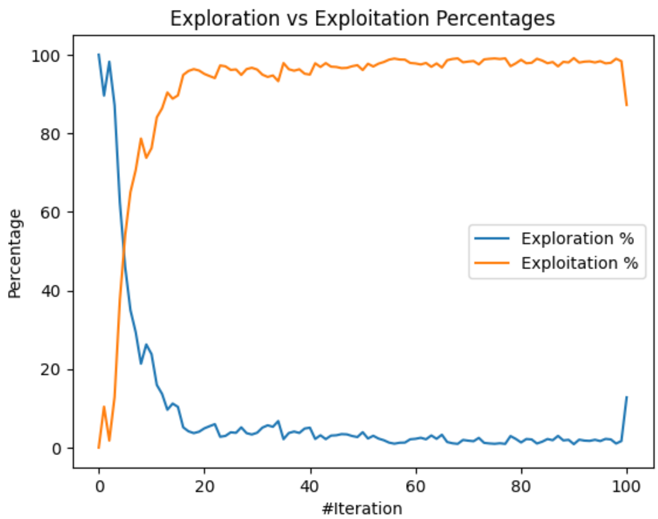


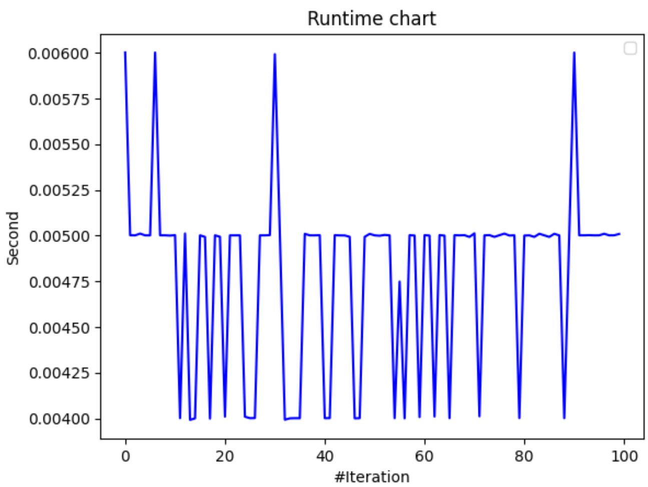


- Przypadek 3

  Wykonanie sekwencyjne (jeden rdzeń), taka sama dolna i górna granica, 100 epok, populacja 50, wagi [0.5,0.2,0.1]







8. Przykład praktyczny wykorzystania

Zadanie projektowania belek spawanych ma na celu znalezienie najniższego zużycia belek spawanych przy określonych ograniczeniach takich jak: naprężenia ścinającego, naprężenia zginającego, obciążenia wyboczeniowego i ugięcia. Problem dotyczy czterech zmiennych: grubości spoiny, długości złącza spawanego, szerokości belki oraz grubości belki. HGS porównano do HS (Harmony search) i CBO (Colliding bodies optimization).

Model matematyczny:

Consider $\vec{x} = [x_1, x_2, x_3, x_4] = [h\ l\ t\ b]$

Minimize $f(\vec{x}) = 1.10471 x_1^2 + 0.04811 x_3 x_4 (14.0 + x_4)$

Subject to $g_1(\vec{x}) = \tau(\vec{x}) - \tau_{max} \leq 0$

$$g_2(\vec{x}) = \sigma(\vec{x}) - \sigma_{max} \leq 0$$

$$g_3(\vec{x}) = \delta(\vec{x}) - \delta_{max} \leq 0$$

$$g_4(\vec{x}) = x_1 - x_4 \leq 0$$

$$g_5(\vec{x}) = P - P_C(\vec{x}) \leq 0$$

$$g_6(\vec{x}) = 0.125 - x_1 \leq 0$$

$$g_7(\vec{x}) = 1.10471 x_1^2 + 0.04811 x_3 x_4 (14.0 + x_2) - 5.0 \leq 0$$

Variable range $0.1 \leq x_1 \leq 2$, $0.1 \leq x_2 \leq 10$, $0.1 \leq x_3 \leq 10$, $0.1 \leq x_4 \leq 2$

where $\tau(\vec{x}) = \sqrt{(\tau')^2 + 2\tau'\tau'' \frac{x_2}{2R} + (\tau'')^2}$ $\tau' = \frac{P}{\sqrt{2} x_1 x_2}$ $\tau'' = \frac{MR}{J}$ $M = P\left(L + \frac{x_2}{2}\right)$

$$R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2}$$

$$J = 2\left\{\sqrt{2} x_1 x_2 \left[\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\}$$

$$\sigma(\vec{x}) = \frac{6PL}{x_4 {x_3}^2}, \quad \delta(\vec{x}) = \frac{6PL^3}{E x_3^2 x_4}$$

$$P_C(\vec{x}) = \frac{4.013E\sqrt{\frac{x_3^2 x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right)$$

$$P = 60001b, \quad L = 14 \in ..\delta_{max} = 0.25 \in ..$$

$$E = 30 \times 1^6 psi, \quad G = 12 \times 10^6 psi$$

$$\tau_{max} = 13600 psi, \quad \sigma_{max} = 30000 psi$$

Wyniki :

Algorytm	Welding seam thickness	welding joint lenght	beam width	beam thickness	Koszt optymalny
HGS	0,2015	3,328	9,0441	0,2056	1,70012
HS	0,2442	6,2231	8,2915	0,2433	2,3807
CBO	0,2434	6,2552	8,2915	0,2444	2,38411