

Sprawozdanie z realizacji projektu z przedmiotu POP

Mateusz Kiełbus
Nel Kułakowska

Użyte dane w eksperymetach znajdują się w folderze resources w plikach easy.txt, medium.txt i hard.txt. Do wyboru plansz z wymienionych plików wykorzystano ziarno generatora losowego o wartości 42. Do wygenerowania populacji początkowej dla algorytmu genetycznego wykorzystano ziarna o wartości od 0 do zadanej wielkości populacji początkowej - 1.

Do uruchomienia skrypu z eksperymetami w formacie .ipynb należy pobrać zależności zawarte w pliku requirements.txt.

Do utworzenia plansz wykorzystaliśmy generator z repozytorium: <https://github.com/robatron/sudoku.js> i utworzyliśmy po 10 plansz dla wyżej wymienionych poziomów trudności.

Opisu problemu

"Sudoku (jap. 数独 sūdoku; od sūji wa dokushin ni kagiru, czyli cyfry muszą być pojedyncze) – łamigłówka, której celem jest wypełnienie diagramu 9×9 w taki sposób, aby w każdym wierszu, w każdej kolumnie i w każdym z dziewięciu pogrubionych kwadratów 3×3 (zwanych „blokami” lub „podkwadratami”) znalazło się po jednej cyfrze od 1 do 9"

cytat z: <https://pl.wikipedia.org/wiki/Sudoku>

Problem Sudoku jest problemem optymalizacji, gdzie poszukujemy poprawnego i pełnego uzupełnienia planszy z uwzględnieniem pewnych ograniczeń dla danej planszy. Dla jednej planszy praktycznie zawsze mamy tylko 1 rozwiązanie, więc nie będziemy szukać ich więcej.

Opis sposobu rozwiązania problemu

W naszym projekcie planujemy porównać dwa różne algorytmy optymalizacyjne do rozwiązywania Sudoku: algorytm genetyczny i algorytm ACO (Ant Colony Optimization).

Algorytm genetyczny:

Algorytm ten będzie operował na populacjach plansz Sudoku i będzie ewoluował rozwiązania, wykorzystując operatory selekcji, krzyżowania i mutacji. Założenia:

- Kolejne osobniki będziemy przedstawiać jako wypełnienie podanej planszy 9×9 - pola na planszy będziemy kodować przy pomocy krotek (*wartość, stała*), gdzie wartość to jedna z cyfr 1-9 (przedstawia wartość na planszy), a stała to stała wartość boolowska (True = pole podane na początku, którego nie można zmienić, False = pole, którego wartość przewidujemy). Uznaliśmy, że kodowanie binarne nie ma większego sensu.
- Osobniki będą inicjalizowane wiersz po wierszu, z założeniem, że wiersz musi być poprawny.

- Krzyżowanie nastąpi zamieniając odpowiednią ze względu na prawdopodobieństwo krzyżowania liczbę wierszy ze sobą (te wiersze mają takie same położenia na planszy). Mutacja = zamienianie losowo z prawdopodobieństwem pm dwóch pól (niewypełnione w oryginalnej planszy) na planszy.
- Do selekcji wykorzystamy selekcję turniejową z różnymi rozmiarami turniejów.
- Funkcja celu: będziemy minimalizować funkcję celu. Celem jest otrzymanie liczby 0, która będzie oznaczać poprawne i pełne rozwiązanie. Będzie ona równa liczbie konfliktów na planszy.

Hiperparametry:

- maksymalna liczba iteracji
- wielkość populacji
- stałe w selekcji
- prawdopodobieństwo krzyżowania i mutacji

Algorytm ACO:

Algorytm ACO inspirowany zachowaniami kolonii mrówek będzie używany do eksploracji przestrzeni rozwiązań. Mrówki będą poruszały się po planszy i pozostawiały feromony na odwiedzonych polach w zależności od jakości rozwiązania. Założenia:

- "sąsiedzi" pola to pola w tej samej kolumnie/wierszu/kwadracie
- plansza będzie przedstawiana jako dwuwymiarowa tablica wypełniona *możliwymi wartościami* (zbiorem liczb z przedziału 1-9, np. {6}, {1,2,5})
- "rozwiążane" pole, to takie, dla którego mamy tylko 1 możliwą wartość. Gdy jakieś pole jest "rozwiążane" dokonujemy usunięcia wartości tego pola z możliwych wartości sąsiadów
- "zepsute" pole, to takie, dla którego mamy 0 możliwych wartości.
- funkcja celu (przy wybieraniu najlepszej mrówki) - będzie musiała być inna niż w algorytmie genetycznym - będzie to liczba pól - liczba rozwiązań pól (minimalizacja, która dąży do wartości = 0, maksymalna wartość = 81)

Algorytm w skrócie:

1. uaktualnij możliwe wartości na planszy
2. zainicjalizuj globalną macierz feromonów
3. dopóki niespełniony warunek stopu:
 - A. każda mrówka dostaje własną kopię planszy
 - B. dla każdej mrówki wylosuj pozycję początkową
 - C. dla liczby pól:
 - a. dla każdej mrówki:
 - i. jeśli pole nie jest rozwiązane:
 1. ustaw jedną z możliwych wartości
 2. zaktualizuj możliwe wartości u sąsiadów
 3. zaktualizuj macierz feromonów mrówki przejdź do kolejnego pola
 - D. znajdź najlepszą mrówkę
 - E. zaktualizuj globalną macierz feromonów
 - F. "wyparowywanie" najlepszej wartości (aby się nie zablokować w jednym rozwiążaniu)

Hiperparametry:

- liczba mrówek
- globalny i lokalny współczynnik aktualizacji macierzy feromonów

- współczynnik parowania feromonów
- współczynnik łakomstwa (przy selekcji wartości dla pola)

Uwaga

Przy poniższych testach należy pamiętać że funkcja celu w ACO /= funkcji celu w algorytmie genetycznym

Poniżej przygotowujemy potrzebne biblioteki i zmienne

```
In [2]: from genethic_algorithm.genethic_algorithm import genethicAlgorithm
from ACO.Solver import Solver
from ACO.utils import create_board_from_str
from representation import makeTable, printTable
import numpy as np
from statsmodels.distributions.empirical_distribution import ECDF
from genethic_algorithm.board import makeBoardsFromFile
from itertools import product
from random import choice, seed
from matplotlib import pyplot as plt
import seaborn as sns

easyBoards = ["952.716843184..7.5476.8.319.24738...781.59432.39124857...91754.1978452638
"98.317...3.4265891.12.9837...69812.77.9.42..8821.73..91978365245431.9786268754913",
"13.769.4.46.581.7375.234.16683.92.5.271.536899456783215.794613839481..6.81.32..9.",
"12768.9.5.581.2.374..9.721.7.943.8523..951.45.6218..323486157989157.326675329481",
"3..54.698649.732158.5.91.3.953268.714829.73.67.63...825684391.71.472.86.297186543",
".6.75...3713.9..658253..9746.12893473486175299.2543..61978.645223.1..69.586924731",
"318.9.524.4.583197597.1.683925.7.431173945268864.3.975.3..5.81.75..6.34.481329756",
"5384.6729..497285372.3.81.4..62349.5.95.673.23.2.954.6..75492319.372.648241683597",
".2..48.7174..192838.132746926.751.48.75482.1.41.96.752.8.2.4695.341.5827592876134",
"134576928759.2.631268913547..6.5...4..7.6...54.513...654378216.671395482982641753"]
mediumBoards = [".63.71.5..516.4.3..7...3.1..251.938.618342.75..9758162.9683...118742659
"4.8...9.2...24.8.115.896374.8..2.6435143697283.6478519....248724....3687.6.4295",
"351.9.7.2.7.1.35866...2513..37.1265..1..569...65...3.17832614.51245.98.7.9.487213",
"3.9..17..4517839626782..13.5.412769..82..6.1.196348257867....1..36....9..5812376",
"51734.974928.16331896452718...7.9..7..98.5195....748..4.31.7.59.7.6834.37.49.1.",
"793856124648721935251...8687.1635425164928733.4578691...6.7458.....",
"46...7..317...3....294.6..7517362..9.325487168461795..293781..57516342.8.84..5..1",
"6.....24.65.1...1.....26857194.3156497827498.31..657..8..4192436578483257691",
"6..95.7....976....57.32.9.6...5391679631724..1574863927..613..9...8976.3396245.71",
".9..16..85312847..486973251.654318.232.69851.81.7256...4815....5.3...8..738.9.25"]
hardBoards = ["8321965.4164275..579843..13.87.1..216439785..7.821.3..1.....3.1.....
"7...3..5....74.383...86.....34.1782247398..61..72.39....63....6..14.8.3431857269",
"8.4..6...265.7..94.....6...12643...9382....6.46918..3423.8.9616.7139...189462...",
".6....4.....871..2..4.93....87.261.7..12394...46.587.8.354179..71968...1.287643",
"6.75421..231..6455.4...27.91...5..87...451.45.1.....294..867748926351.65....",
".85937.1....26.5.....51....5.3679.48...852.6.678341..7521968..819423.....785...",
"524.8.....1..52....8.694.4296..158..6.18492..1924.....736.2842.348.6...4..9.315",
".63519..28.5..3..49..684513...865.....4.7..55.41.2....4.35.2.6238946157.5.27....",
"92864.7....692....5.87.6.283.469....61.5.83959213847.21..8....7.5.94.6.64..1....",
"597.4..6..86.9.....4...6..967913....418.2.936...96..7.8254.36.7964.7....731689..5"]
custom_palette = sns.color_palette("hls", 13)
bestArgs = {}
convergence_data_genethic = {}
seed(42)
boardStrs = [choice(easyBoards), choice(mediumBoards), choice(hardBoards)]
seed(None)
bestArgsACO = {}
convergence_data_ACO = {}
```

Testy Hiperyparametrów

Utworzono zostały wykresy zbieżności dla obu algorytmów dla różnych rodzajów plansz (czyli o różnym poziomie trudności).

Eksperymenty różnią się hiperparametrami. Wykonujemy testy dla wszystkich kombinacji wybranych wartości i następnie wybieramy najlepsze 10 (1. ze względu na wartość rozwiązania, 2. liczbę ewaluacji/iteracji).

Przy algorytmie genetycznym wystarczyło wykonać badania i krzywe ze względu na iteracje, zaś dla algorytmu mrówkowego należało wykonać je ze względu na ewaluacje (inaczej za mało się poszczególne uruchomienia od siebie różniły). Na podstawie tych badań wybraliśmy hiperparametry do następnych testów.

Zestaw parametrów na wykresach algorytmu genetycznego to odpowiednio : [rozmiar populacji, maksymalna liczba iteracji, rozmiar turnieju, prawdopodobieństwo krzyżowania, prawdopodobieństwo mutacji]. Nie ustalano maksymalnej liczby wywołań funkcji celu.

Zestaw parametrów na wykresach algorytmu ACO to odpowiednio : [liczba mrówek, globalny współczynnik aktualizacji macierzy feromonów, lokalny współczynnik aktualizacji macierzy feromonów, łakomstwo, współczynnik parowania]. Nie ustalano maksymalnej liczby wywołań funkcji celu, maksymalna liczba iteracji została ograniczona do 100. To samo znajduje się w pliku tune_hyperparameters.py

Genetyczny

In [19]:

```
pop0Sizes = [50, 75, 100]
maxIterParams = [50, 200, 300]
tournamentSizeParams = [50, 60, 70, 80, 90]
crossoverProbParams = [0.1, 0.2, 0.3, 0.4]
mutationProbParams = [0.2, 0.3, 0.4, 0.5]

bestMaxIter = None
bestTournamentSize = None
bestCrossoverProb = None
bestMutationProb = None
generalBestScore = float("inf")
currentBestScore = None

argsLists = list(product(pop0Sizes, maxIterParams, tournamentSizeParams, crossoverProbPa

for index, argTuple in enumerate(argsLists):
    argsLists[index] = list(argTuple)

iterations = None

for boardStr, boardType in zip(boardStrs, ["easy", "medium", "hard"]):
    iterations_data = {}
    for experimentNumber, argsList in enumerate(argsLists):
        pop0Size = argsList.pop(0)
        pop0 = makeBoardsFromFile(boardStr, pop0Size=pop0Size, random_state=[iteration for _ in range(len(argsList))])
        bestScore, bestSollution, scores, evaluations = genethicAlgorithm(*argsList)
        iterations = len(scores)
        print(f"Status ukończenia testów {round(experimentNumber/len(argsLists)*100, 2)}%")
        currentBestScore = bestScore
        argsList.pop(0)
        argsList.insert(0, pop0Size)
        iterations_data[experimentNumber] = {
            "iterations": iterations,
            "scores": scores,
            "args": argsList
        }
    if not bestMaxIter and not bestTournamentSize and not bestCrossoverProb and not bestMutationProb:
        bestMaxIter = bestTournamentSize = bestCrossoverProb = bestMutationProb = currentBestScore
```

```

bestMaxIter = argsList[1]
bestTournamentSize = argsList[2]
bestCrossoverProb = argsList[3]
bestMutationProb = argsList[4]
else:
    if currentBestScore < generalBestScore:
        generalBestScore = currentBestScore
        bestMaxIter = argsList[1]
        bestTournamentSize = argsList[2]
        bestCrossoverProb = argsList[3]
        bestMutationProb = argsList[4]
if currentBestScore == 0:
    generalBestScore = currentBestScore
    print(f"Status ukończenia testów: 100%")
    break
print(f"Najlepsze znalezione rozwiązanie: {generalBestScore} przy {iterations} iteracjach dla planszy easy")
print(f"Wybrane parametry algorytmu genetycznego dla planszy easy: rozmiar populacji: 50, maksymalna liczba iteracji: 50, rozmiar turnieju: 50, prawdopodobieństwo krzyżowania: 0.1, prawdopodobieństwo mutacji: 0.2")
print(f"Status ukończenia testów: 100%")
sorted_iterations_data = sorted(iterations_data.items(), key = lambda x: (x[1]["score"] - generalBestScore) / generalBestScore)
convergence_data_genetic[boardType] = sorted_iterations_data[:10]

for boardType, data in convergence_data_genetic.items():
    sns.set_palette(custom_palette[:10])
    iterations_data = [iter_data["iterations"] for _, iter_data in data]
    scores_data = [iter_data["scores"] for _, iter_data in data]
    args_data = [iter_data["args"] for _, iter_data in data]
    plt.figure(figsize=(15, 10))
    for i in range(len(iterations_data)):
        iterations = np.arange(1, iterations_data[i]+1)
        plt.plot(iterations, scores_data[i], label=str(args_data[i]), alpha=0.6)

    plt.xlabel('Liczba iteracji')
    plt.ylabel('Najlepsza wartość funkcji celu')
    plt.title(f'Wykres zbieżności dla algorytmu genetycznego dla planszy o trudności: {boardType}')
    plt.legend(ncol =3)
    plt.show()

```

Status ukończenia testów: 100%

Najlepsze znalezione rozwiązanie: 0 przy 20 iteracjach dla planszy easy

Wybrane parametry algorytmu genetycznego dla planszy easy: rozmiar populacji: 50, maksymalna liczba iteracji: 50, rozmiar turnieju: 50, prawdopodobieństwo krzyżowania: 0.1, prawdopodobieństwo mutacji: 0.2

Status ukończenia testów: 100%

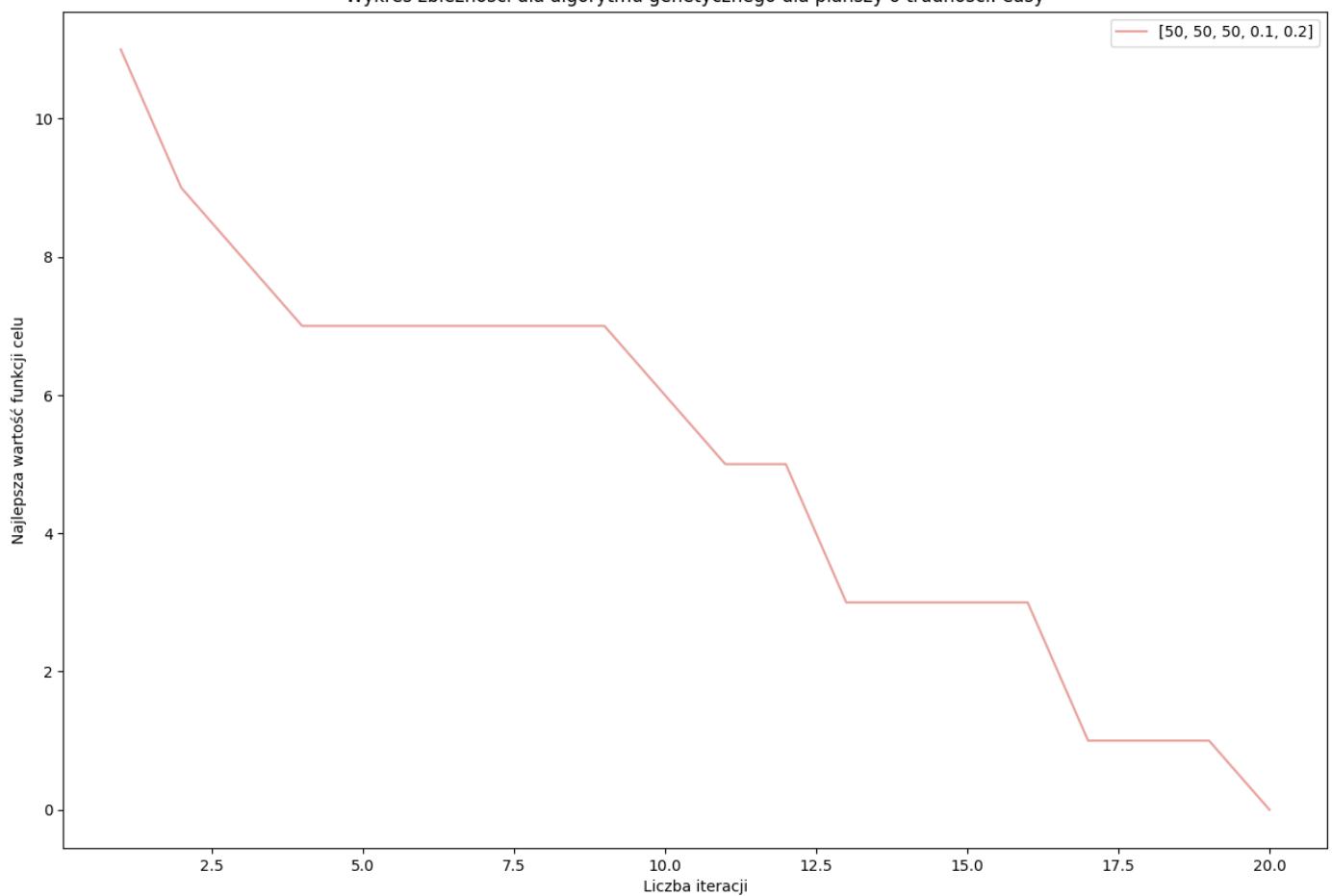
Najlepsze znalezione rozwiązanie: 0 przy 33 iteracjach dla planszy medium

Wybrane parametry algorytmu genetycznego dla planszy medium: rozmiar populacji: 50, maksymalna liczba iteracji: 200, rozmiar turnieju: 50, prawdopodobieństwo krzyżowania: 0.1, prawdopodobieństwo mutacji: 0.2

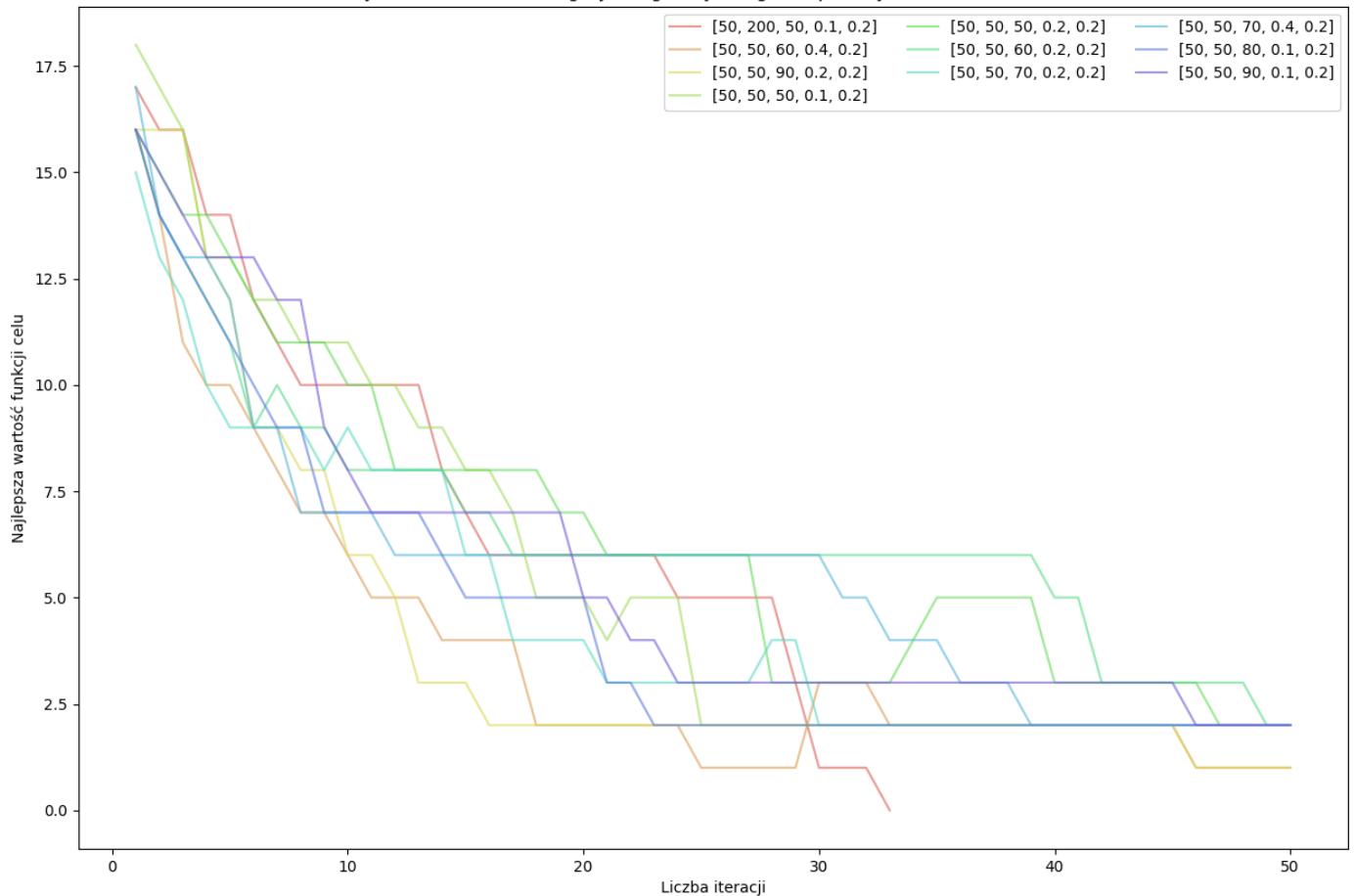
Najlepsze znalezione rozwiązanie: 1 przy 300 iteracjach dla planszy hard

Wybrane parametry algorytmu genetycznego dla planszy hard: rozmiar populacji: 100, maksymalna liczba iteracji: 300, rozmiar turnieju: 80, prawdopodobieństwo krzyżowania: 0.1, prawdopodobieństwo mutacji: 0.2

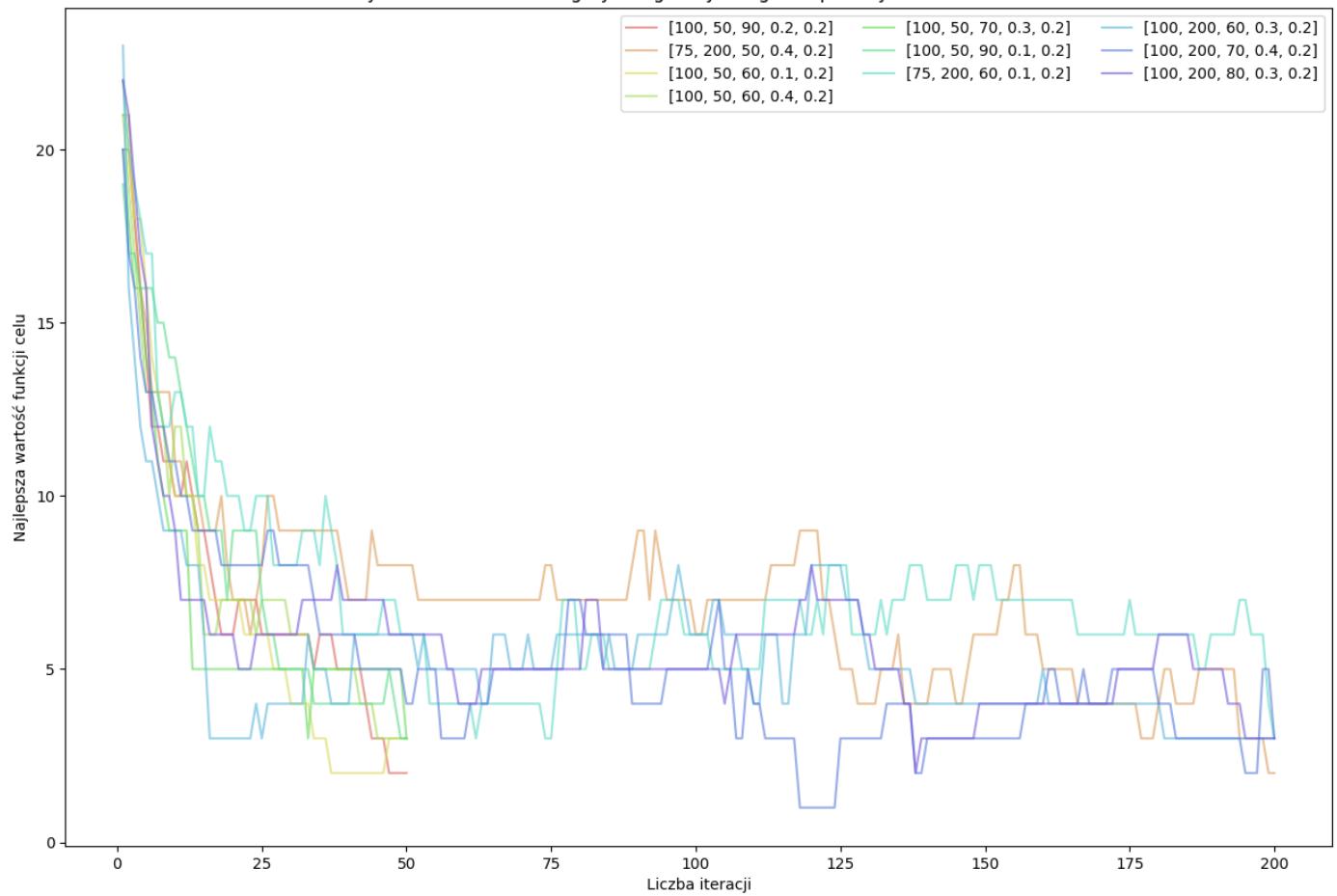
Wykres zbieżności dla algorytmu genetycznego dla planszy o trudności: easy



Wykres zbieżności dla algorytmu genetycznego dla planszy o trudności: medium



Wykres zbieżności dla algorytmu genetycznego dla planszy o trudności: hard



In [20]:

```

ants_number = [1, 2, 50, 100]
global_pher_update = [0.1, 0.5, 0.7, 0.9]
local_pher_update = [0.1, 0.3, 0.5, 0.9]
greedines = [0.1, 0.5, 0.9]
evaporation_parameter = [0.005, 0.1, 0.5, 0.9]

argsLists = list(product(ants_number, global_pher_update, local_pher_update, greedines,
for index, argsTuple in enumerate(argsLists):
    argsLists[index] = list(argsTuple)

for boardStr, boardType in zip(boardStrs, ["easy", "medium", "hard"]):
    best_ants_num = None
    best_global_pher_update = None
    best_local_pher_update = None
    best_greedines = None
    best_evaporation_parameter = None
    best_evaluations = float("inf")
    generalBestScore = float("inf")
    evaluations_data = {}

    for experimentNumber, argsList in enumerate(argsLists):
        solver = Solver(*argsList, max_iterations=100)
        board = create_board_from_str(boardStr)
        bestScore, bestSollution, scores, evaluations, iter = solver.solve(board)
        print(f"Status ukończenia testów {round(experimentNumber/len(argsLists)*100, 2)}")

        evaluations_data[experimentNumber] = {
            "evaluations": evaluations,
            "scores": scores,
            "args": argsList
        }

        if not best_ants_num or bestScore < generalBestScore or (bestScore == generalBes

```

```

best_ants_num = argsList[0]
best_global_pher_update = argsList[1]
best_local_pher_update = argsList[2]
best_greedines = argsList[3]
best_evaporation_parameter = argsList[4]
generalBestScore = bestScore
best_evaluations = evaluations

print(f"Najlepsze Znalezione rozwiązanie: {generalBestScore} przy {iter} iteracjach")
print(f"Wybrane parametry algorytmu ACO dla planszy {boardType}: liczba mrówek: {bes
bestArgsACO[boardType] = [best_ants_num, best_global_pher_update, best_local_pher_up
sorted_evaluations_data = sorted(evaluations_data.items(), key = lambda x: (x[1][ "sc
convergence_data_ACO[boardType] = sorted_evaluations_data[:10]

for boardType, data in convergence_data_ACO.items():
    sns.set_palette(custom_palette[:10])
    evaluations_data = [eval_data["evaluations"] for _, eval_data in data]
    scores_data = [eval_data["scores"] for _, eval_data in data]
    args_data = [eval_data["args"] for _, eval_data in data]
    plt.figure(figsize=(15, 10))
    for i in range(len(evaluations_data)):
        evaluations = np.arange(1, evaluations_data[i]+1)
        plt.plot(evaluations, scores_data[i], label=str(args_data[i]), alpha=0.6)

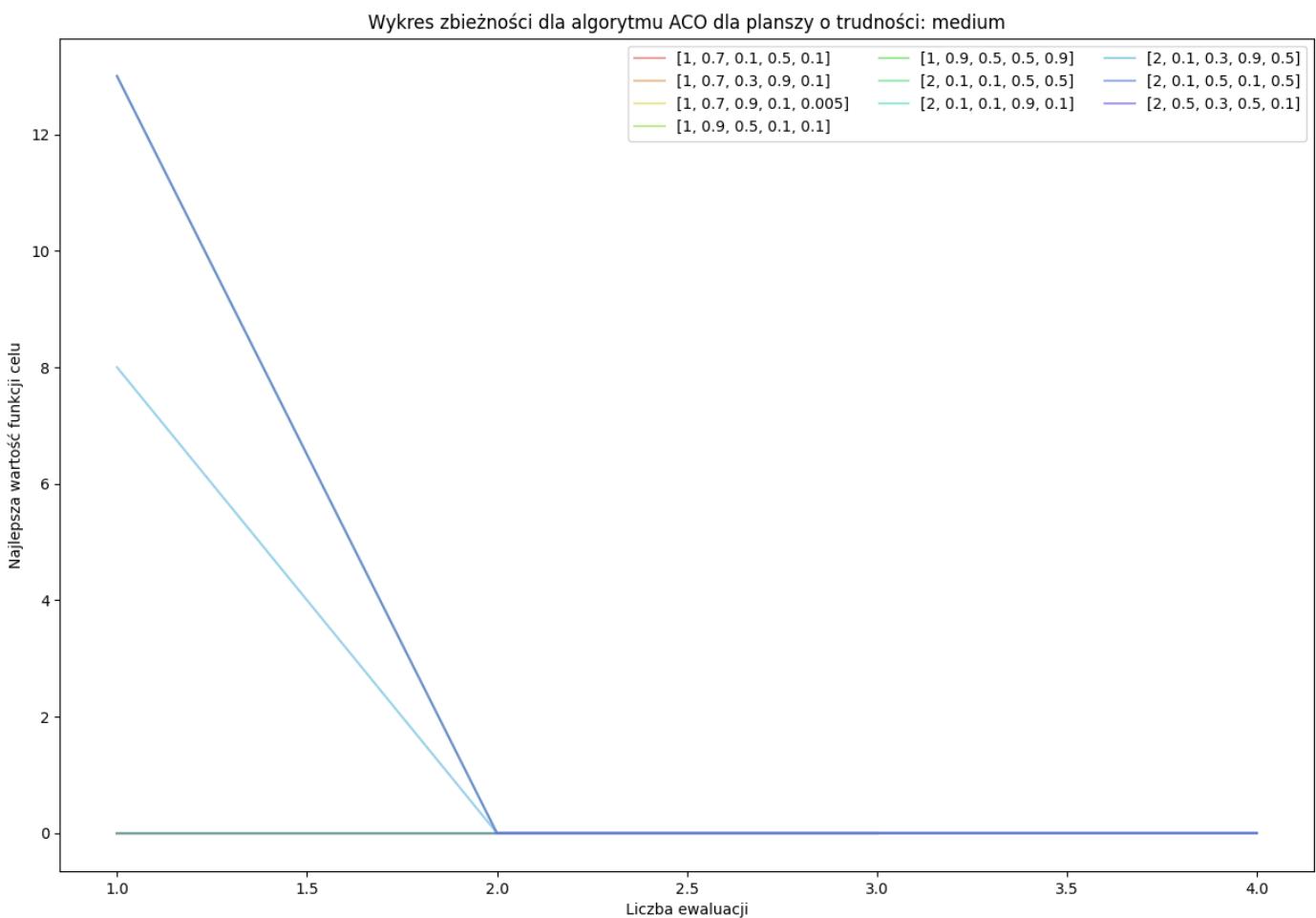
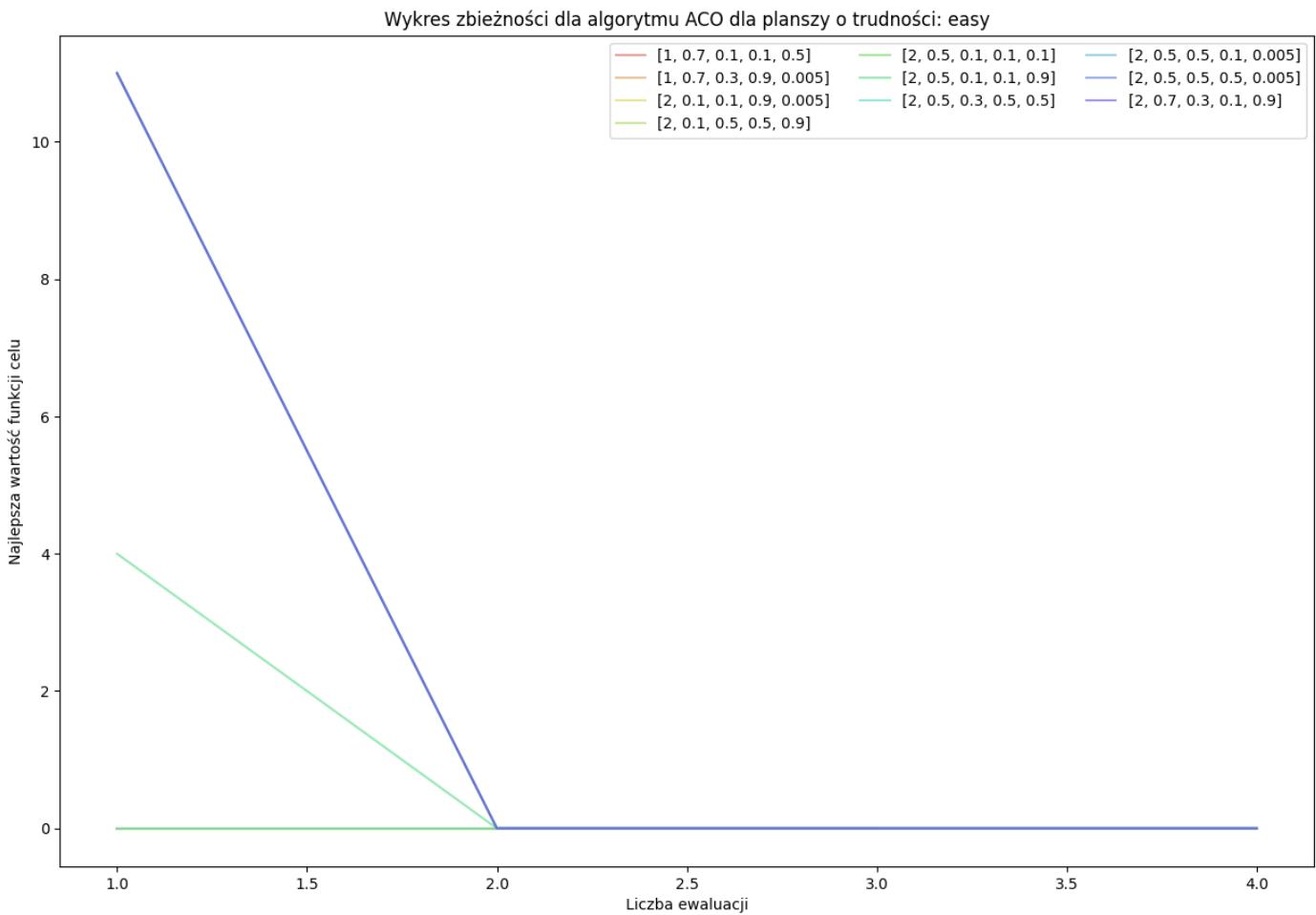
    plt.xlabel('Liczba ewaluacji')
    plt.ylabel('Najlepsza wartość funkcji celu')
    plt.title(f'Wykres zbieżności dla algorytmu ACO dla planszy o trudności: {boardType}')
    plt.legend(ncol =3)
    plt.show()

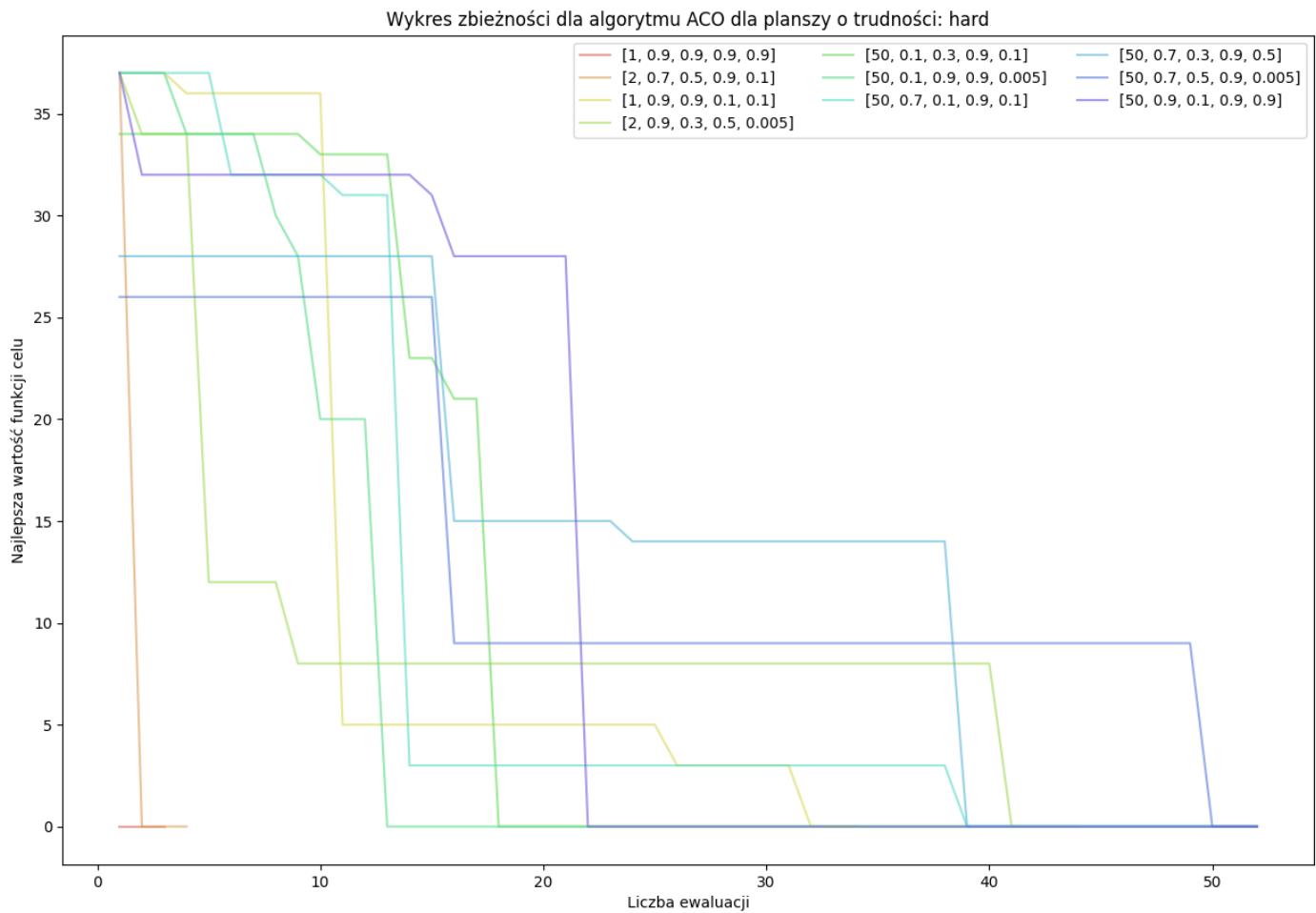
```

Najlepsze Znalezione rozwiązanie: 0 przy 1 iteracjach dla planszy easy
 Wybrane parametry algorytmu ACO dla planszy easy: liczba mrówek: 1, globalne odświeżanie feromonu: 0.7, lokalne odświeżanie feromonu: 0.1, współczynnik zachłanności: 0.1, wspólnik parowania feromonu: 0.5

Najlepsze Znalezione rozwiązanie: 0 przy 1 iteracjach dla planszy medium
 Wybrane parametry algorytmu ACO dla planszy medium: liczba mrówek: 1, globalne odświeżanie feromonu: 0.7, lokalne odświeżanie feromonu: 0.1, współczynnik zachłanności: 0.5, współczynnik parowania feromonu: 0.1

Najlepsze Znalezione rozwiązanie: 0 przy 14 iteracjach dla planszy hard
 Wybrane parametry algorytmu ACO dla planszy hard: liczba mrówek: 1, globalne odświeżanie feromonu: 0.9, lokalne odświeżanie feromonu: 0.9, współczynnik zachłanności: 0.9, wspólnik parowania feromonu: 0.9





Już tutaj możemy zauważyć, że algorytm ACO sobie zdecydowanie lepiej radzi od algorytmu genetycznego. Zdołał znaleźć rozwiązanie wiele razy dla planszy "hard" podczas gdy genetyczny ani razu. W dodatku korzystanie z niego wiąże się z mniejszym kosztem obliczeniowym (mniej iteracji, wywołań funkcji celu...).

Algorytm genetyczny radzi sobie lepiej przy większej ilości iteracji oraz większej populacji, czego mogliśmy się spodziewać. Jednakże radzi sobie lepiej też gdy prawdopodobieństwo krzyżowania i mutacji jest niskie, co może być niespodziewane, przez to właśnie algorytm ten szybko utyka w optimum lokalnym dla bardziej trudnych plansz. Algorytm ACO nie potrzebuje dużej liczby mrówek, 1-2 starczają. Rekompensuje to sobie z pomocą wysokiego współczynnika globalnego odświeżania feromonu. Lokalny współczynnik różni się w zależności od poziomu trudności, chociaż dla łatwiejszych nie wydaje się mieć dużego znaczenia. Dla bardziej trudnych plansz współczynnik ten występuje tylko przy liczbie mrówek = 1, gdzie tak naprawdę działa jak globalny. Inaczej jest niski, co jest zgodne ze standardowymi wartościami. Podobnie przy liczbie mrówek=1 współczynnik parowania jest duży, aby ta 1 mrówka nie utykała w optimum lokalnym. Przy większej ilości mrówek tenże współczynnik maleje (bo tym razem przed utknieniem próbuje zabezpieczyć większość mrówek).

Testy szybkości przy zadanej maksymalnej liczbie iteracji

Utworzymy wykresy ECDF dla wybranych maksymalnych liczb iteracji.

Utworzymy też tabelki, gdzie dla każdego eksperymentu policzymy ilość rozwiązań ("Znaleziono najlepsze rozwiązania"). To samo znajduje się w pliku max_iterations_ecdf.py

In [10]:

```
if not bestArgs:
    bestArgs["easy"] = [50, 50, 50, 0.1, 0.2]
    bestArgs["medium"] = [50, 200, 50, 0.1, 0.2]
    bestArgs["hard"] = [100, 300, 80, 0.1, 0.2]
```

```

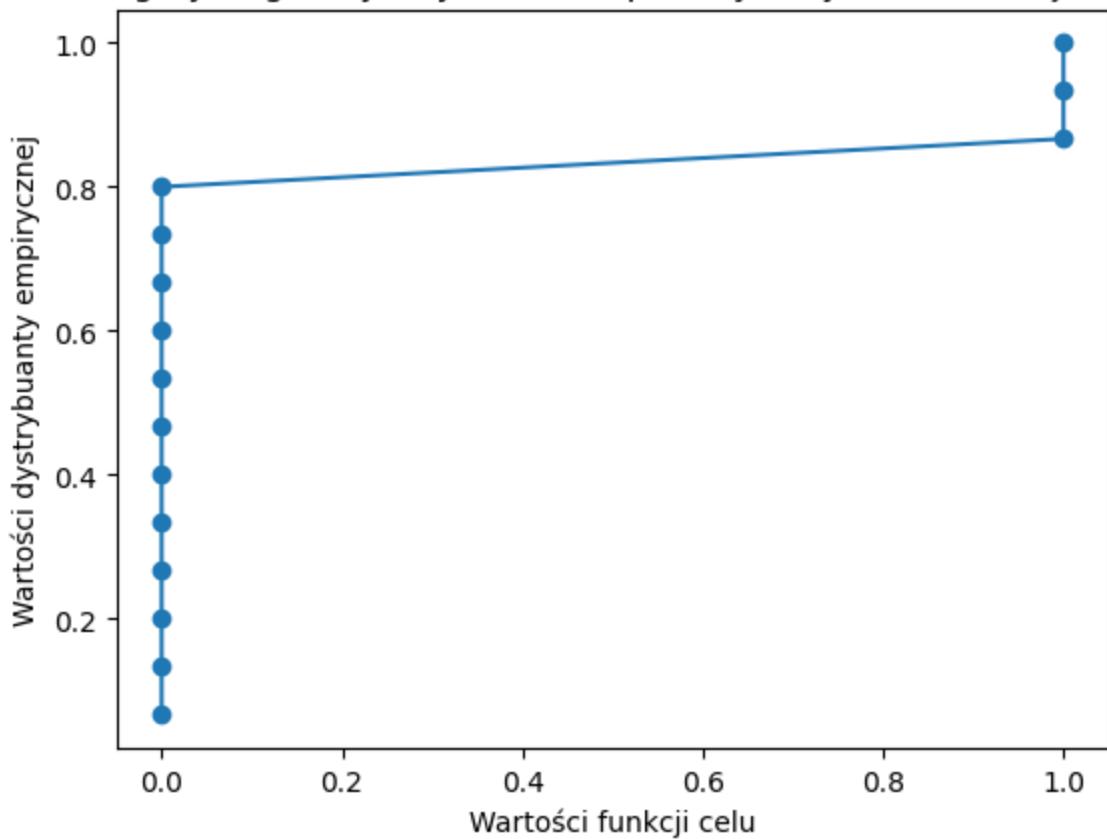
if not bestArgsACO:
    bestArgsACO["easy"] = [1, 0.7, 0.1, 0.1, 0.5]
    bestArgsACO["medium"] = [1, 0.7, 0.1, 0.5, 0.1]
    bestArgsACO["hard"] = [1, 0.9, 0.9, 0.9, 0.9]
iterations = [75, 100, 150, 200, 300]

foundBestScores = []
foundBestACOScores = []
for boardStr, boardType in zip(boardStrs, ["easy", "medium", "hard"]):
    for maxIterations in iterations:
        bestScores = []
        bestACOScores = []
        for experimentNumber in range(15):
            pop0 = makeBoardsFromFile(boardStr, pop0Size=bestArgs[boardType][0], random_
                argsList = [pop0, maxIterations]
                argsList.extend(bestArgs[boardType][2:]))
            bestScore, bestSollution, scores, evaluations = genethicAlgorithm(*argsList)
            bestScores.append(bestScore)
            solver = Solver(*bestArgsACO[boardType], max_iterations=maxIterations)
            board = create_board_from_str(boardStr)
            bestScore, bestSollution, scores, evaluations, iter = solver.solve(board)
            bestACOScores.append(bestScore)
        ecdf = ECDF(bestScores)
        plt.plot(ecdf.x, ecdf.y, marker='o', linestyle='--')
        plt.xlabel("Wartości funkcji celu")
        plt.ylabel('Wartości dystrybuanty empirycznej')
        plt.title(f"Algorytm genetyczny ECDF dla planszy {boardType} liczba iteracji {ma
        plt.show()
        numberOfBestScores = len([value for value in bestScores if value == 0])
        foundBestScores.append([numberOfBestScores, boardType, maxIterations])
        ecdf = ECDF(bestACOScores)
        plt.plot(ecdf.x, ecdf.y, marker='o', linestyle='--')
        plt.xlabel("Wartości funkcji celu")
        plt.ylabel('Wartości dystrybuanty empirycznej')
        plt.title(f"ACO ECDF dla planszy {boardType} liczba iteracji {maxIterations}")
        plt.show()
        numberOfBestScores = len([value for value in bestACOScores if value == 0])
        foundBestACOScores.append([numberOfBestScores, boardType, maxIterations])

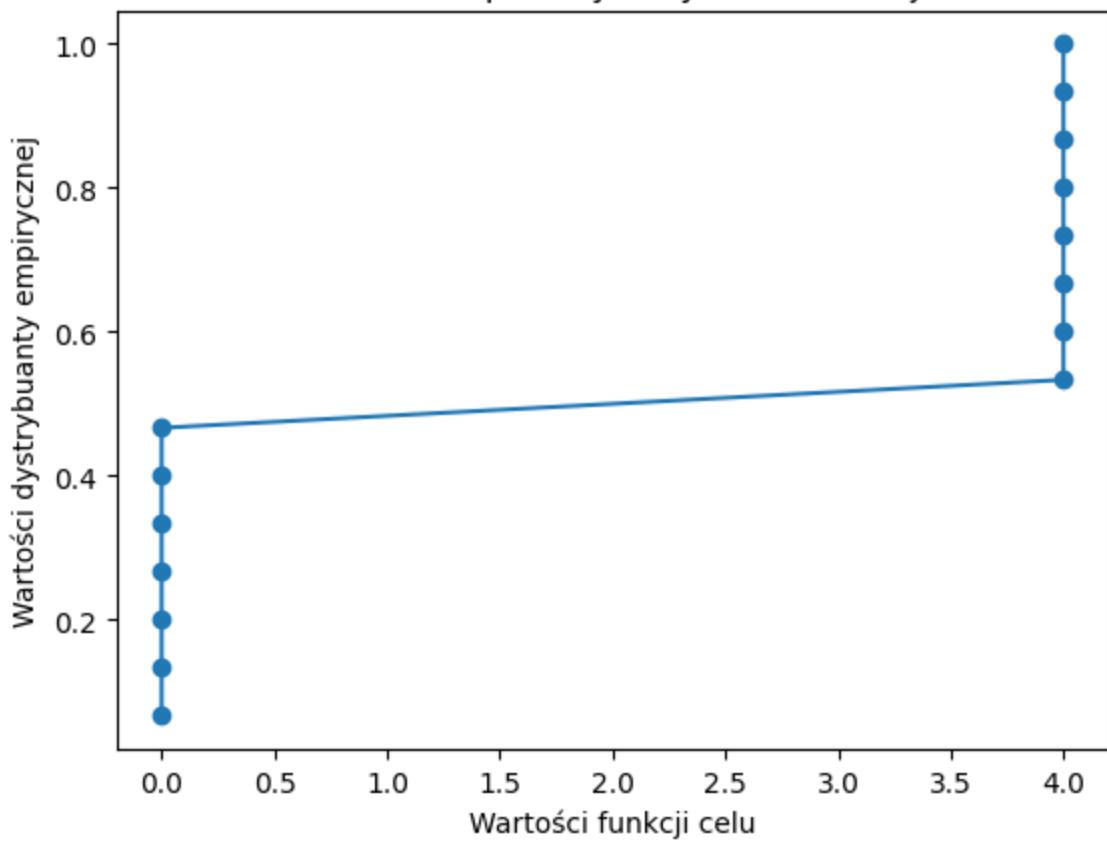
print()
print("Wyniki dla algorytmu genetycznego")
table = makeTable(foundBestScores, ["Znaleziono najlepsze rozwiązania", "Plansza", "Iter
print()
printTable(table)
print(f"Średnia znalezionych rozwiązań: {np.mean([value[0] for value in foundBestScores]}
print(f"Odchylenie standardowe znalezionych rozwiązań: {np.std([value[0] for value in fo
print(f"Średnia znalezionych rozwiązań dla prostej planszy: {np.mean([value[0] for value
print(f"Odchylenie standardowe znalezionych rozwiązań dla prostej planszy: {np.std([valu
print(f"Średnia znalezionych rozwiązań dla średniej planszy: {np.mean([value[0] for valu
print(f"Odchylenie standardowe znalezionych rozwiązań dla średniej: {np.std([value[0] fo
print(f"Średnia znalezionych rozwiązań dla trudnej planszy: {np.mean([value[0] for value
print(f"Odchylenie standardowe znalezionych rozwiązań dla trudnej planszy: {np.std([valu
print()
print("Wyniki dla algorytmu ACO")
table = makeTable(foundBestACOScores, ["Znaleziono najlepsze rozwiązania", "Plansza", "I
print()
printTable(table)
print(f"Średnia znalezionych rozwiązań: {np.mean([value[0] for value in foundBestACOScor
print(f"Odchylenie standardowe znalezionych rozwiązań: {np.std([value[0] for value in fo
print(f"Średnia znalezionych rozwiązań dla prostej planszy: {np.mean([value[0] for value
print(f"Odchylenie standardowe znalezionych rozwiązań dla prostej planszy: {np.std([valu
print(f"Średnia znalezionych rozwiązań dla średniej planszy: {np.mean([value[0] for valu
print(f"Odchylenie standardowe znalezionych rozwiązań dla średniej: {np.std([value[0] fo
print(f"Średnia znalezionych rozwiązań dla trudnej planszy: {np.mean([value[0] for value
print(f"Odchylenie standardowe znalezionych rozwiązań dla trudnej planszy: {np.std([valu

```

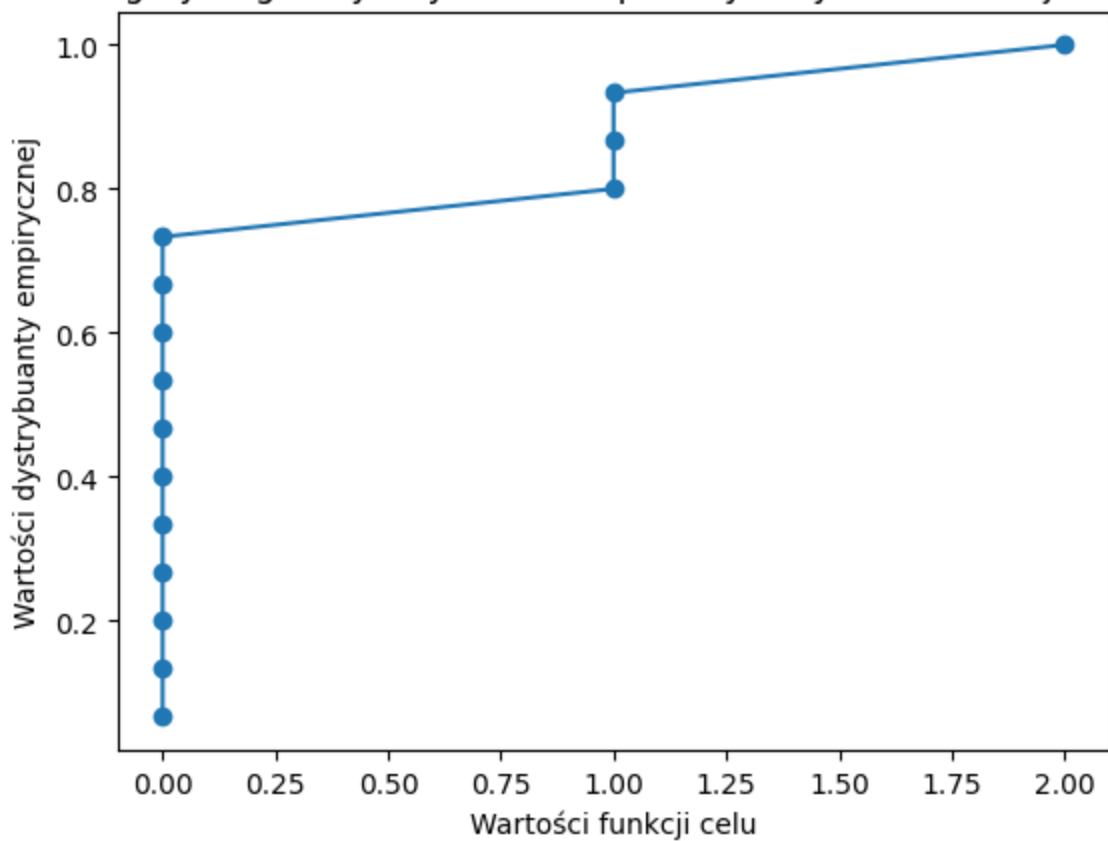
Algorytm genetyczny ECDF dla planszy easy liczba iteracji 75



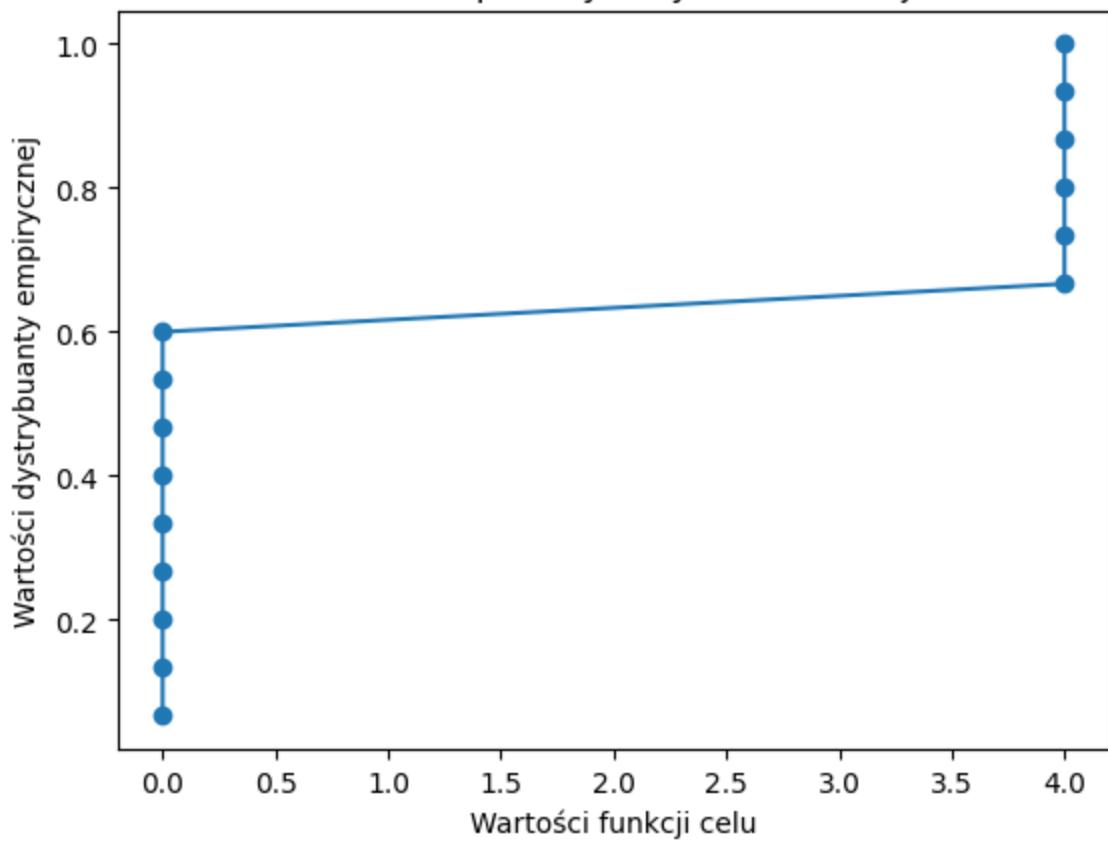
ACO ECDF dla planszy easy liczba iteracji 75



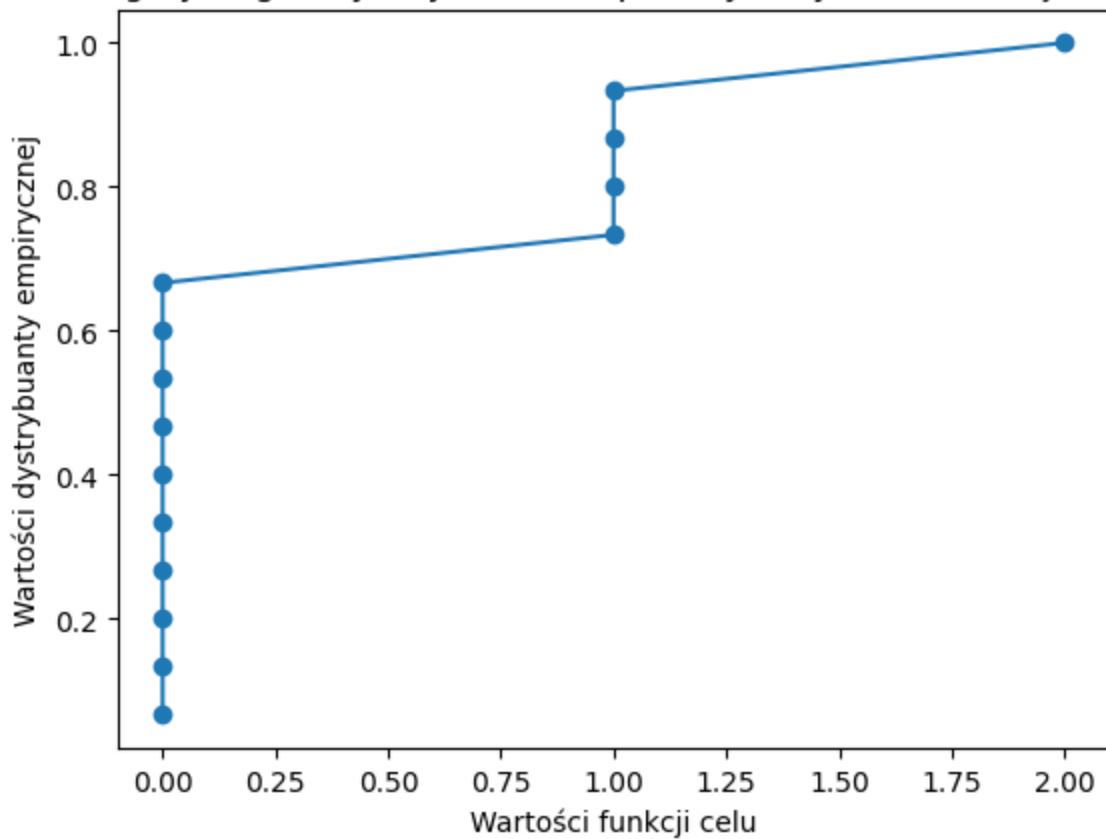
Algorytm genetyczny ECDF dla planszy easy liczba iteracji 100



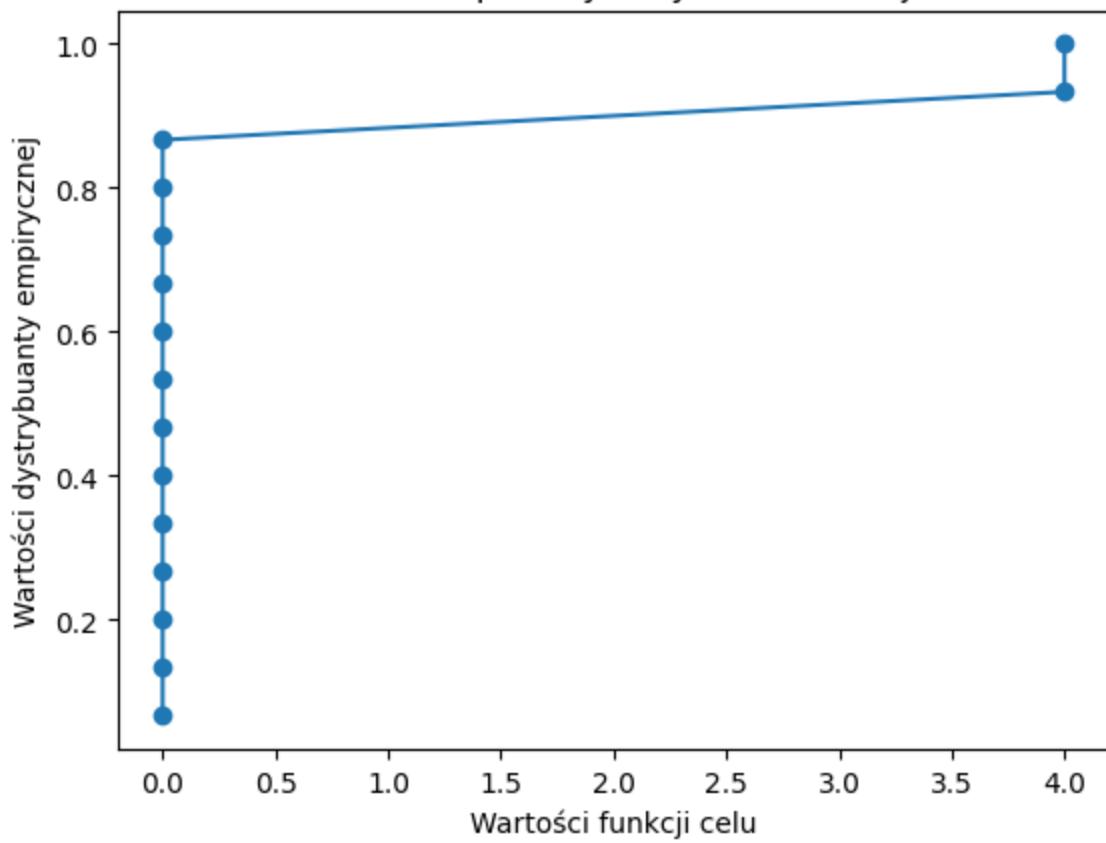
ACO ECDF dla planszy easy liczba iteracji 100



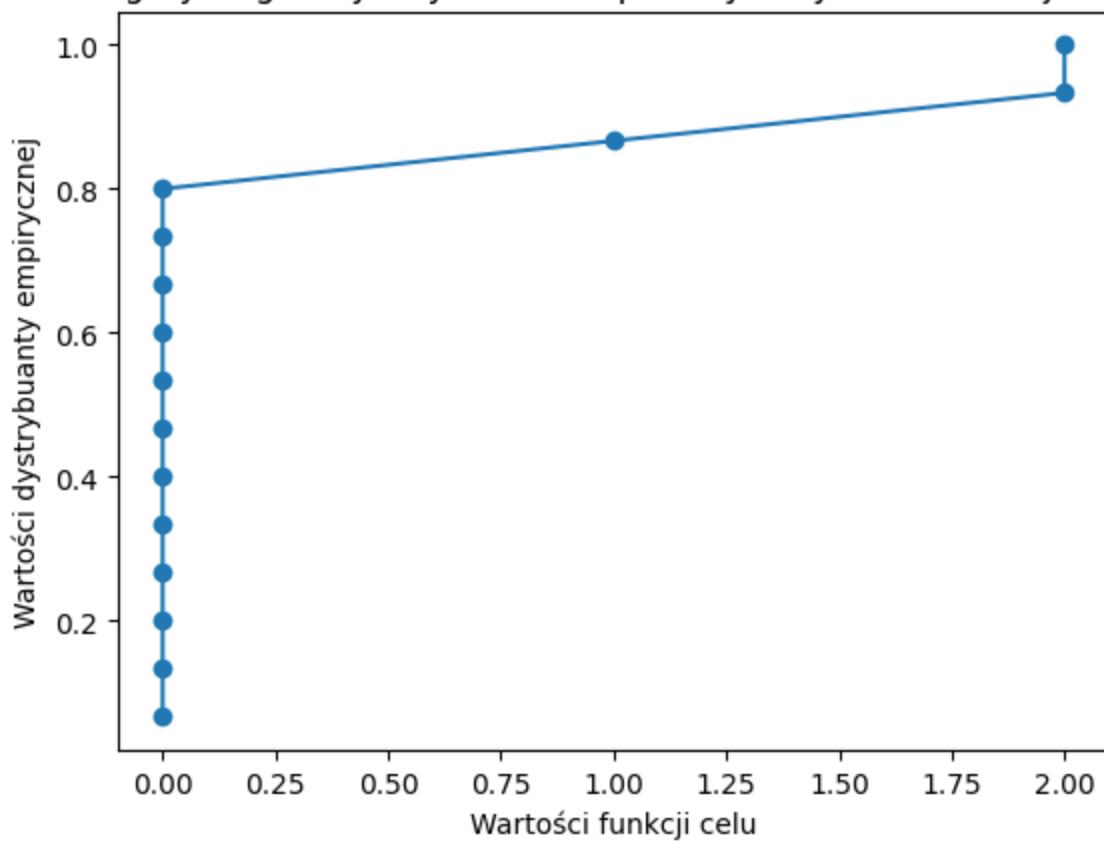
Algorytm genetyczny ECDF dla planszy easy liczba iteracji 150



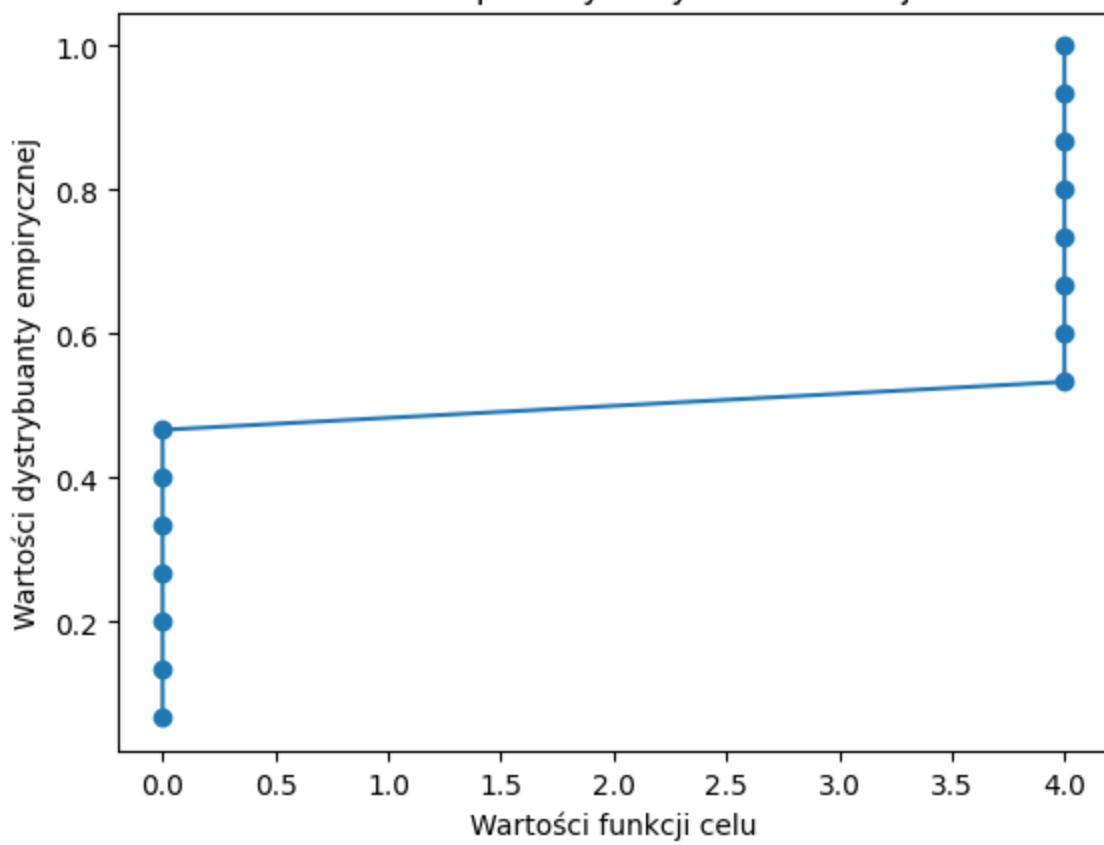
ACO ECDF dla planszy easy liczba iteracji 150



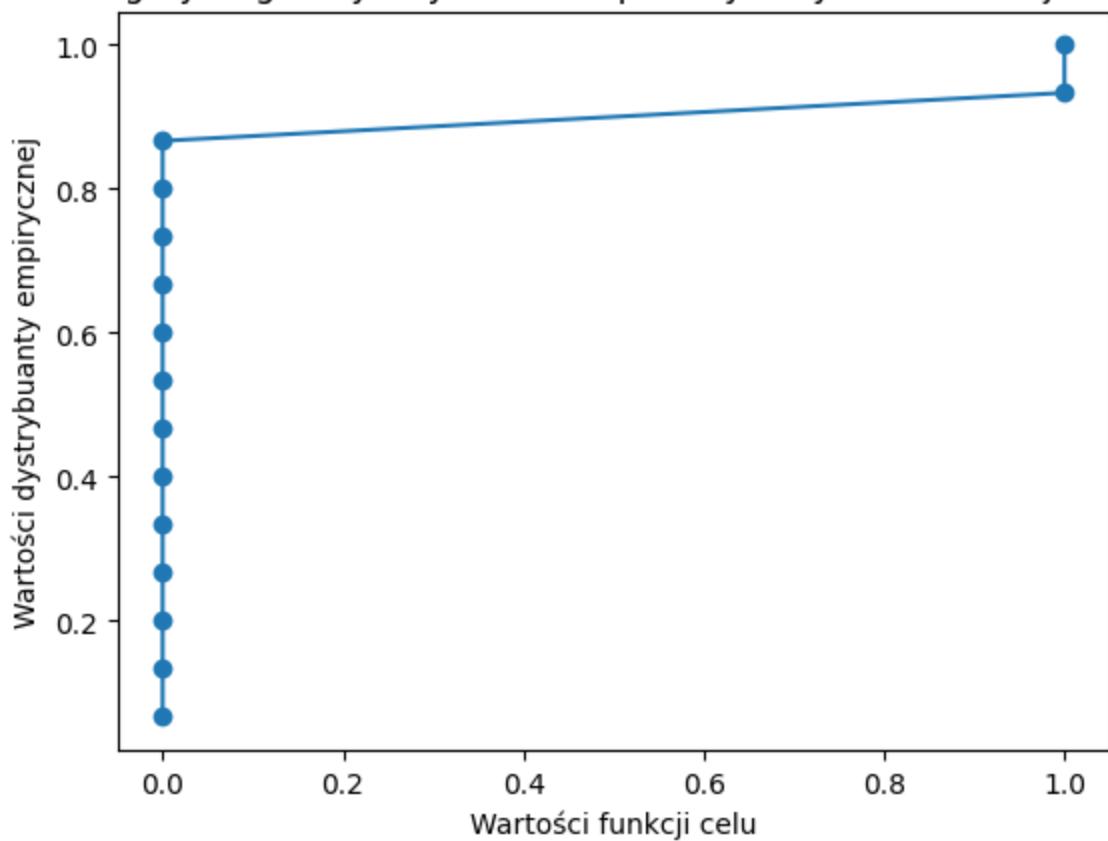
Algorytm genetyczny ECDF dla planszy easy liczba iteracji 200



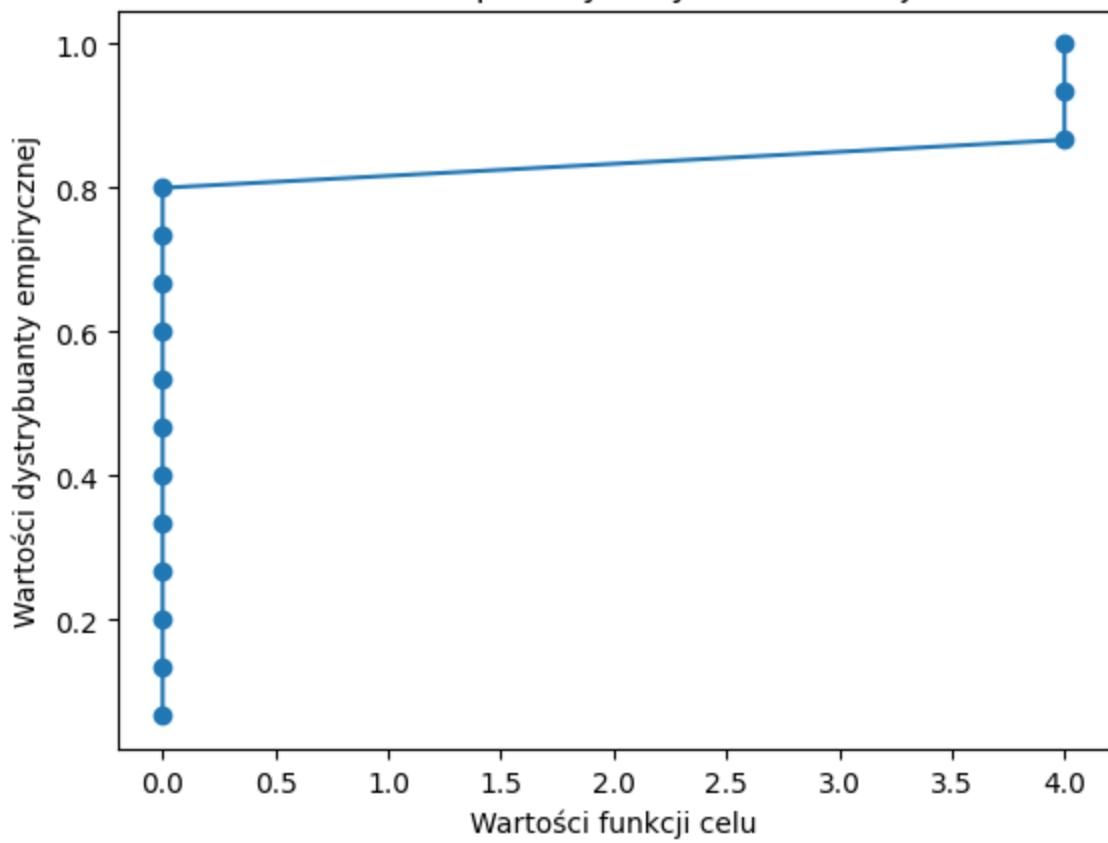
ACO ECDF dla planszy easy liczba iteracji 200



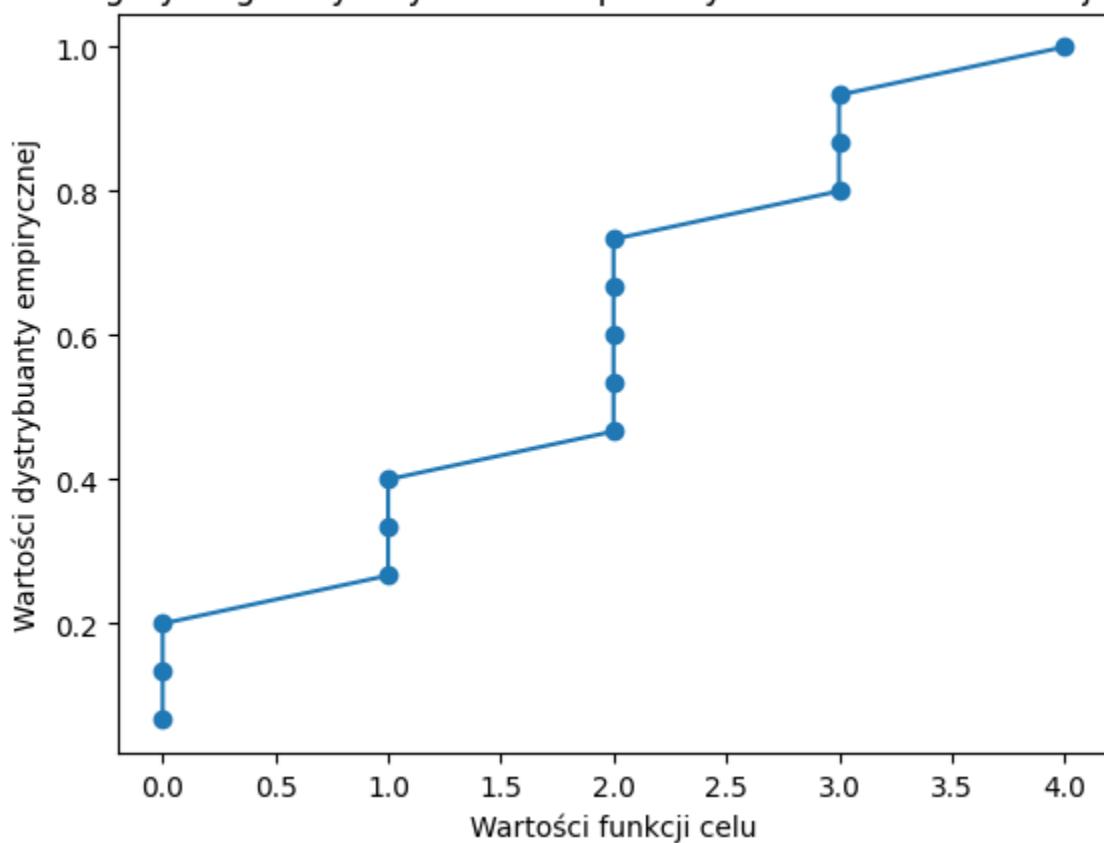
Algorytm genetyczny ECDF dla planszy easy liczba iteracji 300



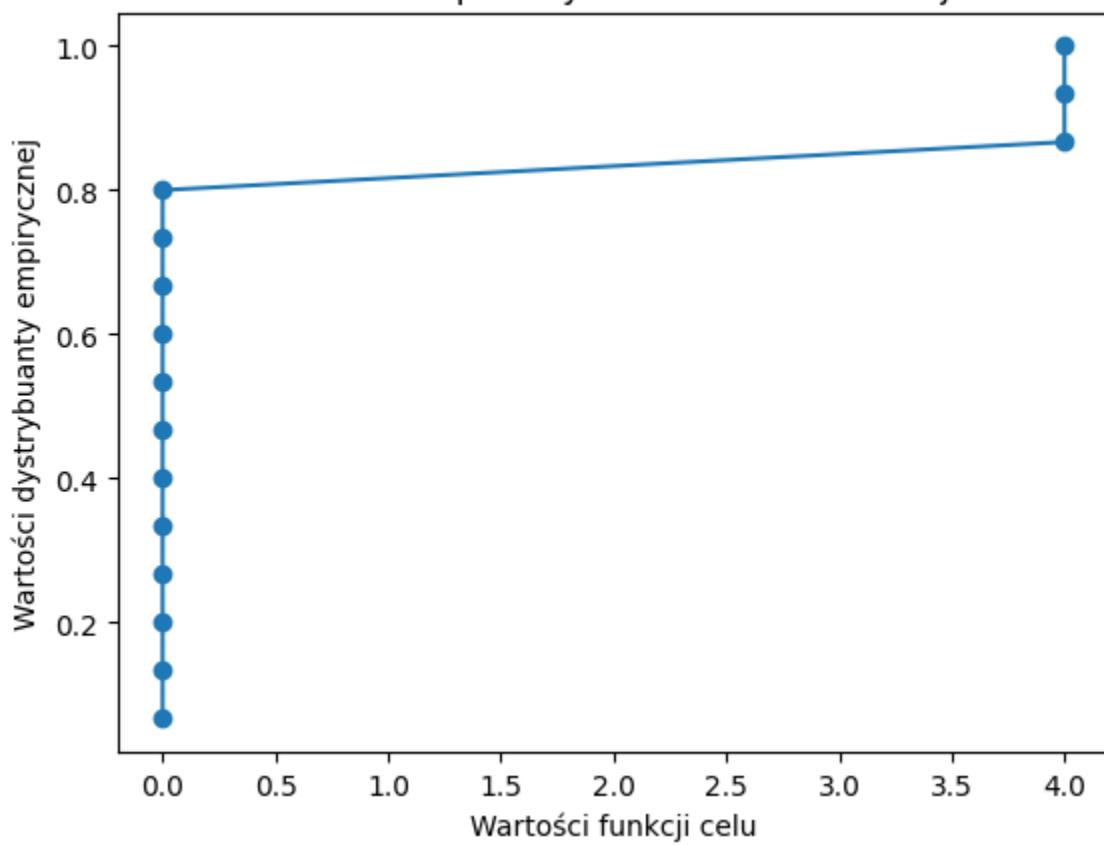
ACO ECDF dla planszy easy liczba iteracji 300



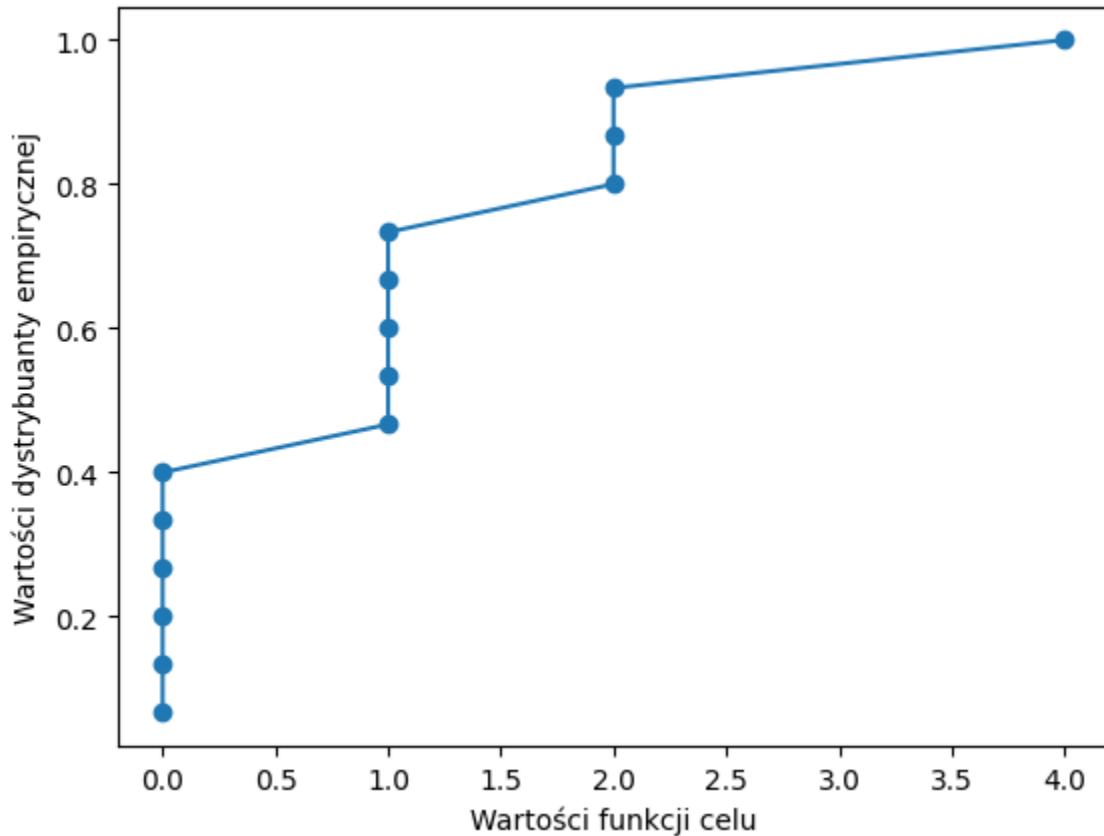
Algorytm genetyczny ECDF dla planszy medium liczba iteracji 75



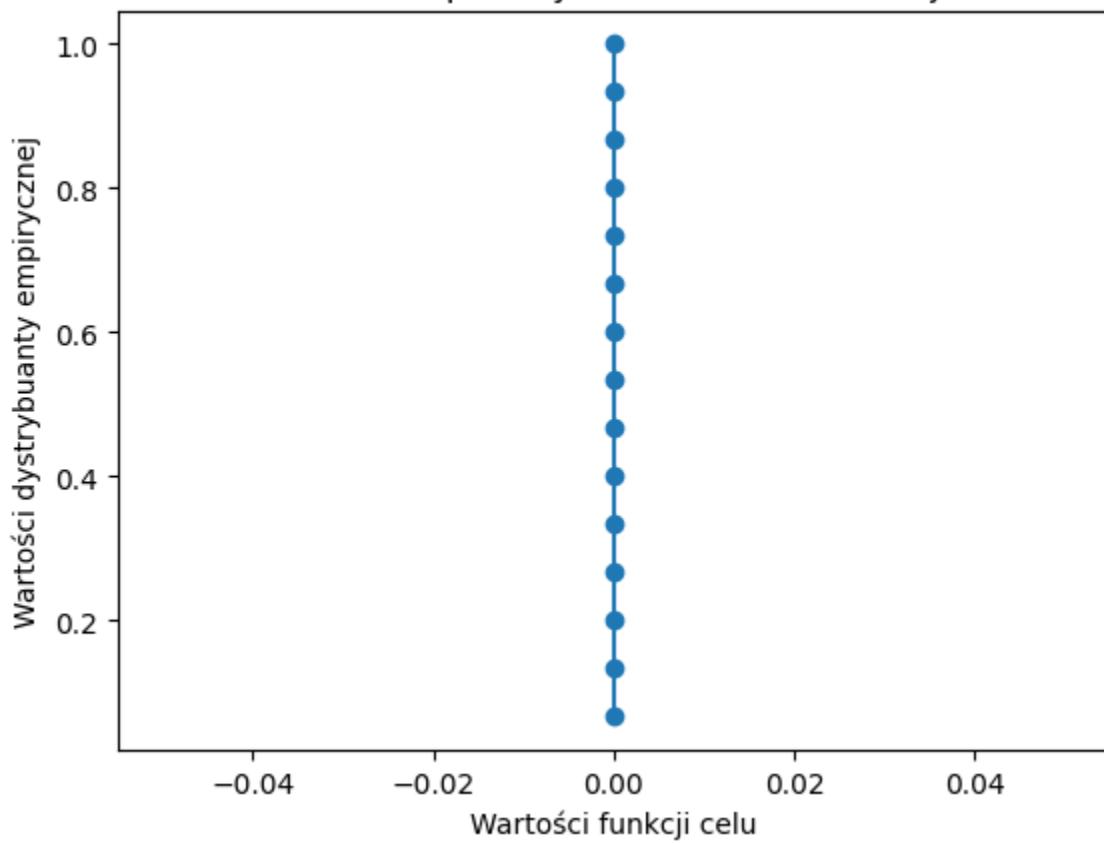
ACO ECDF dla planszy medium liczba iteracji 75



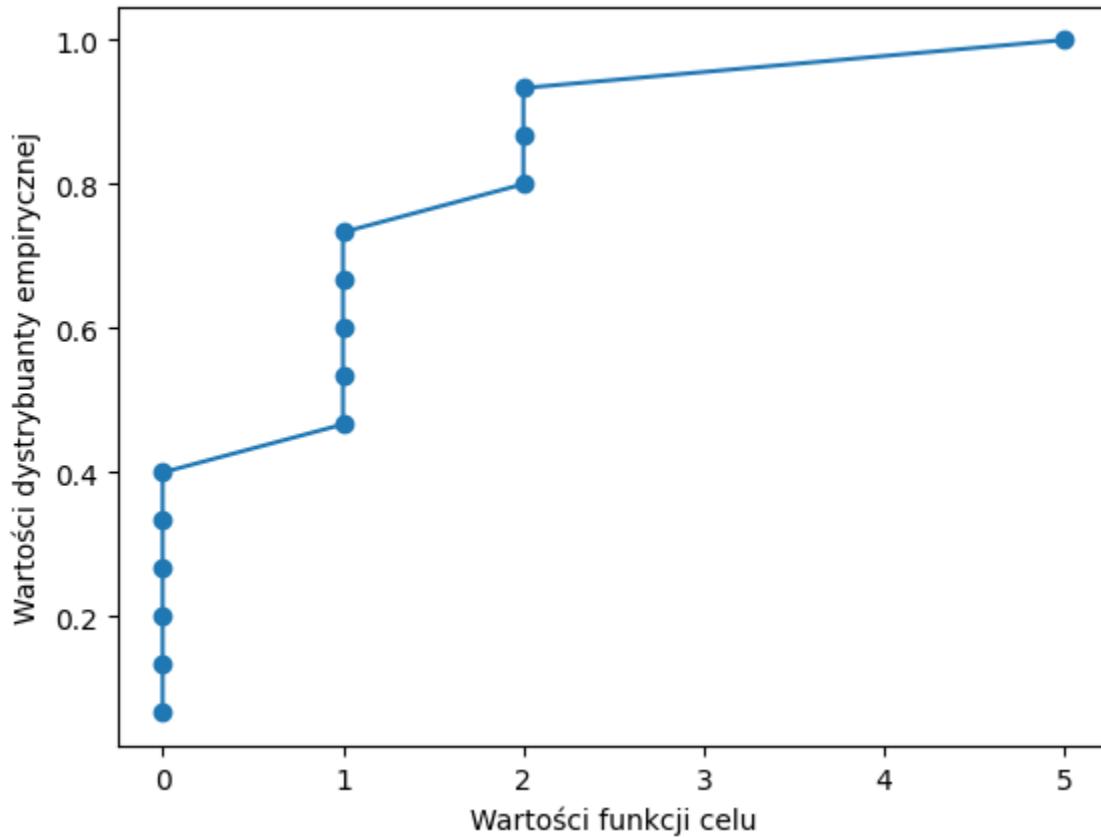
Algorytm genetyczny ECDF dla planszy medium liczba iteracji 100



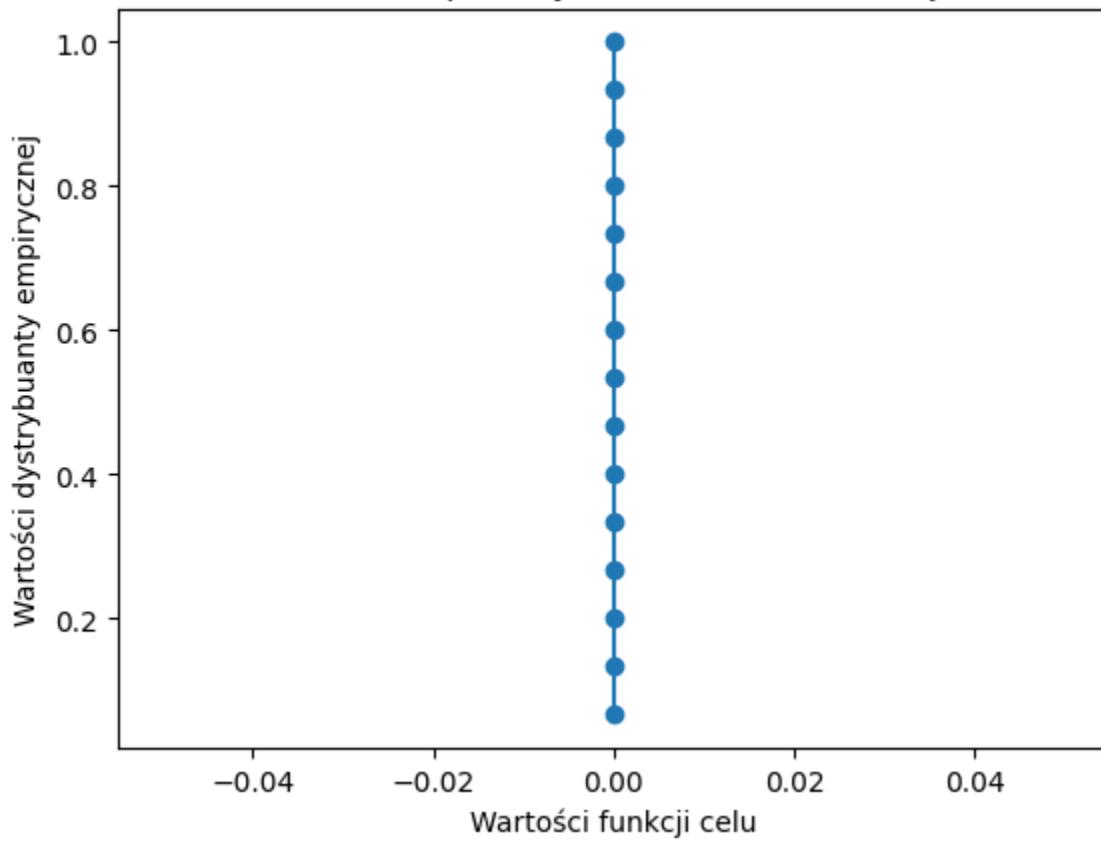
ACO ECDF dla planszy medium liczba iteracji 100



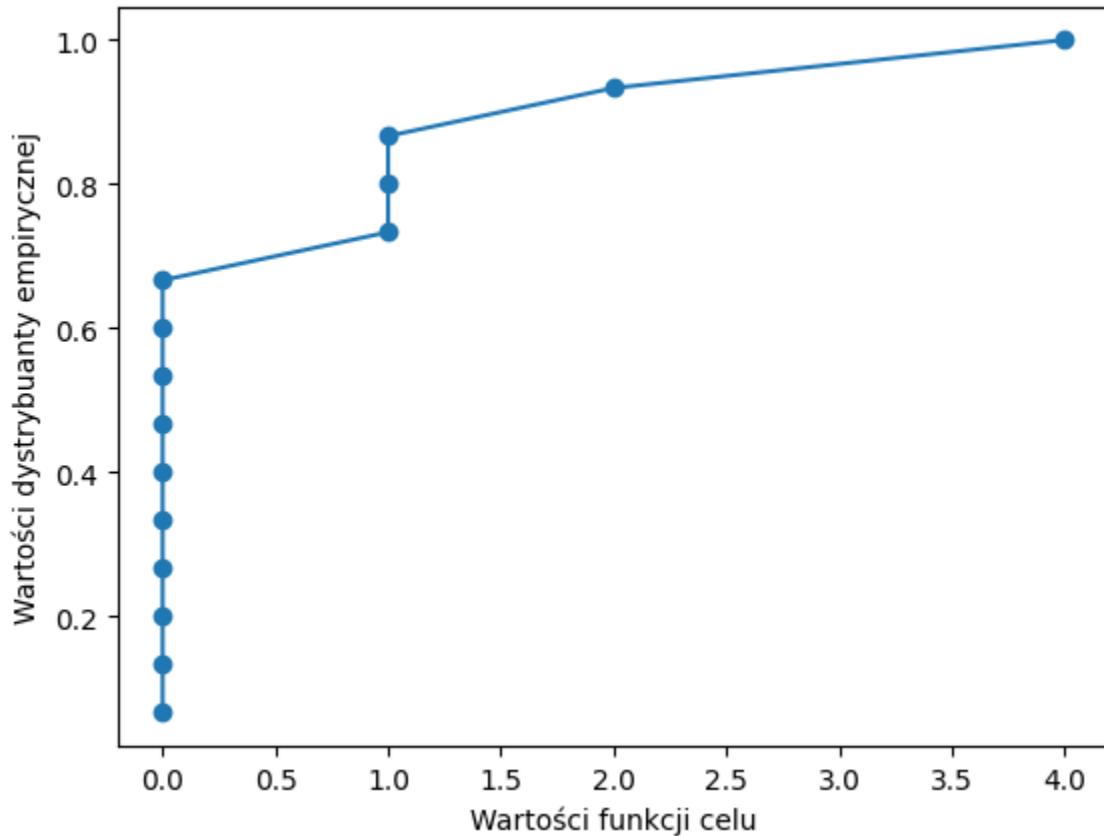
Algorytm genetyczny ECDF dla planszy medium liczba iteracji 150



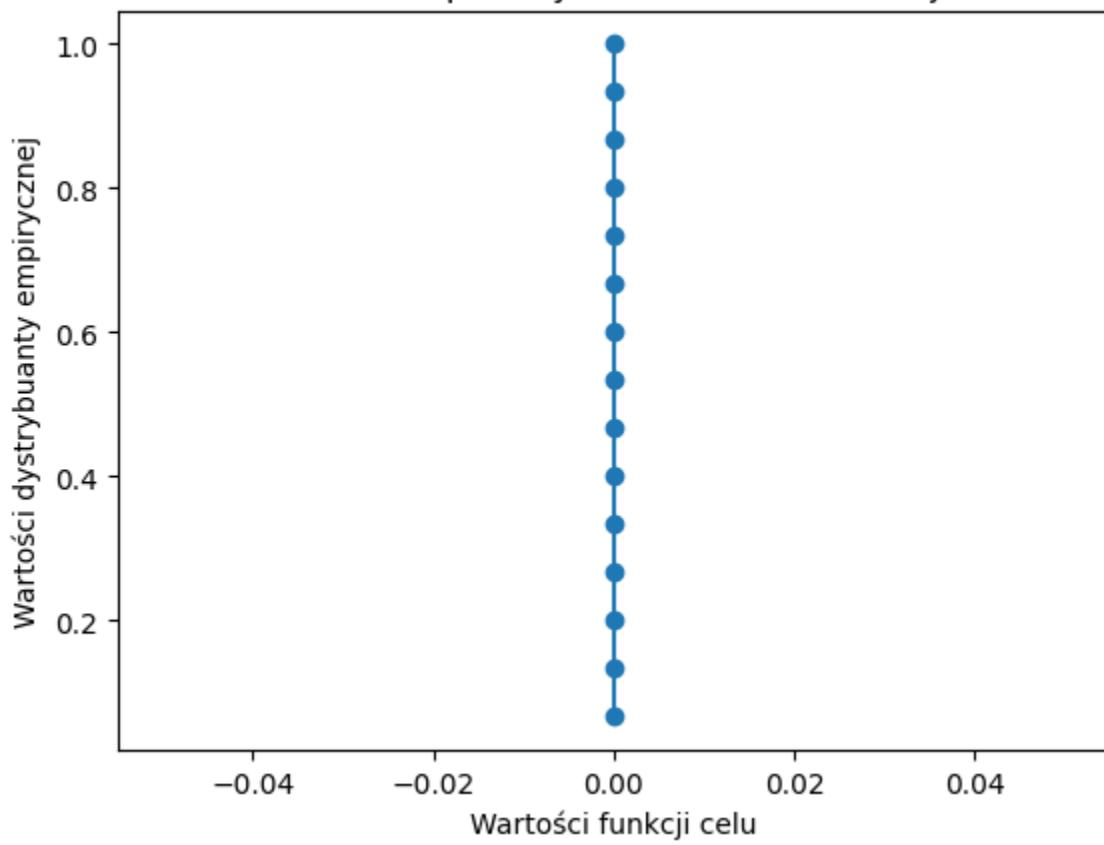
ACO ECDF dla planszy medium liczba iteracji 150



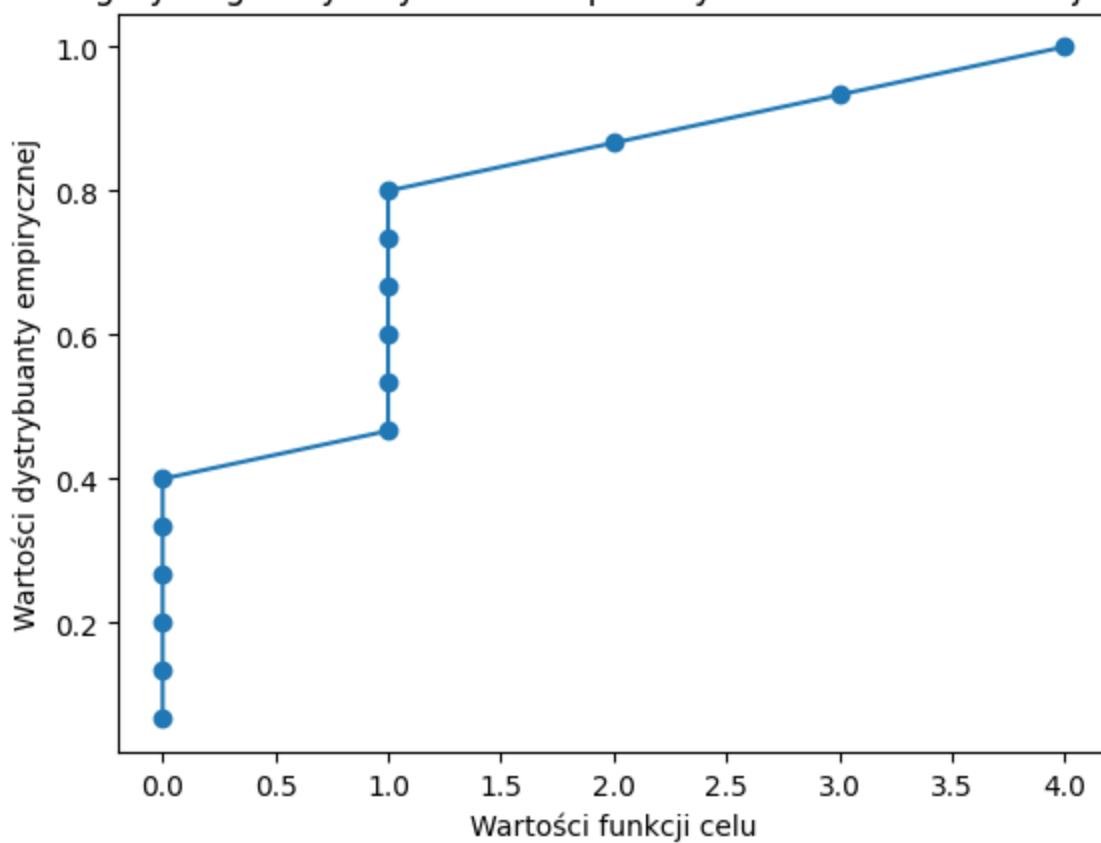
Algorytm genetyczny ECDF dla planszy medium liczba iteracji 200



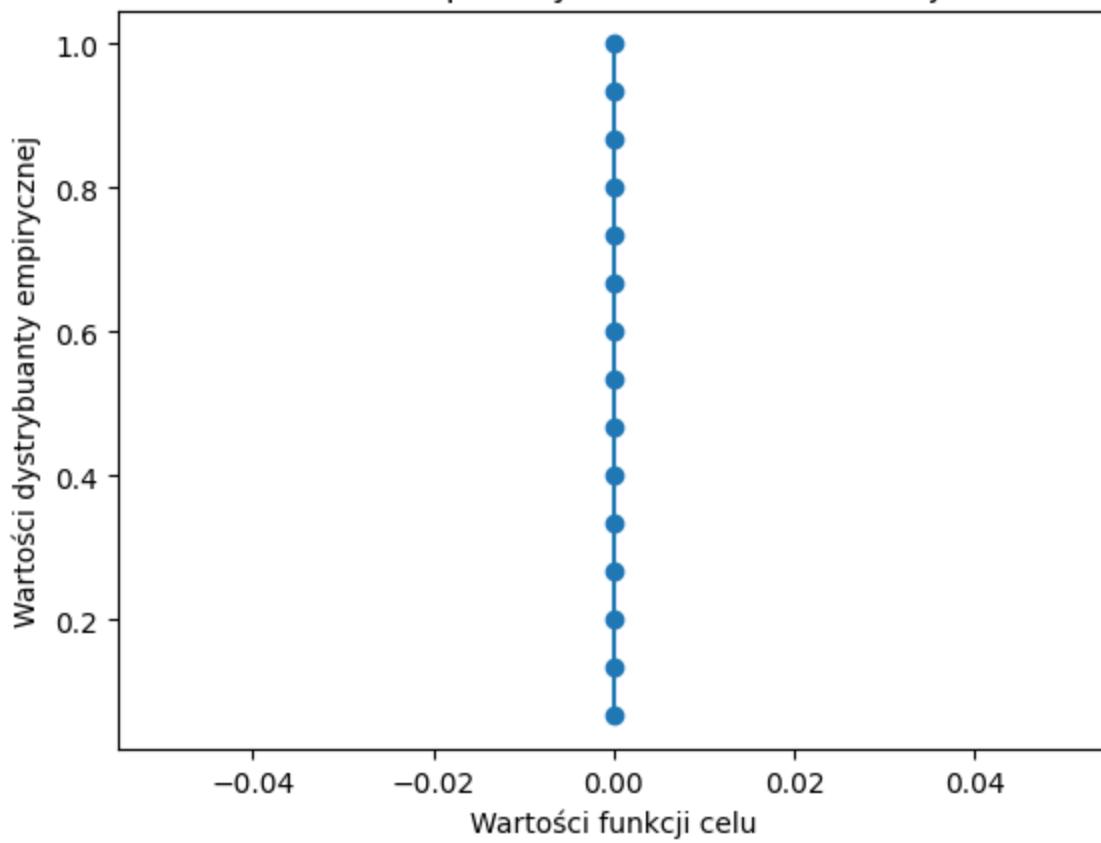
ACO ECDF dla planszy medium liczba iteracji 200



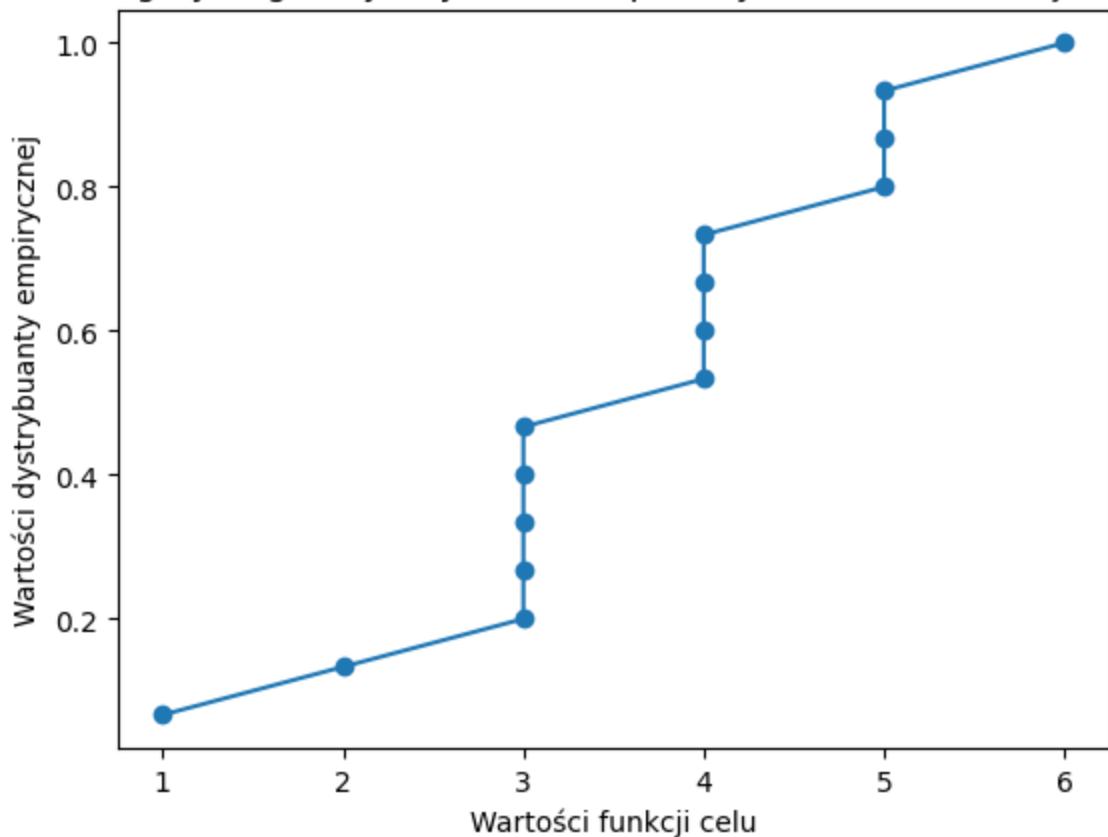
Algorytm genetyczny ECDF dla planszy medium liczba iteracji 300



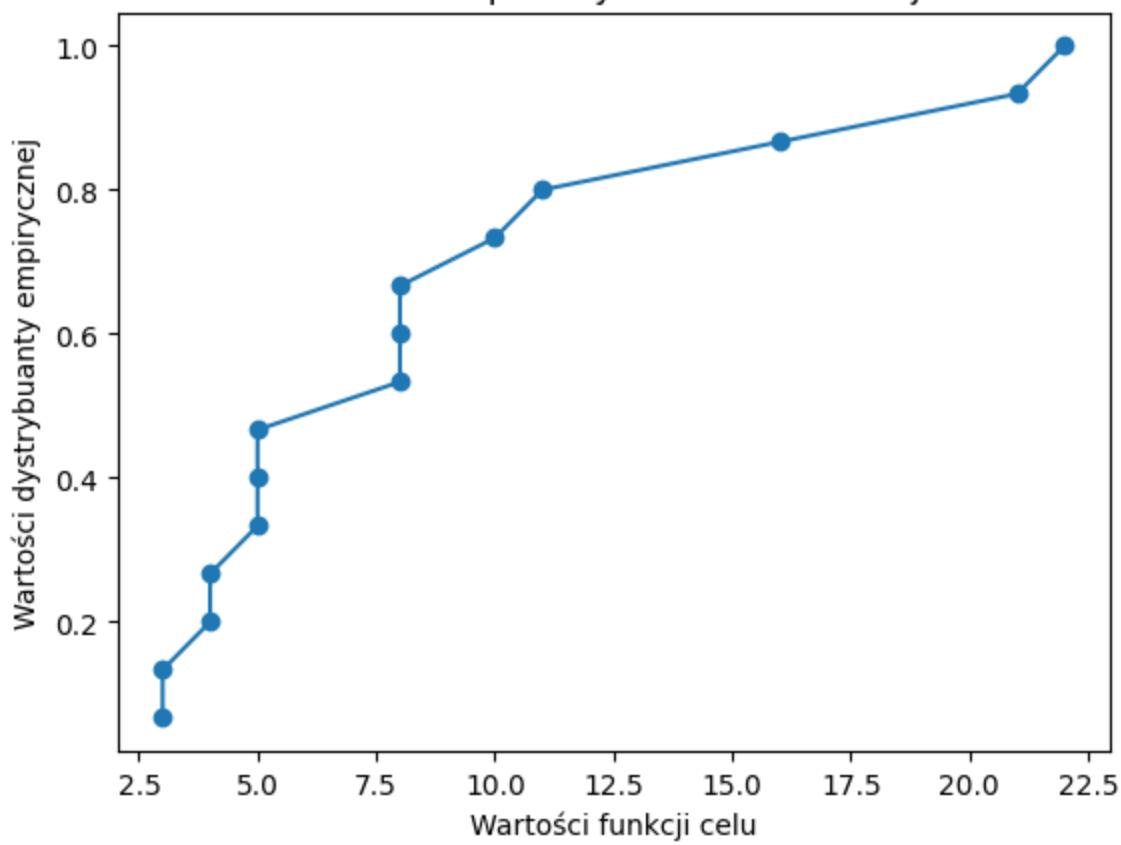
ACO ECDF dla planszy medium liczba iteracji 300



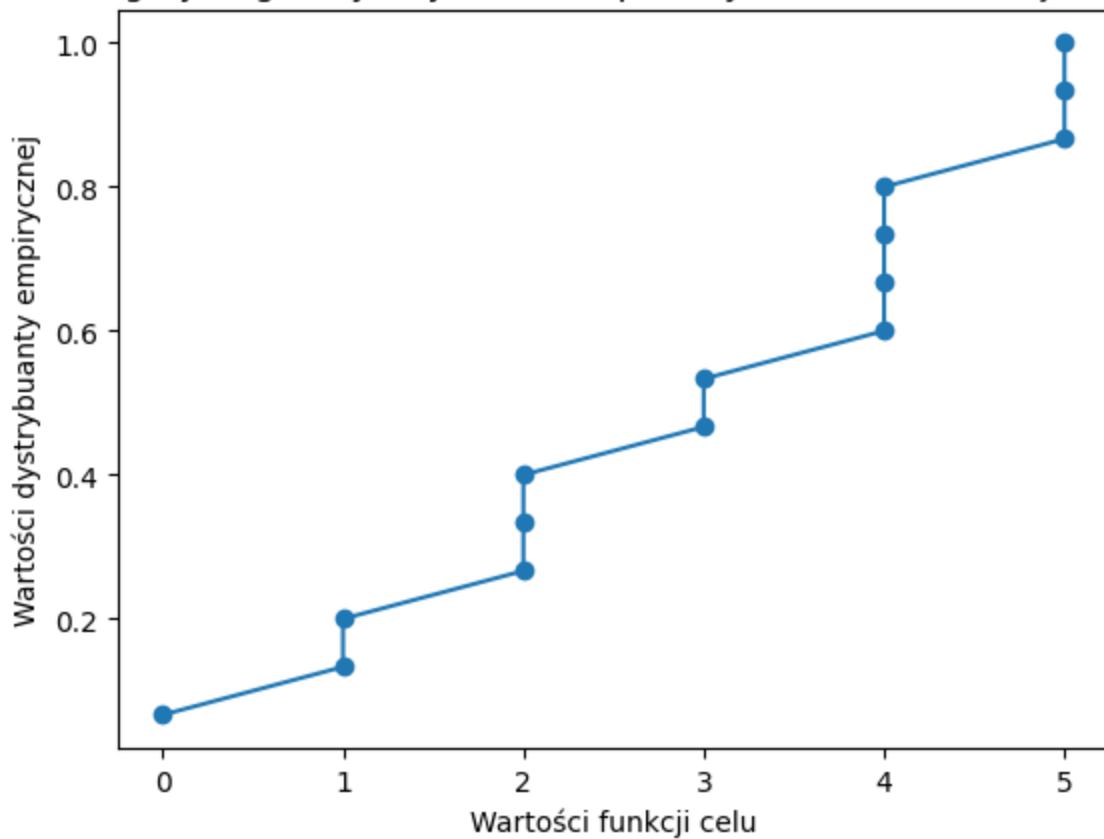
Algorytm genetyczny ECDF dla planszy hard liczba iteracji 75



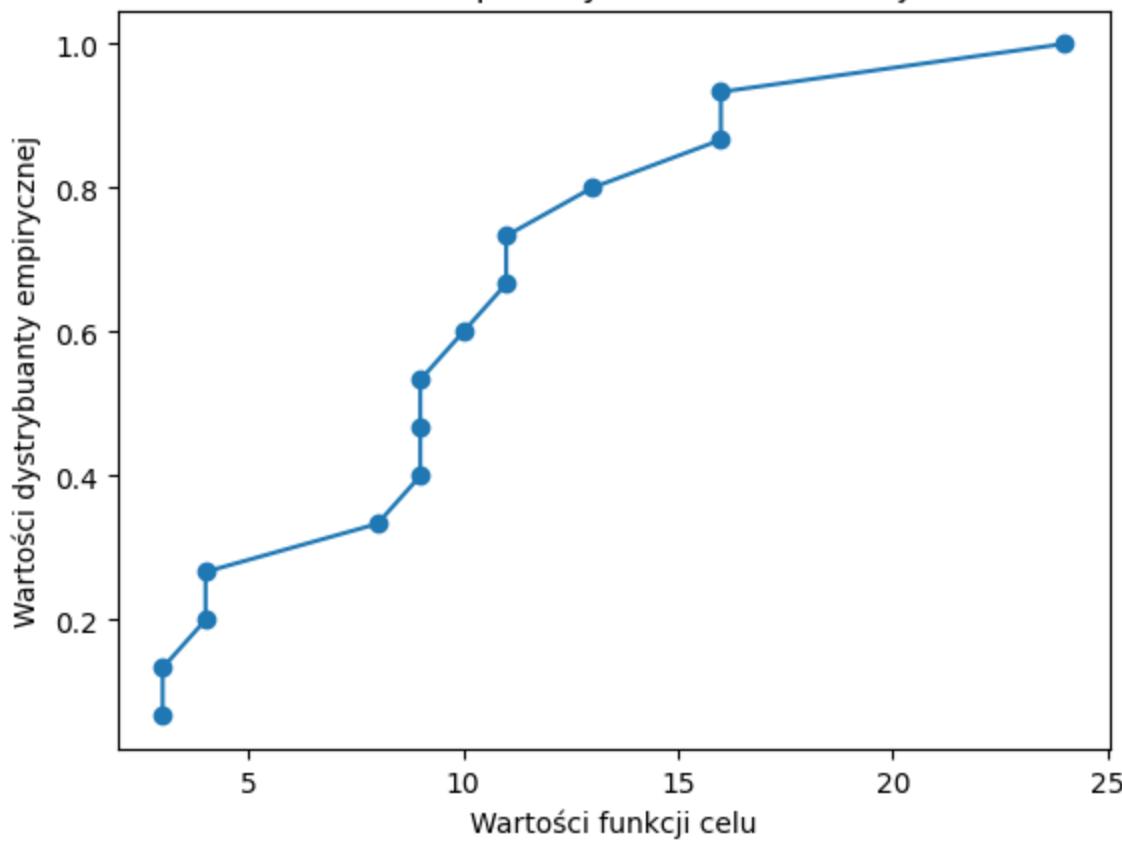
ACO ECDF dla planszy hard liczba iteracji 75



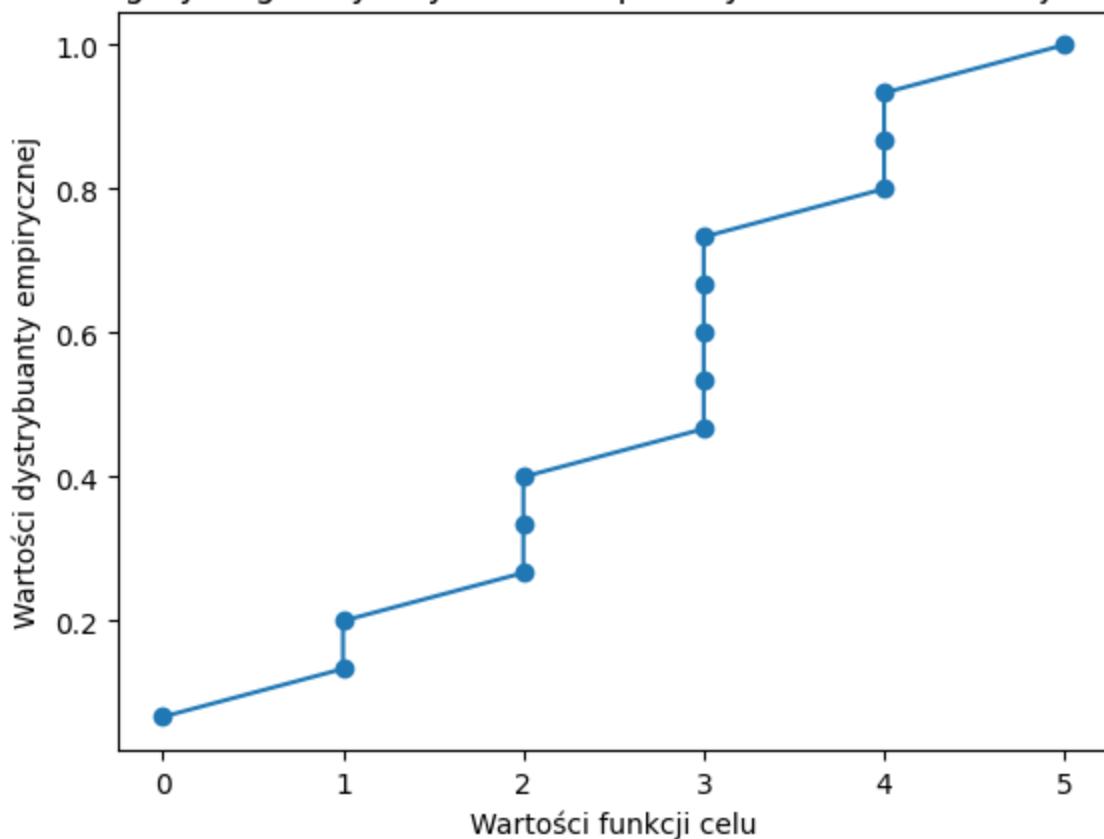
Algorytm genetyczny ECDF dla planszy hard liczba iteracji 100



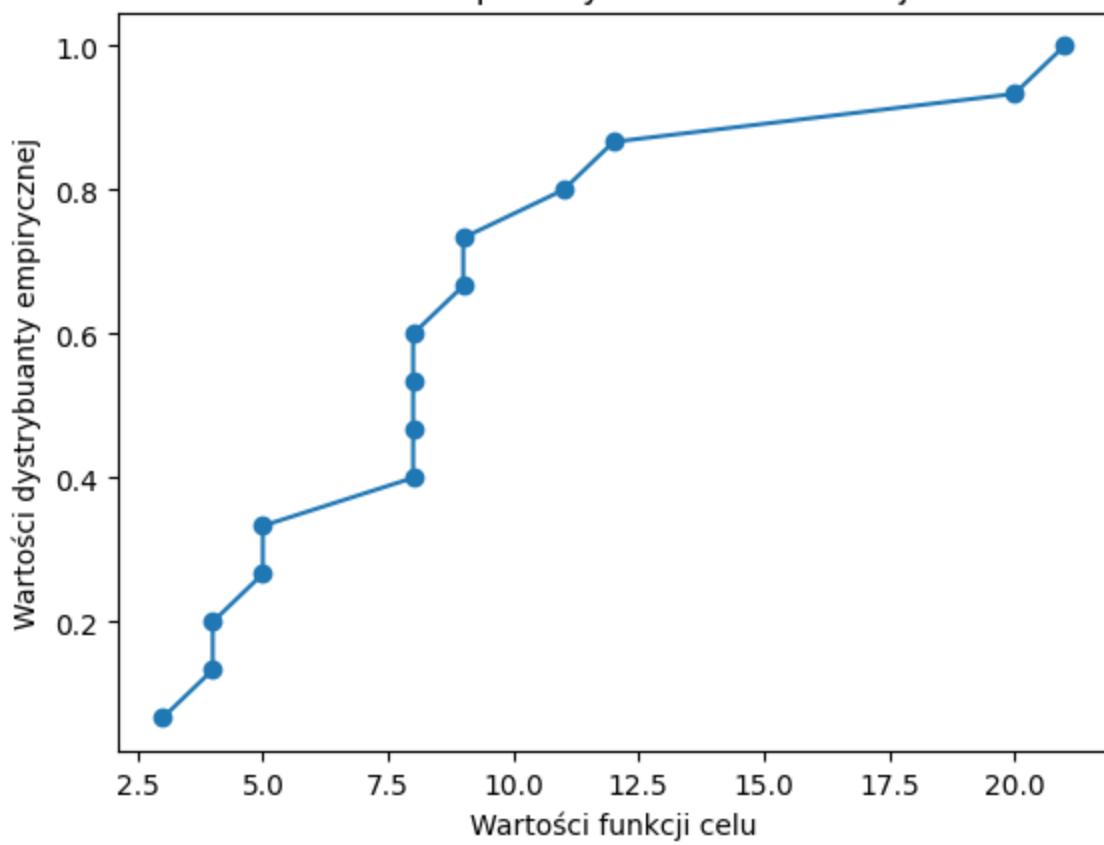
ACO ECDF dla planszy hard liczba iteracji 100



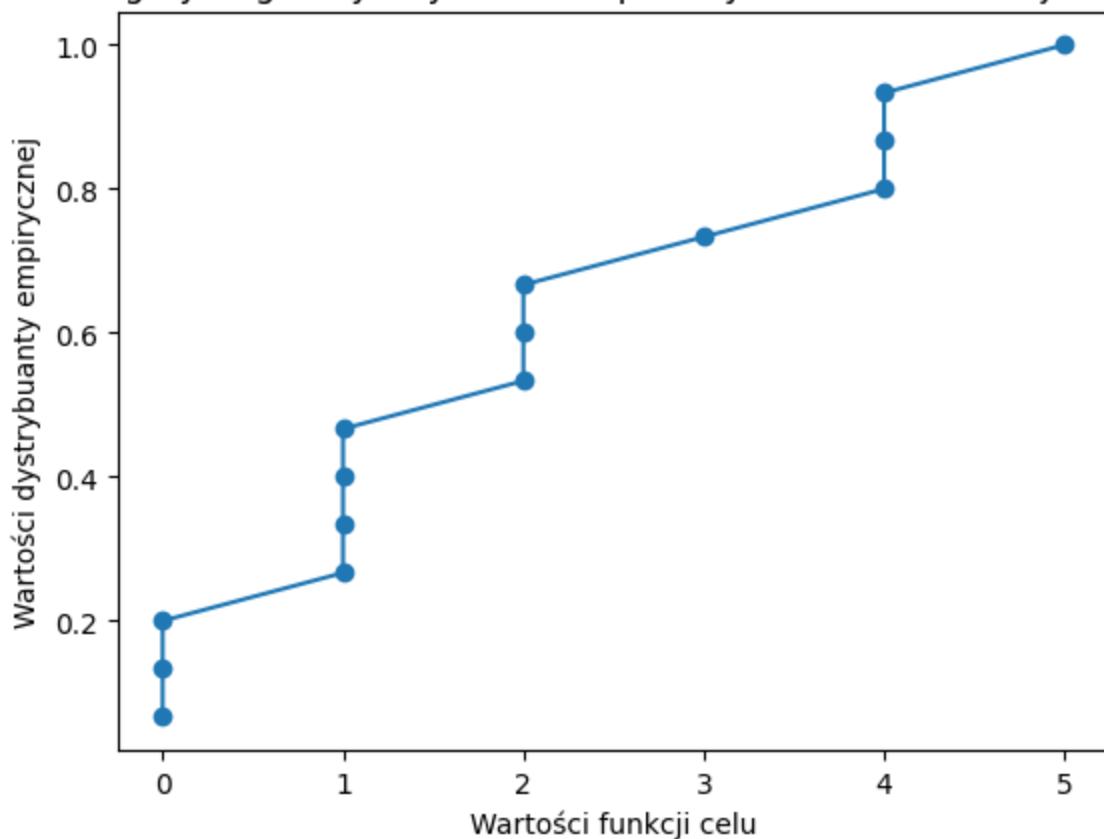
Algorytm genetyczny ECDF dla planszy hard liczba iteracji 150



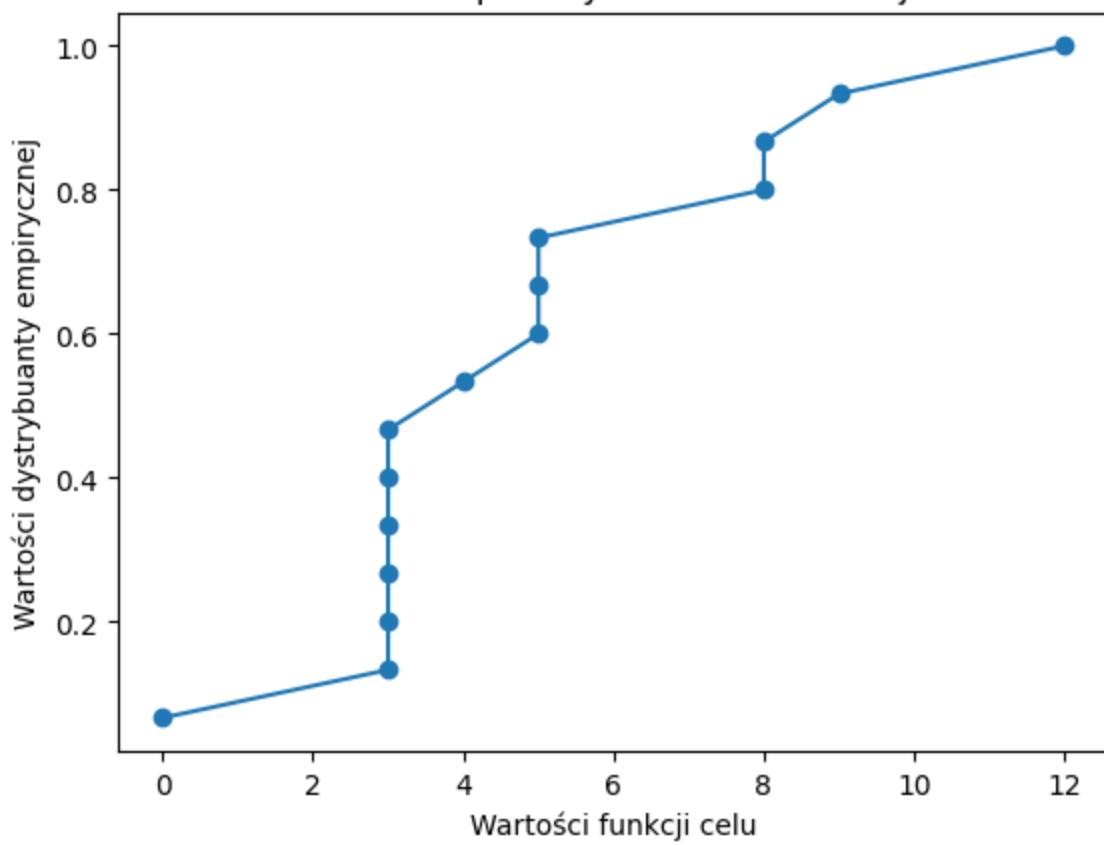
ACO ECDF dla planszy hard liczba iteracji 150



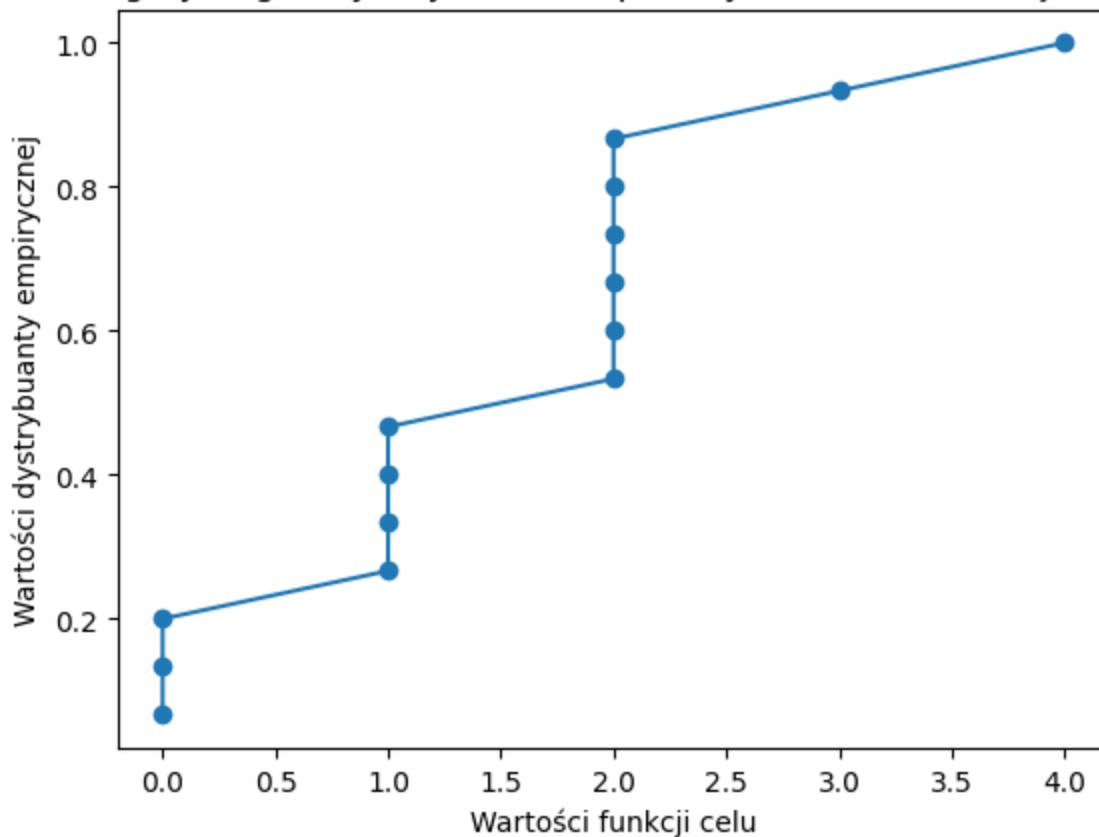
Algorytm genetyczny ECDF dla planszy hard liczba iteracji 200



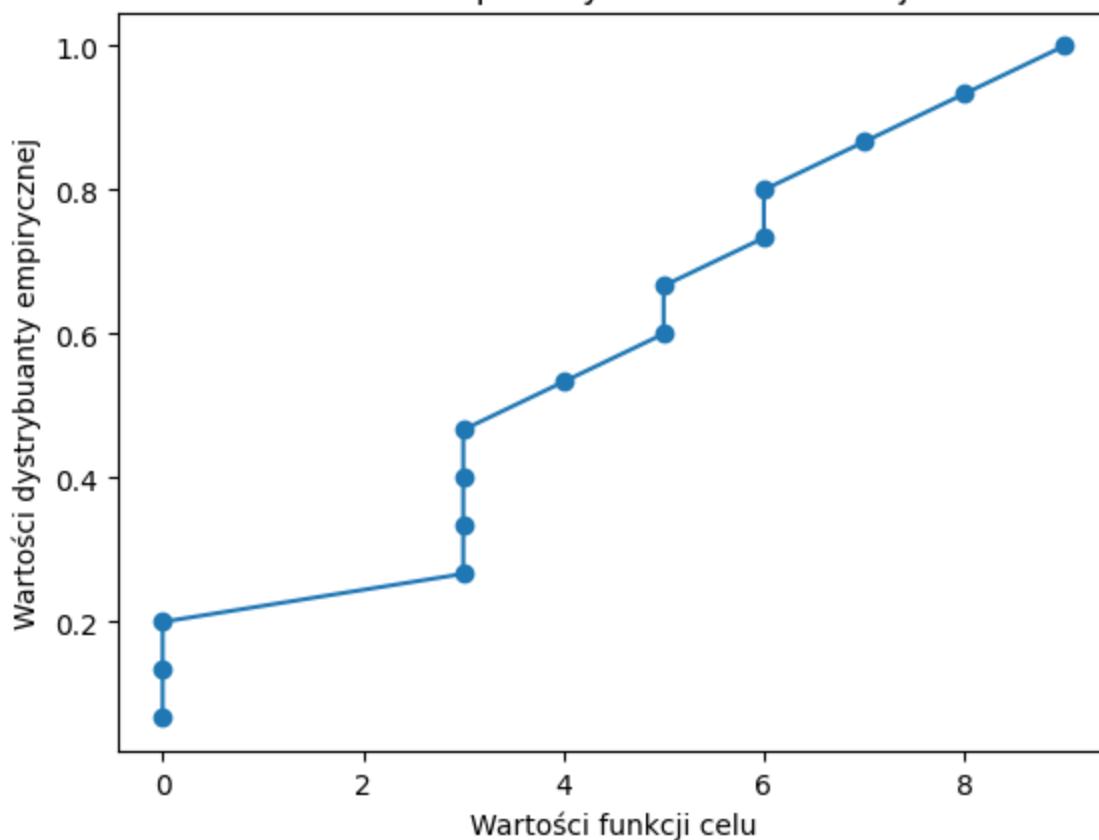
ACO ECDF dla planszy hard liczba iteracji 200



Algorytm genetyczny ECDF dla planszy hard liczba iteracji 300



ACO ECDF dla planszy hard liczba iteracji 300



Wyniki dla algorytmu genetycznego

Znaleziono najlepsze rozwiązania	Plansza	Iteracje
12	easy	75
11	easy	100

10	easy	150
12	easy	200
13	easy	300
3	medium	75
6	medium	100
6	medium	150
10	medium	200
6	medium	300
0	hard	75
1	hard	100
1	hard	150
3	hard	200
3	hard	300

Średnia znalezionych rozwiązań: 6.466666666666667

Odchylenie standardowe znalezionych rozwiązań: 4.3797513881751575

Średnia znalezionych rozwiązań dla prostej planszy: 11.6

Odchylenie standardowe znalezionych rozwiązań dla prostej planszy: 1.019803902718557

Średnia znalezionych rozwiązań dla średniej planszy: 6.2

Odchylenie standardowe znalezionych rozwiązań dla średniej: 2.227105745132009

Średnia znalezionych rozwiązań dla trudnej planszy: 1.6

Odchylenie standardowe znalezionych rozwiązań dla trudnej planszy: 1.2000000000000002

Wyniki dla algorytmu ACO

Znaleziono najlepsze rozwiązania	Plansza	Iteracje
7	easy	75
9	easy	100
13	easy	150
7	easy	200
12	easy	300
12	medium	75
15	medium	100
15	medium	150
15	medium	200
15	medium	300
0	hard	75
0	hard	100
0	hard	150

1	hard	200
3	hard	300

Średnia znalezionych rozwiązań: 8.266666666666667
 Odchylenie standardowe znalezionych rozwiązań: 5.8931221681625505
 Średnia znalezionych rozwiązań dla prostej planszy: 9.6
 Odchylenie standardowe znalezionych rozwiązań dla prostej planszy: 2.4979991993593593
 Średnia znalezionych rozwiązań dla średniej planszy: 14.4
 Odchylenie standardowe znalezionych rozwiązań dla średniej: 1.2
 Średnia znalezionych rozwiązań dla trudnej planszy: 0.8
 Odchylenie standardowe znalezionych rozwiązań dla trudnej planszy: 1.1661903789690602

Testy szybkości przy ustalonej maksymalnej liczbie iteracji

Podobnie jak wyżej, utworzymy wykresy ECDF tylko tym razem policzymy dystrybuantę potrzebnych iteracji do otrzymania rozwiązania/zakończenia algorytmów dla kolejnych plansz, gdzie maksymalna liczba iteracji = 300. To samo znajduje się w pliku minim_iterations_ecdf.py

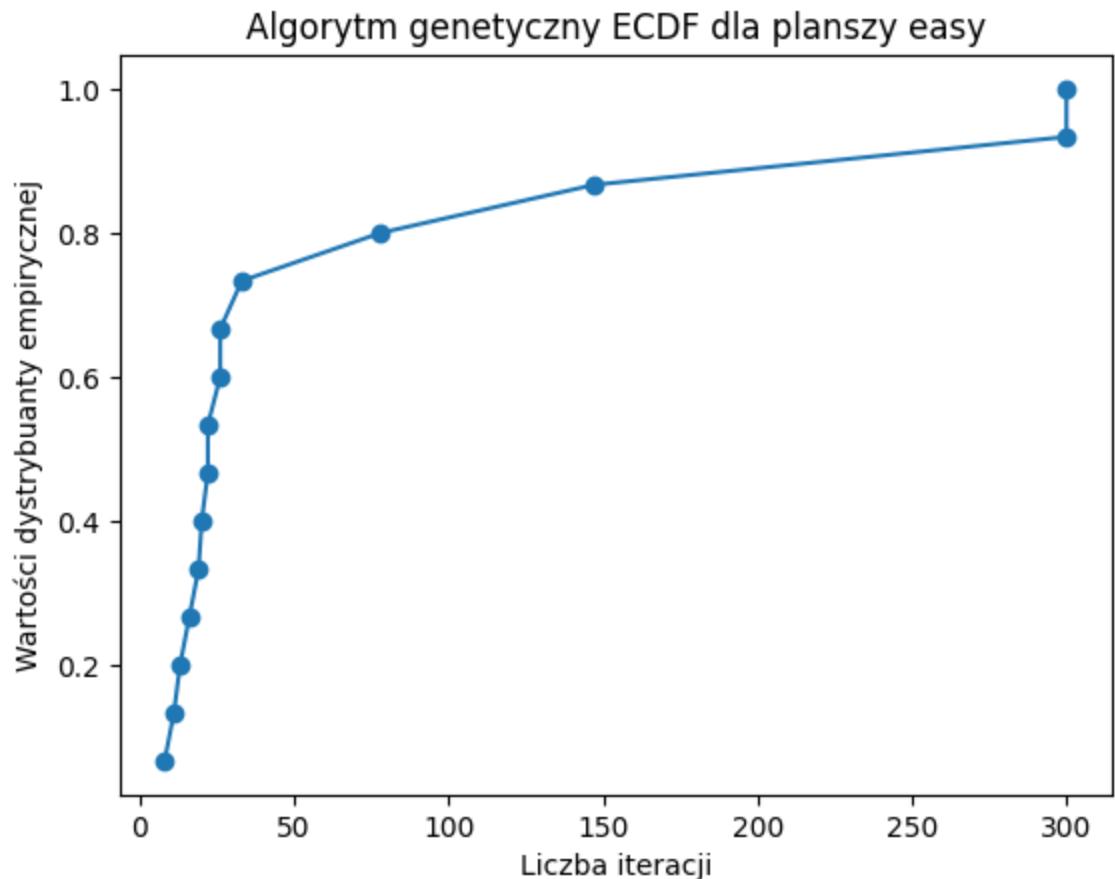
```
In [11]: results = []
resultsACO = []
for boardStr, boardType in zip(boardStrs, ["easy", "medium", "hard"]):
    boardScores = []
    bestScores = []
    boardACOScores = []
    bestACOScores = []
    for experimentNumber in range(15):
        pop0 = makeBoardsFromFile(boardStr, pop0Size=bestArgs[boardType][0], random_stat=argsList = [pop0, 300]
        argsList.extend(bestArgs[boardType][2:]))
        bestScore, bestSollution, scores, evaluations = genethicAlgorithm(*argsList)
        boardScores.append(len(scores))
        bestScores.append(bestScore)
        solver = Solver(*bestArgsACO[boardType], max_iterations=300)
        board = create_board_from_str(boardStr)
        bestScore, bestSollution, scores, evaluations, iterations = solver.solve(board)
        boardACOScores.append(iterations)
        bestACOScores.append(bestScore)
    ecdf = ECDF(boardScores)
    plt.plot(ecdf.x, ecdy.y, marker='o', linestyle='--')
    plt.xlabel("Liczba iteracji")
    plt.ylabel('Wartości dystrybuanty empirycznej')
    plt.title(f"Algorytm genetyczny ECDF dla planszy {boardType}")
    plt.show()
    for numberofIterations, best in zip(boardScores, bestScores):
        results.append([numberofIterations, boardType, best])
    ecdf = ECDF(boardACOScores)
    plt.plot(ecdf.x, ecdy.y, marker='o', linestyle='--')
    plt.xlabel("Liczba iteracji")
    plt.ylabel('Wartości dystrybuanty empirycznej')
    plt.title(f"ACO ECDF dla planszy {boardType}")
    plt.show()
    for numberofIterations, best in zip(boardACOScores, bestACOScores):
        resultsACO.append([numberofIterations, boardType, best])

print()
print(f"wyniki dla algorytmu genetycznego")
table = makeTable(results, ["Ilość iteracji", "Plansza", "Znalezione najlepsze rozwiązań"])
print()
printTable(table)
print(f"Średnia liczba iteracji: {np.mean([value[0] for value in results])}")
print(f"Odchylenie standardowe liczby iteracji: {np.std([value[0] for value in results])}
```

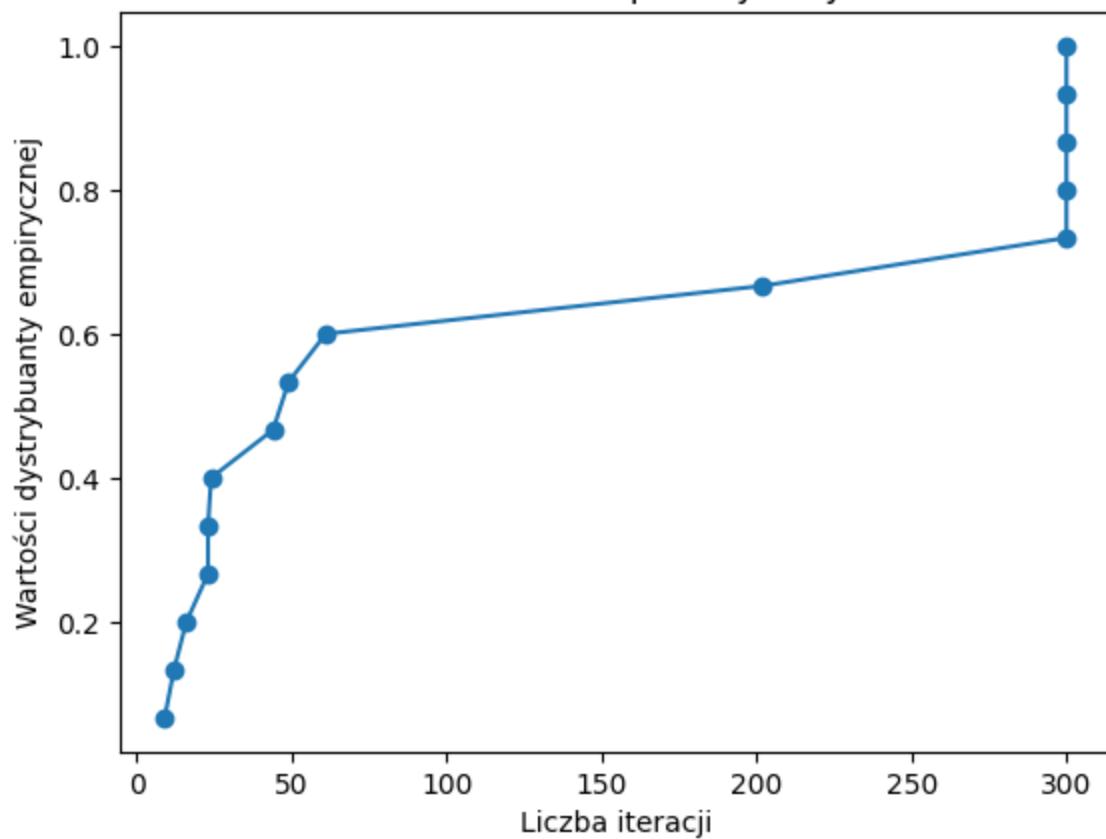
```

print("Najlepsze znalezione rozwiązanie: {min([value[2] for value in results])}")
print("Średnia liczba iteracji dla prostej planszy: {np.mean([value[0] for value in res
print("Odchylenie standardowe liczby iteracji dla prostej planszy: {np.std([value[0] fo
print("Najlepsze znalezione rozwiązanie dla prostej planszy: {min([value[2] for value i
print("Średnia liczba iteracji dla średniej planszy: {np.mean([value[0] for value in re
print("Odchylenie standardowe liczby iteracji dla średniej planszy: {np.std([value[0] fo
print("Najlepsze znalezione rozwiązanie dla średniej planszy: {min([value[2] for value
print("Średnia liczba iteracji dla trudnej planszy: {np.mean([value[0] for value in res
print("Odchylenie standardowe liczby iteracji dla trudnej planszy: {np.std([value[0] fo
print("Najlepsze znalezione rozwiązanie dla trudnej planszy: {min([value[2] for value i
print()
print("Wyniki dla algorytmu ACO")
table = makeTable(resultsACO, ["Ilość iteracji", "Plansza", "Znalezione najlepsze rozwia
print()
printTable(table)
print("Średnia liczba iteracji: {np.mean([value[0] for value in resultsACO])}")
print("Odchylenie standardowe liczby iteracji: {np.std([value[0] for value in resultsAC
print("Najlepsze znalezione rozwiązanie: {min([value[2] for value in resultsACO])}")
print("Średnia liczba iteracji dla prostej planszy: {np.mean([value[0] for value in res
print("Odchylenie standardowe liczby iteracji dla prostej planszy: {np.std([value[0] fo
print("Najlepsze znalezione rozwiązanie dla prostej planszy: {min([value[2] for value i
print("Średnia liczba iteracji dla średniej planszy: {np.mean([value[0] for value in re
print("Odchylenie standardowe liczby iteracji dla średniej planszy: {np.std([value[0] fo
print("Najlepsze znalezione rozwiązanie dla średniej planszy: {min([value[2] for value
print("Średnia liczba iteracji dla trudnej planszy: {np.mean([value[0] for value in res
print("Odchylenie standardowe liczby iteracji dla trudnej planszy: {np.std([value[0] fo
print("Najlepsze znalezione rozwiązanie dla trudnej planszy: {min([value[2] for value i

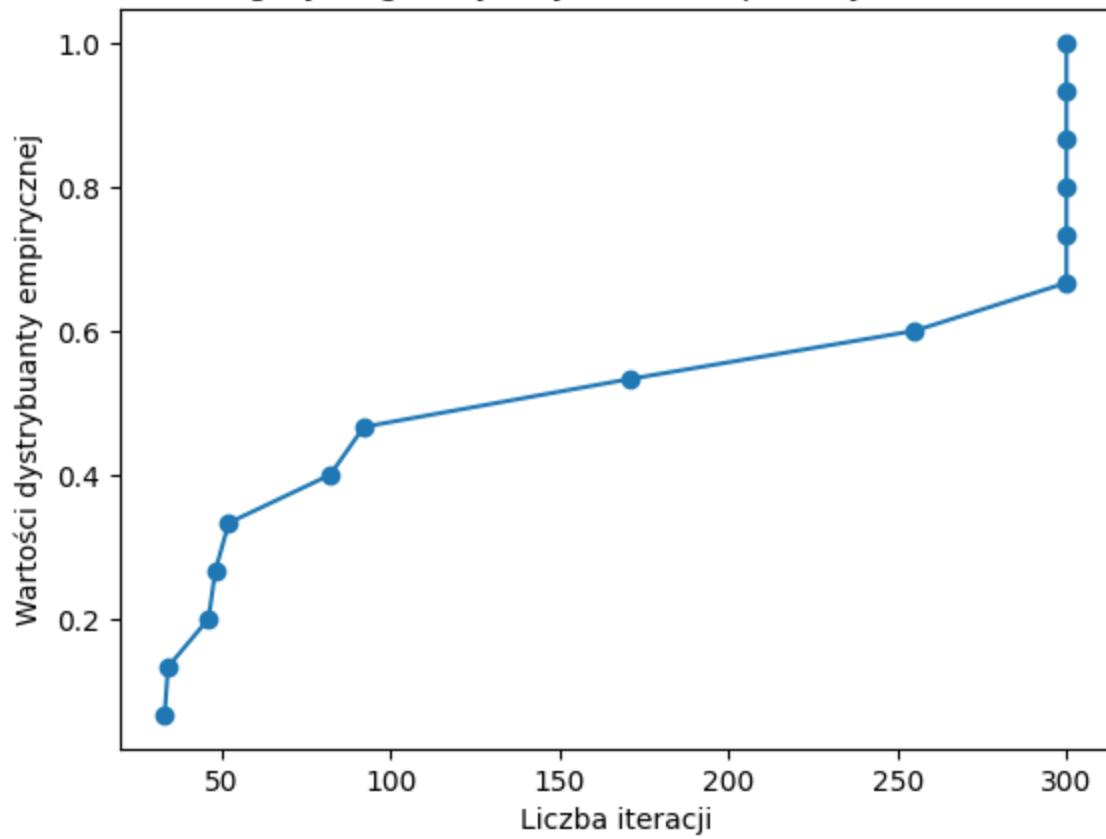
```



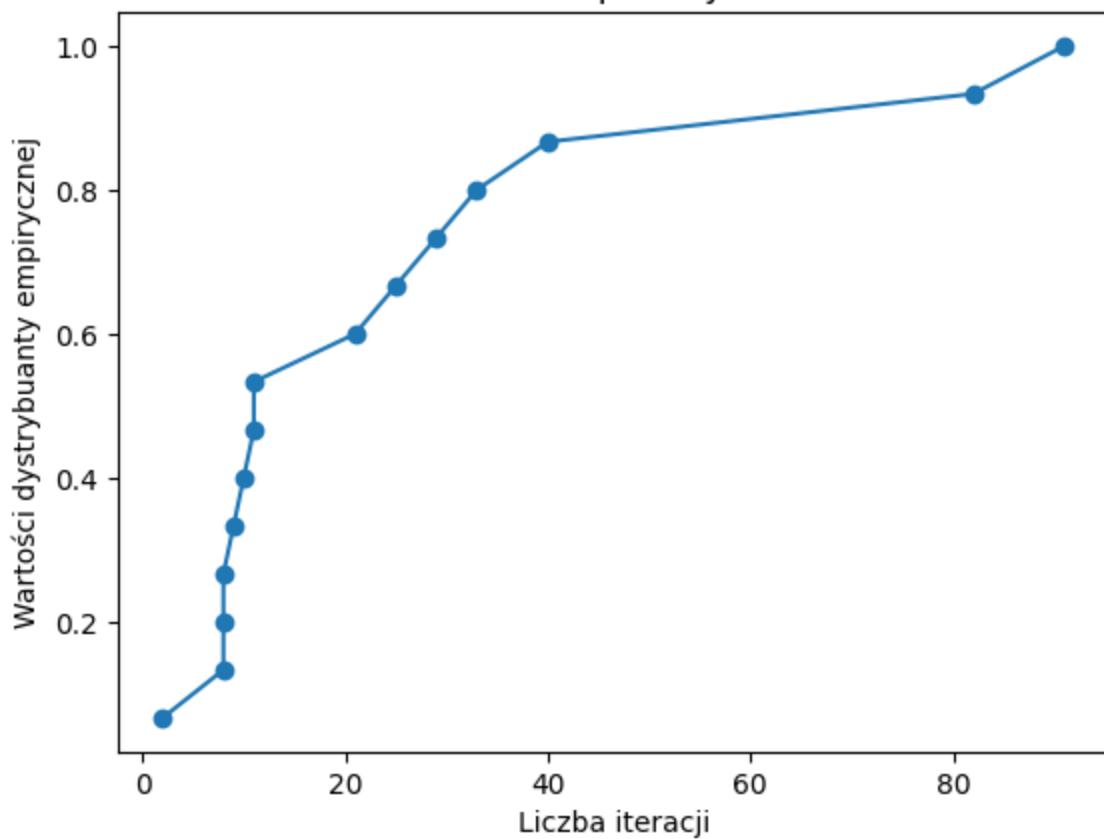
ACO ECDF dla planszy easy



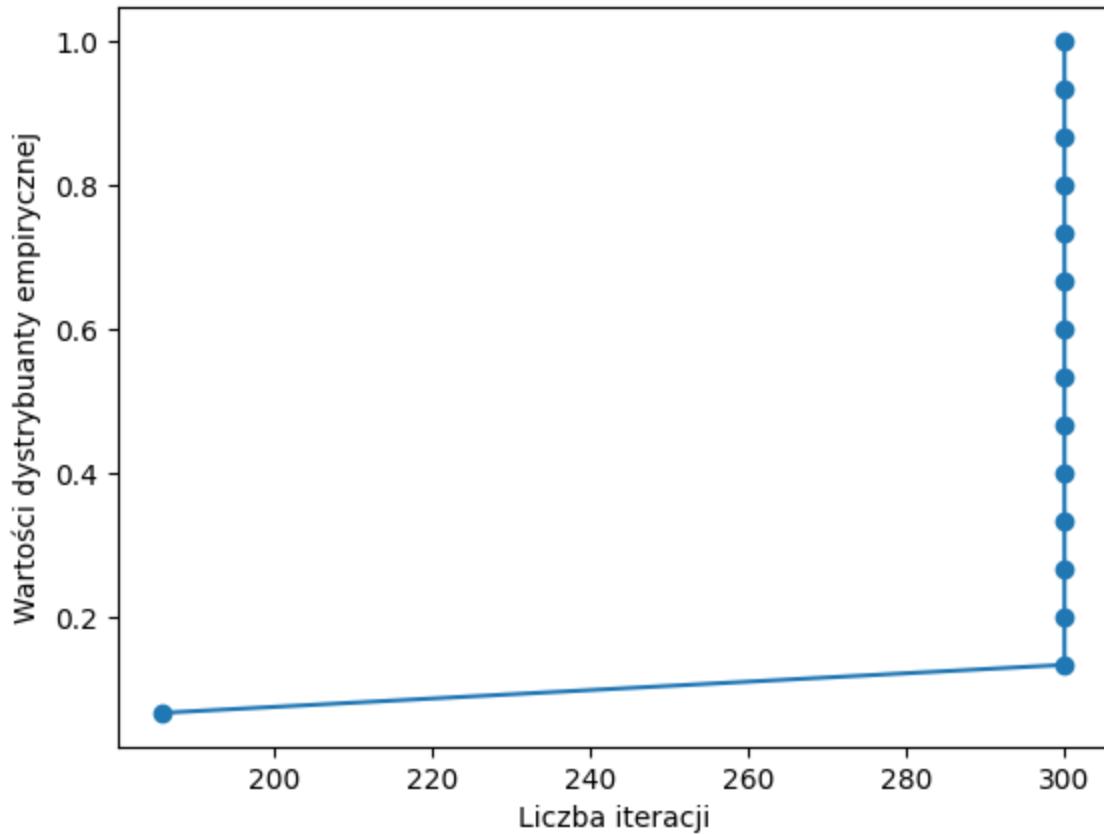
Algorytm genetyczny ECDF dla planszy medium



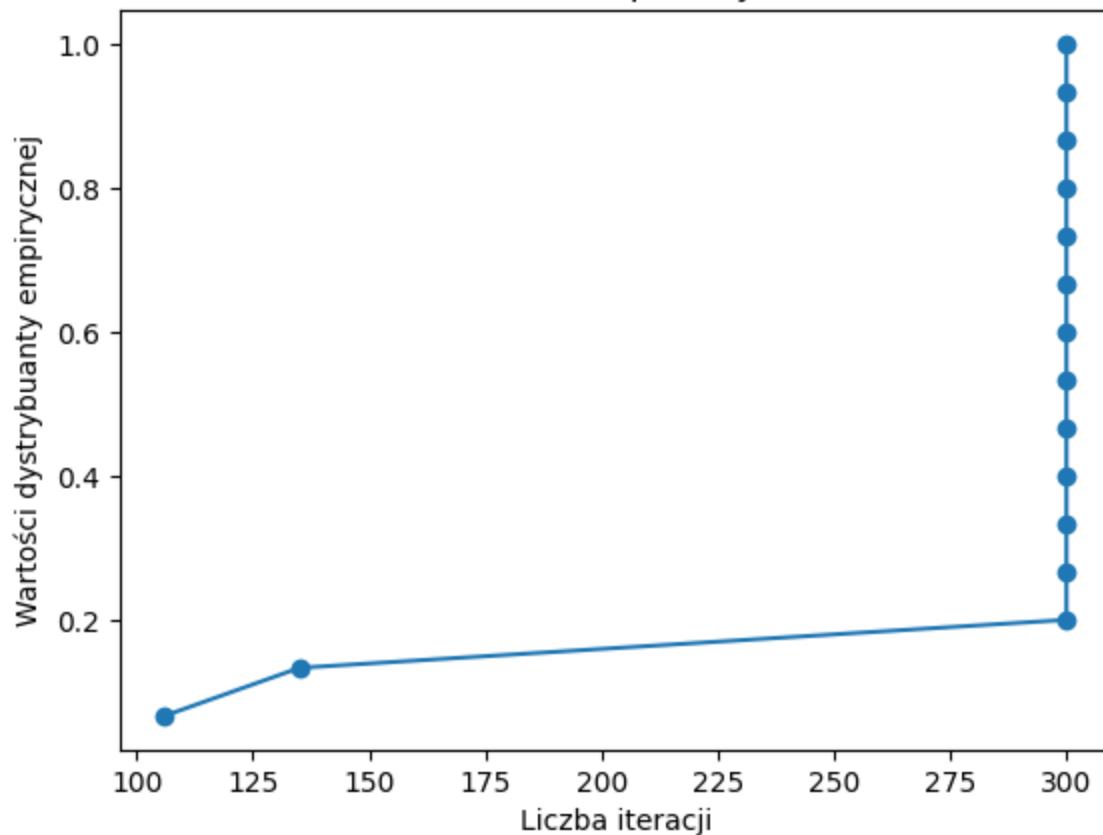
ACO ECDF dla planszy medium



Algorytm genetyczny ECDF dla planszy hard



ACO ECDF dla planszy hard



Wyniki dla algorytmu genetycznego

Ilość iteracji	Plansza	Znalezione najlepsze rozwiązanie
13	easy	0
19	easy	0
11	easy	0
300	easy	3
22	easy	0
22	easy	0
20	easy	0
78	easy	0
33	easy	0
26	easy	0
26	easy	0
8	easy	0
300	easy	1
16	easy	0
147	easy	0
300	medium	1

300	medium	1
34	medium	0
46	medium	0
33	medium	0
171	medium	0
48	medium	0
300	medium	1
300	medium	2
300	medium	3
255	medium	0
92	medium	0
300	medium	3
52	medium	0
82	medium	0
300	hard	2
300	hard	3
300	hard	5
186	hard	0
300	hard	1
300	hard	2
300	hard	2
300	hard	1
300	hard	3
300	hard	1
300	hard	3
300	hard	2
300	hard	3
300	hard	1
300	hard	3

Średnia liczba iteracji: 178.66666666666666

Odchylenie standardowe liczby iteracji: 127.3151819523326

Najlepsze znalezione rozwiązanie: 0

Średnia liczba iteracji dla prostej planszy: 69.4

Odchylenie standardowe liczby iteracji dla prostej planszy: 96.68767587788358

Najlepsze znalezione rozwiązanie dla prostej planszy: 0

Średnia liczba iteracji dla średniej planszy: 174.2

Odchylenie standardowe liczby iteracji dla średniej: 116.51563557451564

Najlepsze znalezione rozwiązanie dla średniej planszy: 0
Średnia liczba iteracji dla trudnej planszy: 292.4
Odchylenie standardowe liczby iteracji dla trudnej planszy: 28.436596139481953
Najlepsze znalezione rozwiązanie dla trudnej planszy: 0

Wyniki dla algorytmu ACO

Ilość iteracji	Plansza	Znalezione najlepsze rozwiązanie
300	easy	4
12	easy	0
49	easy	0
9	easy	0
300	easy	4
202	easy	0
61	easy	0
44	easy	0
24	easy	0
300	easy	4
23	easy	0
16	easy	0
300	easy	4
23	easy	0
300	easy	4
2	medium	0
11	medium	0
82	medium	0
11	medium	0
29	medium	0
33	medium	0
25	medium	0
8	medium	0
8	medium	0
21	medium	0
40	medium	0
91	medium	0
8	medium	0

10	medium	0
9	medium	0
300	hard	3
300	hard	17
300	hard	9
300	hard	3
300	hard	5
300	hard	8
300	hard	3
300	hard	10
300	hard	3
135	hard	0
106	hard	0
300	hard	4
300	hard	12
300	hard	4
300	hard	5

Średnia liczba iteracji: 144.26666666666668

Odchylenie standardowe liczby iteracji: 131.99989898986036

Najlepsze znalezione rozwiązań: 0

Średnia liczba iteracji dla prostej planszy: 130.86666666666667

Odchylenie standardowe liczby iteracji dla prostej planszy: 127.56324792910466

Najlepsze znalezione rozwiązań dla prostej planszy: 0

Średnia liczba iteracji dla średniej planszy: 25.866666666666667

Odchylenie standardowe liczby iteracji dla średniej: 26.05600805103413

Najlepsze znalezione rozwiązań dla średniej planszy: 0

Średnia liczba iteracji dla trudnej planszy: 276.06666666666666

Odchylenie standardowe liczby iteracji dla trudnej planszy: 61.247548703782606

Najlepsze znalezione rozwiązań dla trudnej planszy: 0

Testy szybkości przy ustalonym budżecie wywołań funkcji celu

Znowu tworzymy wykresy ECDF ze względu na otrzymane rozwiązania, jednakże tym razem badamy wpływ maksymalnej ilości ewaluacji.

To samo znajduje się w pliku max_evaluations_ecdf.py

In [4]:

```
results = []
resultsACO = []
maxEvaluations = [5000, 10000, 15000, 20000]
for boardStr, boardType in zip(boardStrs, ["easy", "medium", "hard"]):
    for evaluation in maxEvaluations:
        bestScores = []
        bestACOScores = []
        for experimentNumber in range(15):
            pop0 = makeBoardsFromFile(boardStr, pop0Size=bestArgs[boardType][0], random_
argsList = [pop0, 20000]
```

```

argsList.extend(bestArgs[boardType][2:])
bestScore, bestSollution, scores, evaluations = geneticAlgorithm(*argsList,
bestScores.append(bestScore)
solver = Solver(*bestArgsACO[boardType], max_iterations=20000, max_evaluatio
board = create_board_from_str(boardStr)
bestScore, bestSollution, scores, evaluations, iterations = solver.solve(boa
bestACOScores.append(bestScore)

ecdf = ECDF(bestScores)
plt.plot(ecdf.x, ecdf.y, marker='o', linestyle='--')
plt.xlabel("Wartości funkcji celu")
plt.ylabel('Wartości dystrybuanty empirycznej')
plt.title(f"Algorytm genetyczny ECDF dla planszy {boardType} liczba ewaluacji f.
plt.show()

numberOfBestScores = len([value for value in bestScores if value == 0])
results.append([numberOfBestScores, boardType, evaluation])

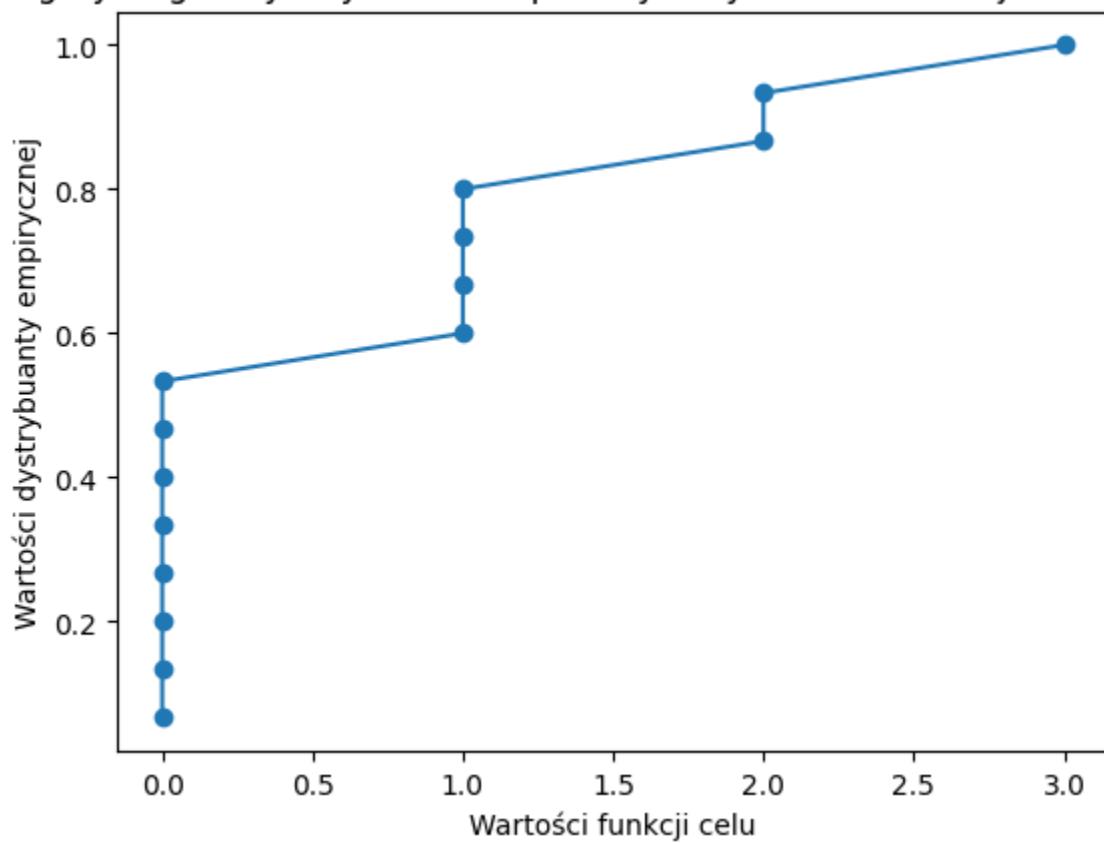
ecdf = ECDF(bestACOScores)
plt.plot(ecdf.x, ecdf.y, marker='o', linestyle='--')
plt.xlabel("Wartości funkcji celu")
plt.ylabel('Wartości dystrybuanty empirycznej')
plt.title(f"ACO ECDF dla planszy {boardType} liczba ewaluacji f. celu {evaluatio
plt.show()

numberOfBestScores = len([value for value in bestACOScores if value == 0])
resultsACO.append([numberOfBestScores, boardType, evaluation])

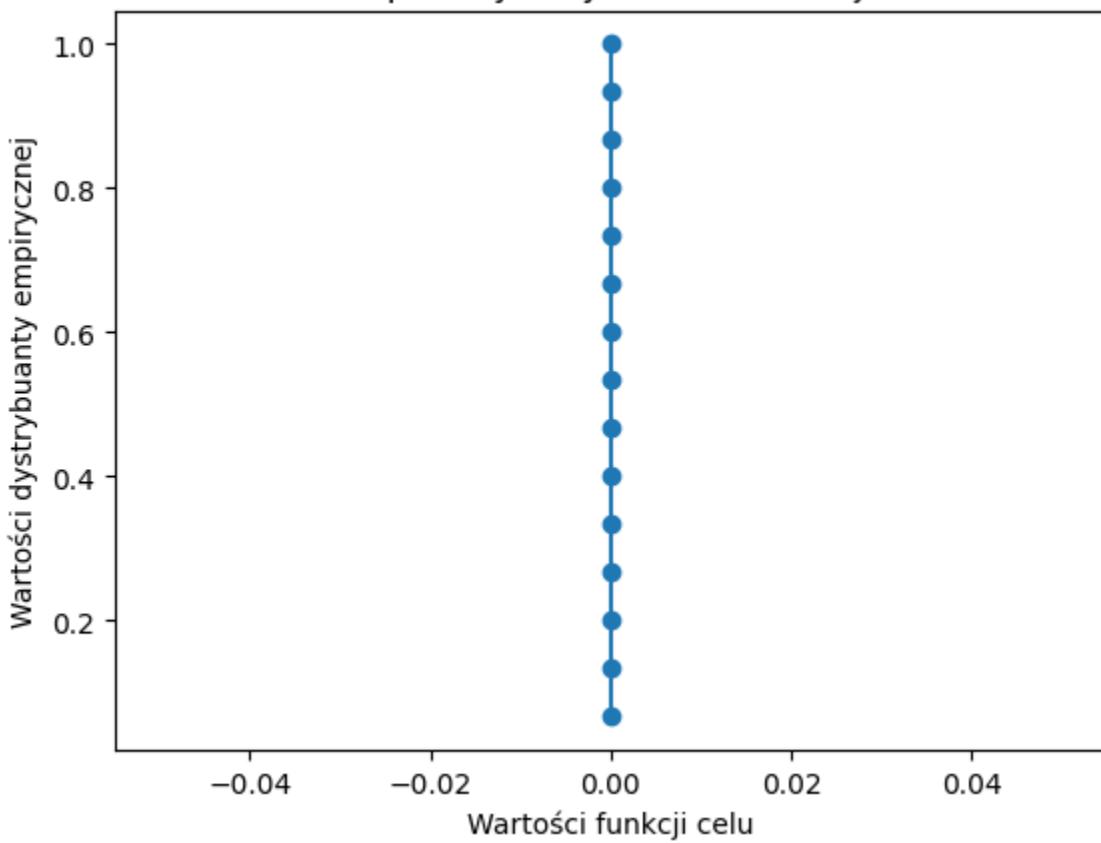
print()
print("Wyniki dla algorytmu genetycznego")
table = makeTable(results, ["Znaleziono najlepsze rozwiązania", "Plansza", "Liczba ewalu
print()
printTable(table)
print(f"Średnia znalezionych rozwiązań: {np.mean([value[0] for value in results])}")
print(f"Odchylenie standardowe znalezionych rozwiązań: {np.std([value[0] for value in re
print(f"Średnia znalezionych rozwiązań dla prostej planszy: {np.mean([value[0] for value in
print(f"Odchylenie standardowe znalezionych rozwiązań dla prostej planszy: {np.std([valu
print(f"Średnia znalezionych rozwiązań dla średniej planszy: {np.mean([value[0] for valu
print(f"Odchylenie standardowe znalezionych rozwiązań dla średniej: {np.std([value[0] fo
print(f"Średnia znalezionych rozwiązań dla trudnej planszy: {np.mean([value[0] for value
print(f"Odchylenie standardowe znalezionych rozwiązań dla trudnej planszy: {np.std([valu
print()
print("Wyniki dla algorytmu ACO")
table = makeTable(resultsACO, ["Znaleziono najlepsze rozwiązania", "Plansza", "Liczba ew
print()
printTable(table)
print(f"Średnia znalezionych rozwiązań: {np.mean([value[0] for value in resultsACO])}")
print(f"Odchylenie standardowe znalezionych rozwiązań: {np.std([value[0] for value in re
print(f"Średnia znalezionych rozwiązań dla prostej planszy: {np.mean([value[0] for value in
print(f"Odchylenie standardowe znalezionych rozwiązań dla prostej planszy: {np.std([valu
print(f"Średnia znalezionych rozwiązań dla średniej planszy: {np.mean([value[0] for valu
print(f"Odchylenie standardowe znalezionych rozwiązań dla średniej: {np.std([value[0] fo
print(f"Średnia znalezionych rozwiązań dla trudnej planszy: {np.mean([value[0] for value
print(f"Odchylenie standardowe znalezionych rozwiązań dla trudnej planszy: {np.std([valu

```

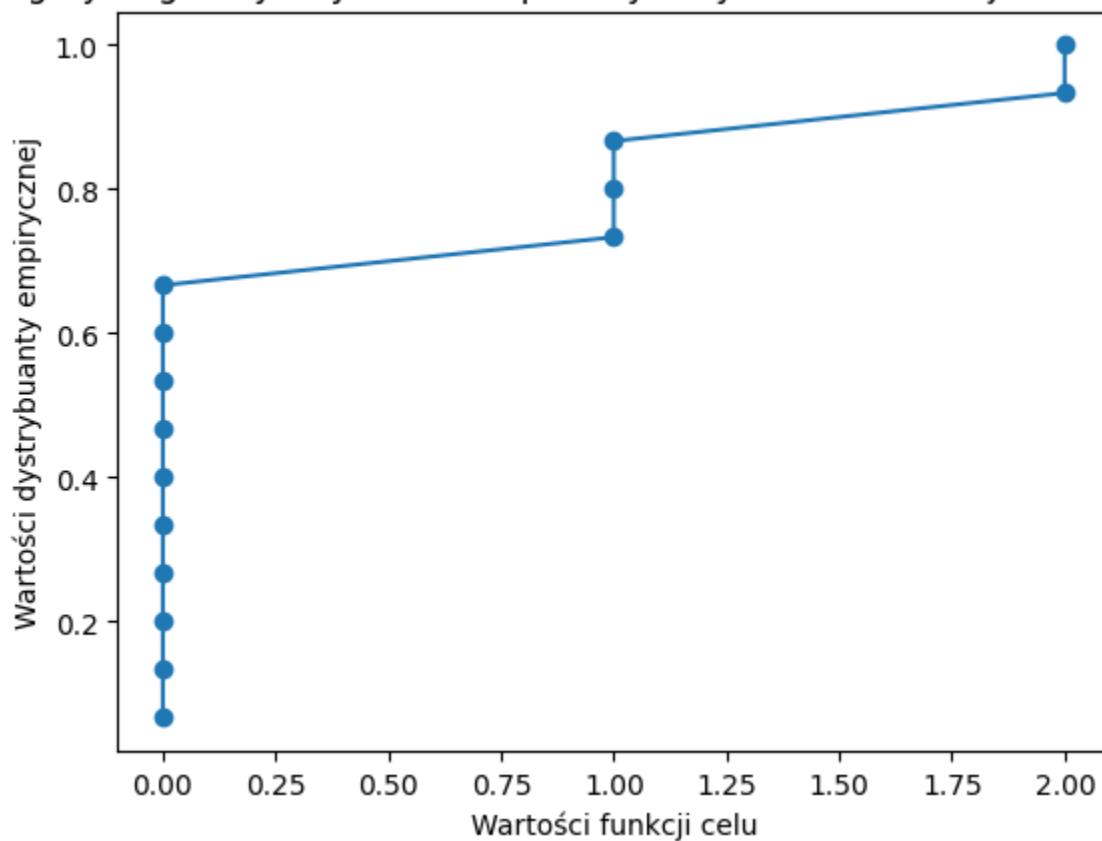
Algorytm genetyczny ECDF dla planszy easy liczba ewaluacji f. celu 5000



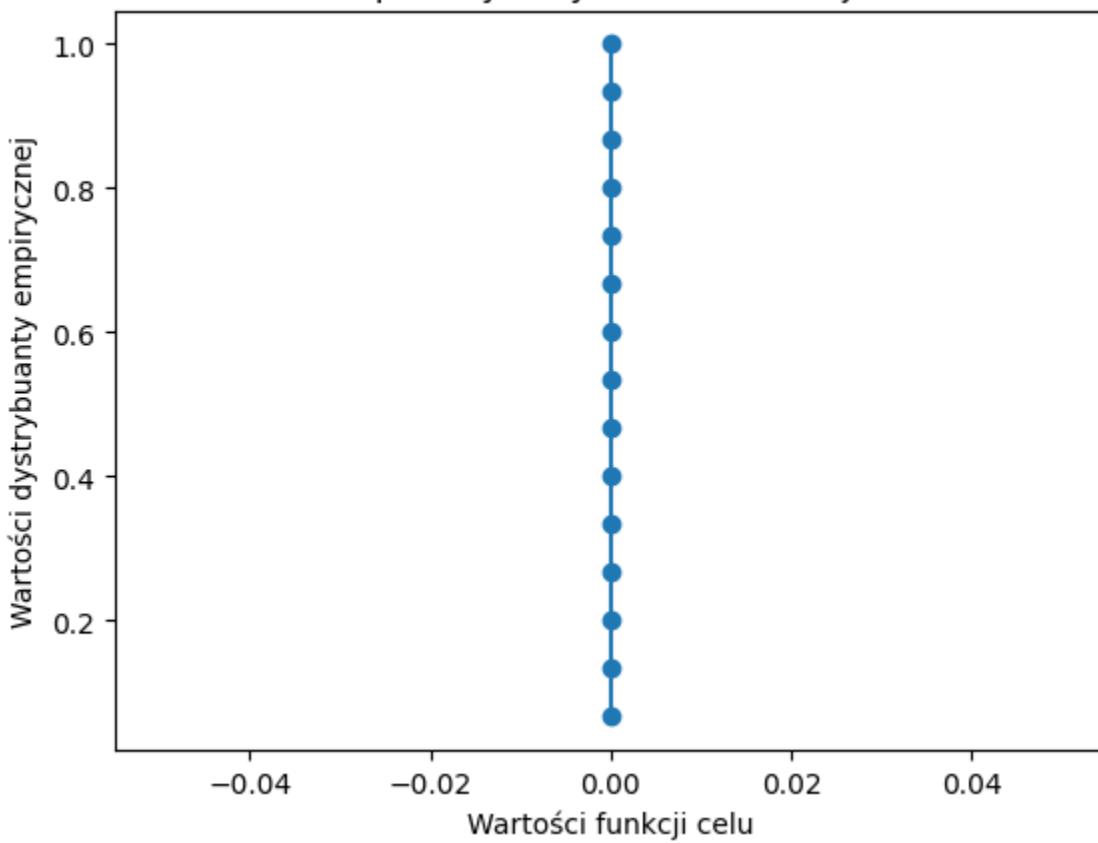
ACO ECDF dla planszy easy liczba ewaluacji f. celu 5000



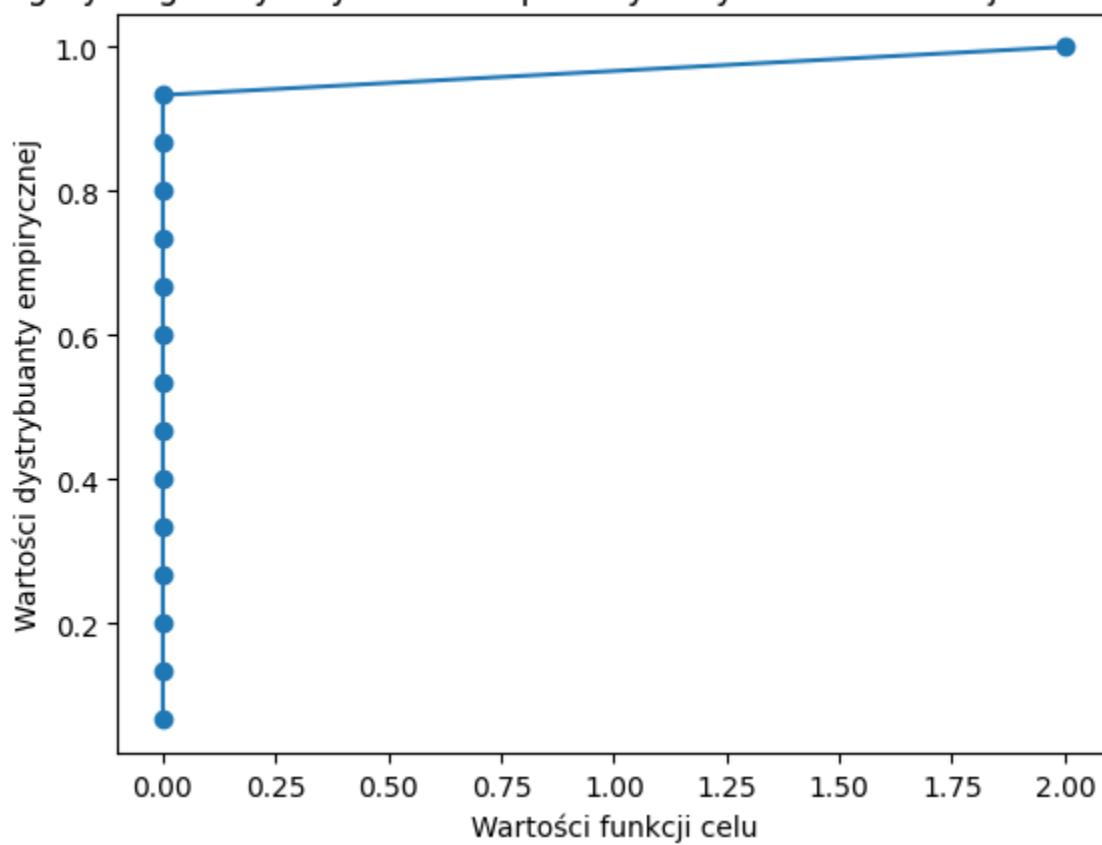
Algorytm genetyczny ECDF dla planszy easy liczba ewaluacji f. celu 10000



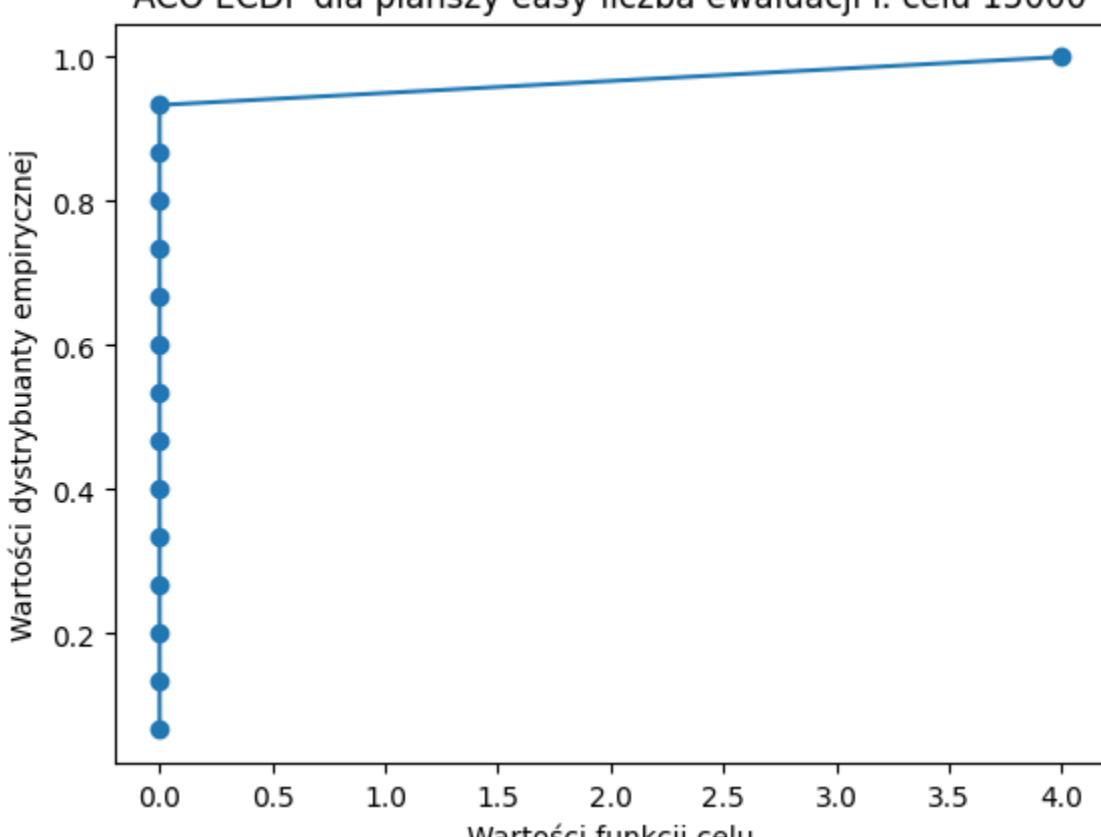
ACO ECDF dla planszy easy liczba ewaluacji f. celu 10000



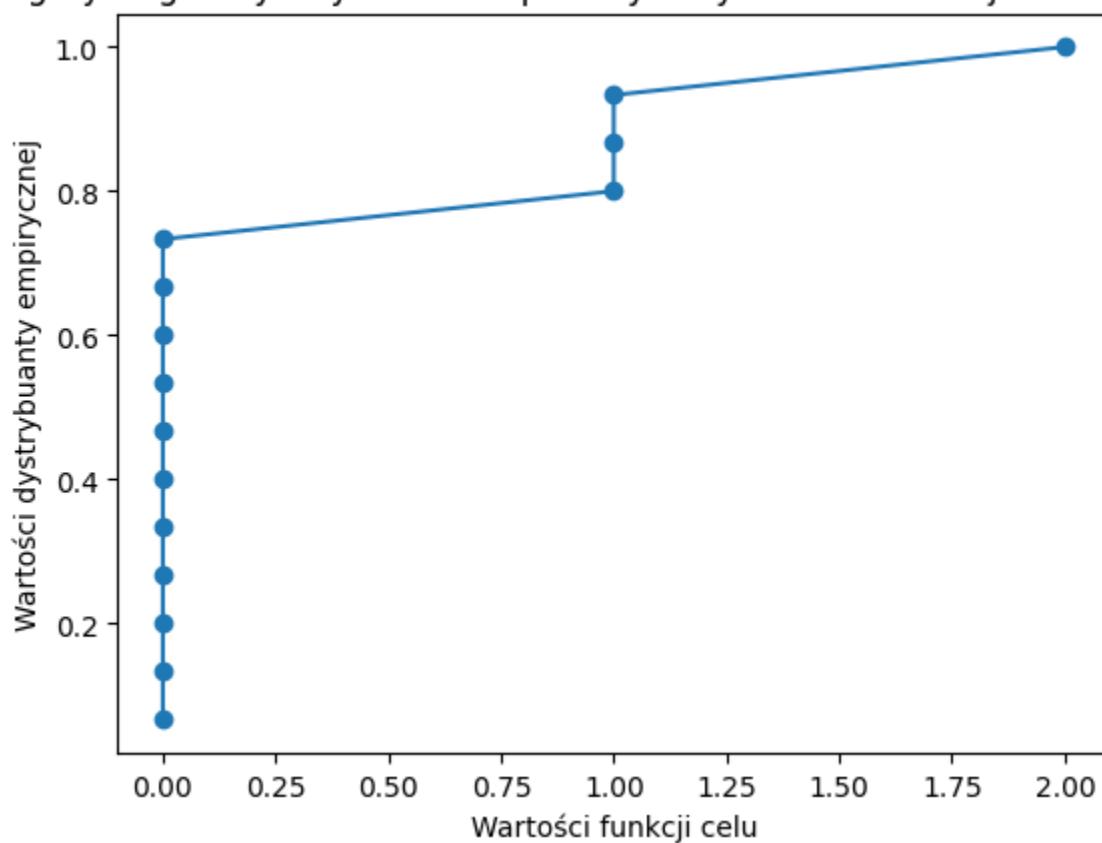
Algorytm genetyczny ECDF dla planszy easy liczba ewaluacji f. celu 15000



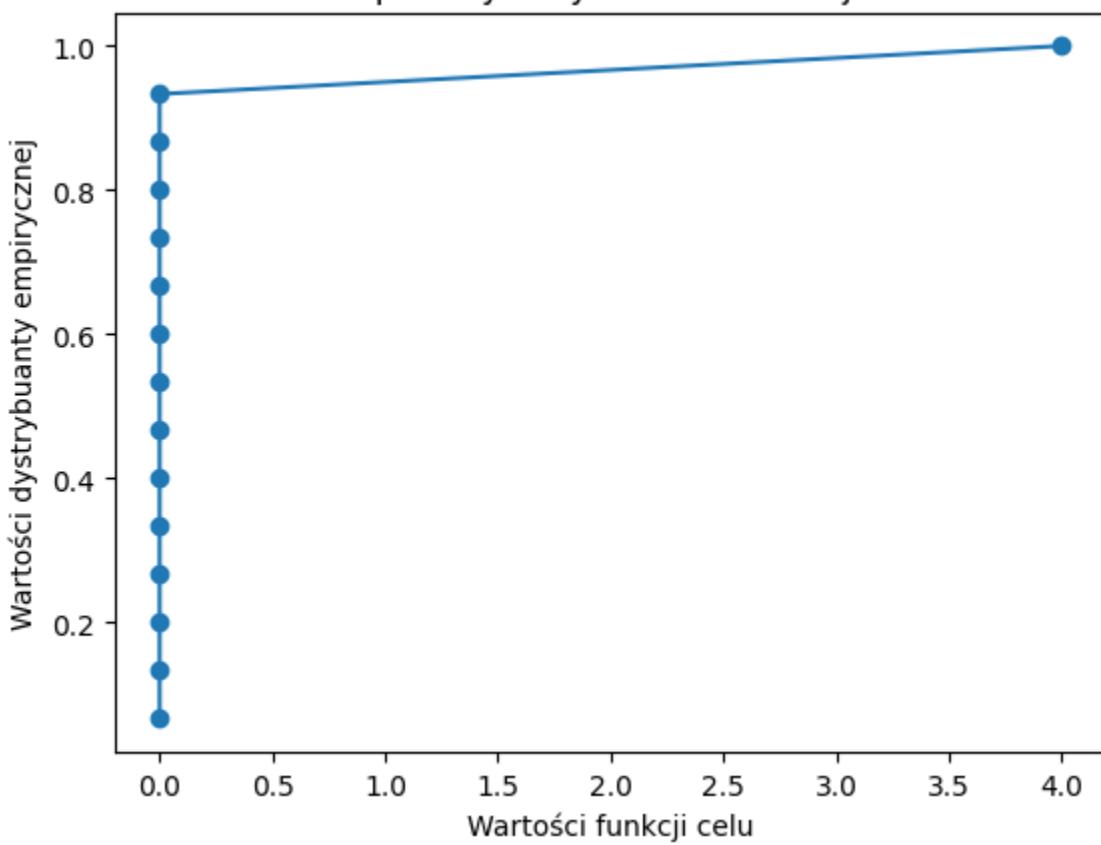
ACO ECDF dla planszy easy liczba ewaluacji f. celu 15000



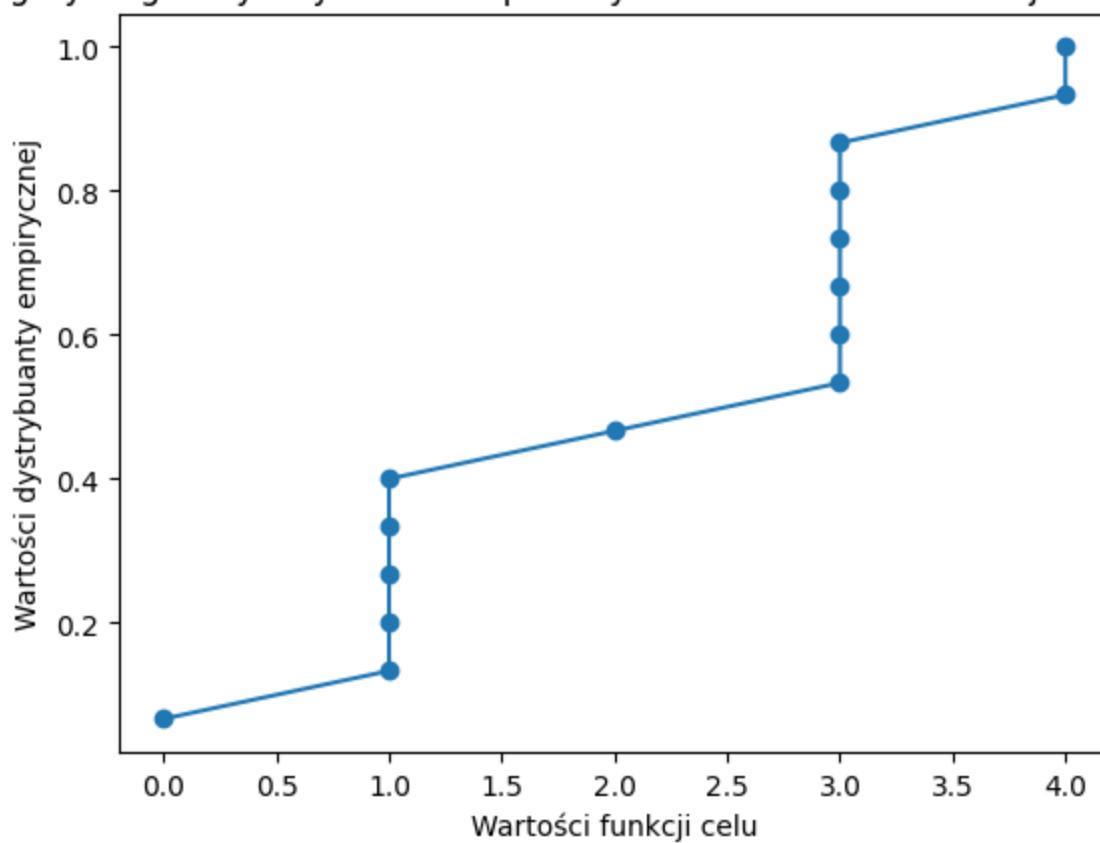
Algorytm genetyczny ECDF dla planszy easy liczba ewaluacji f. celu 20000



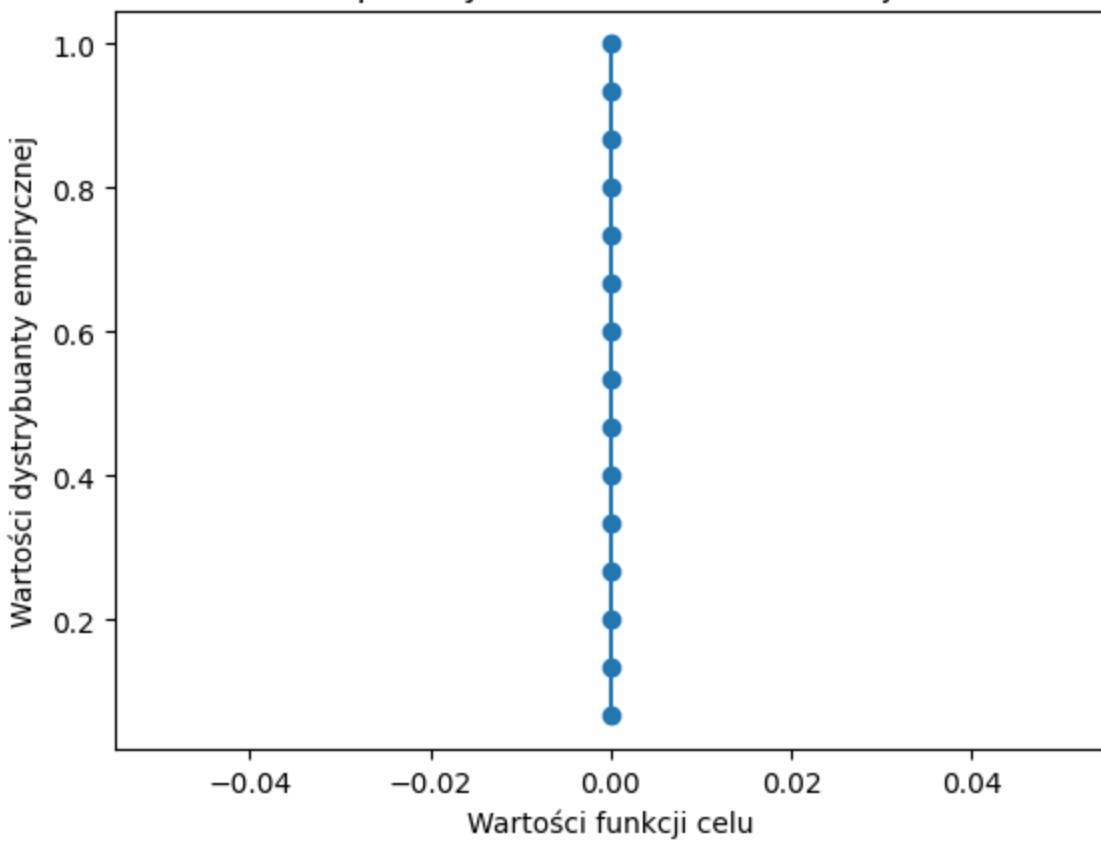
ACO ECDF dla planszy easy liczba ewaluacji f. celu 20000



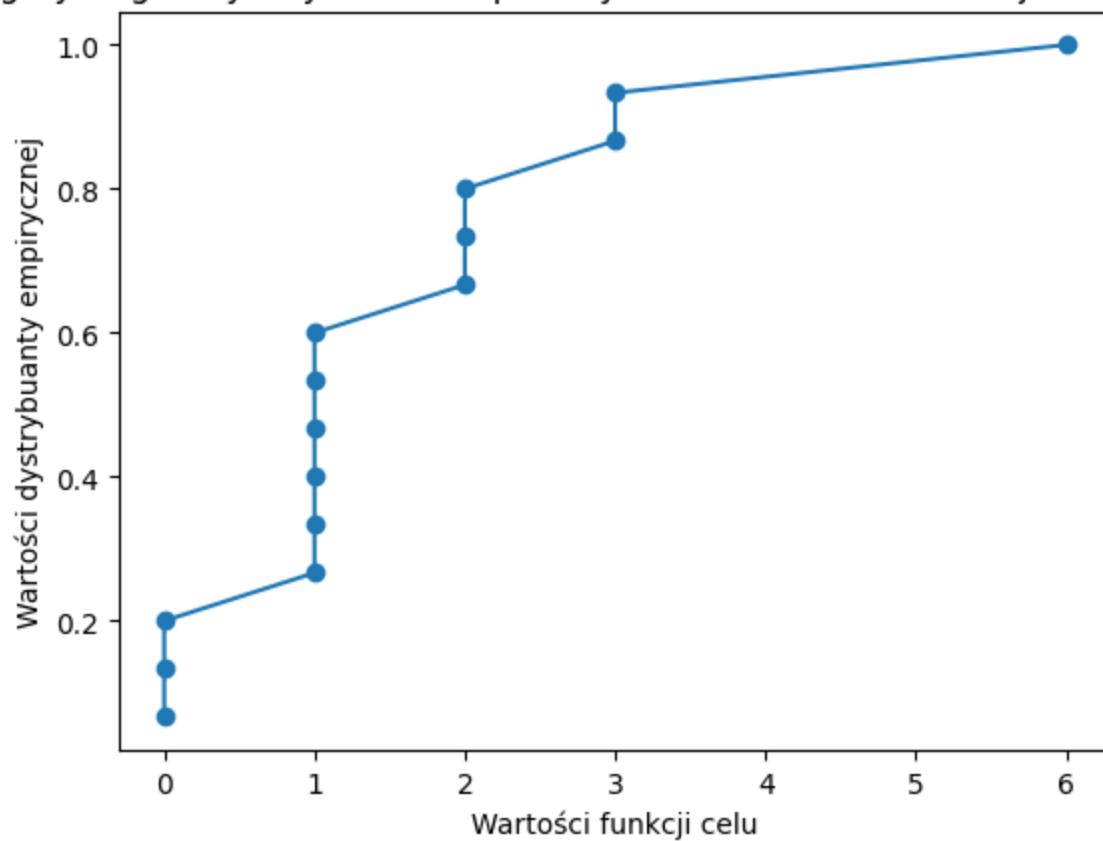
Algorytm genetyczny ECDF dla planszy medium liczba ewaluacji f. celu 5000



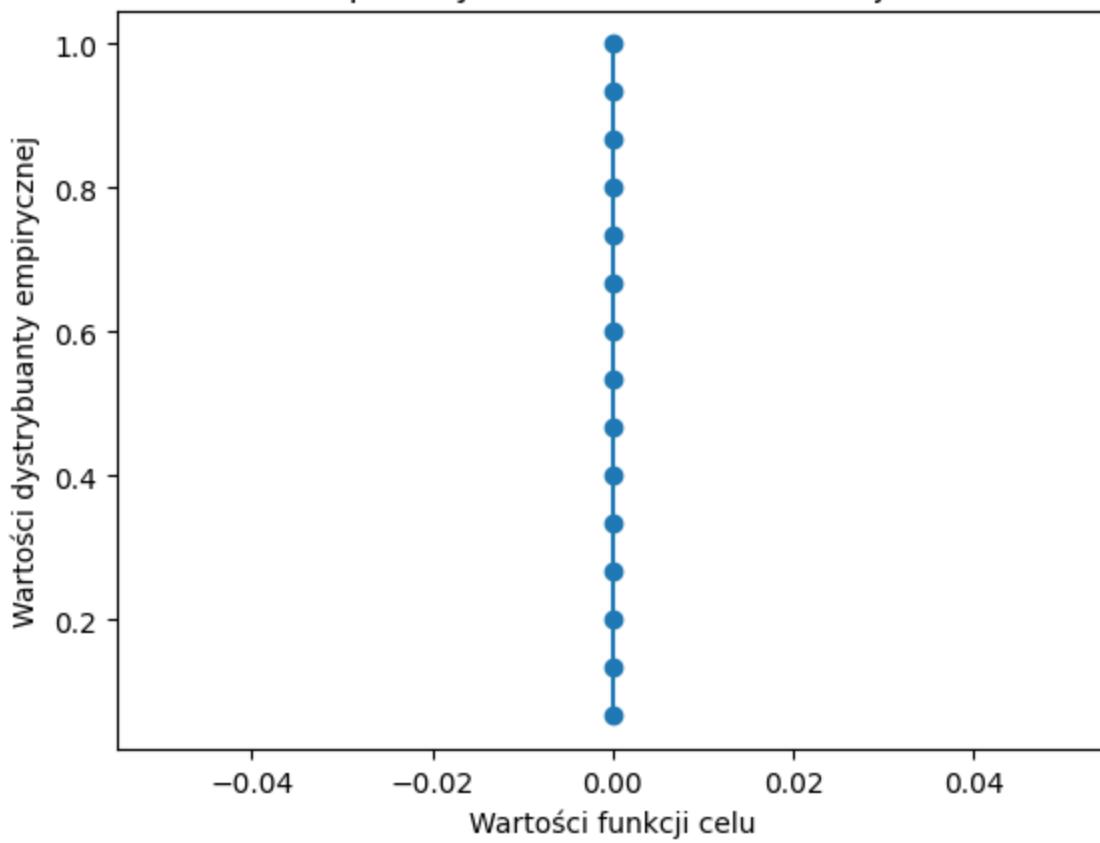
ACO ECDF dla planszy medium liczba ewaluacji f. celu 5000



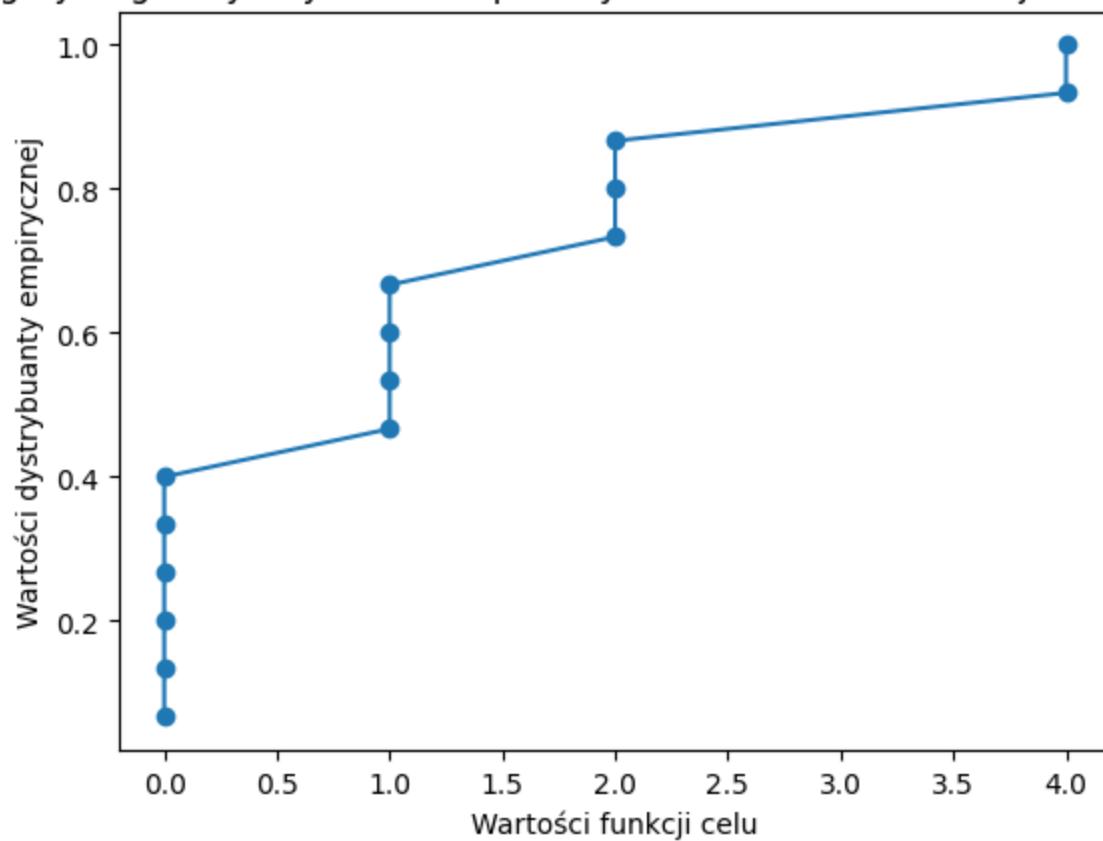
Algorytm genetyczny ECDF dla planszy medium liczba ewaluacji f. celu 10000



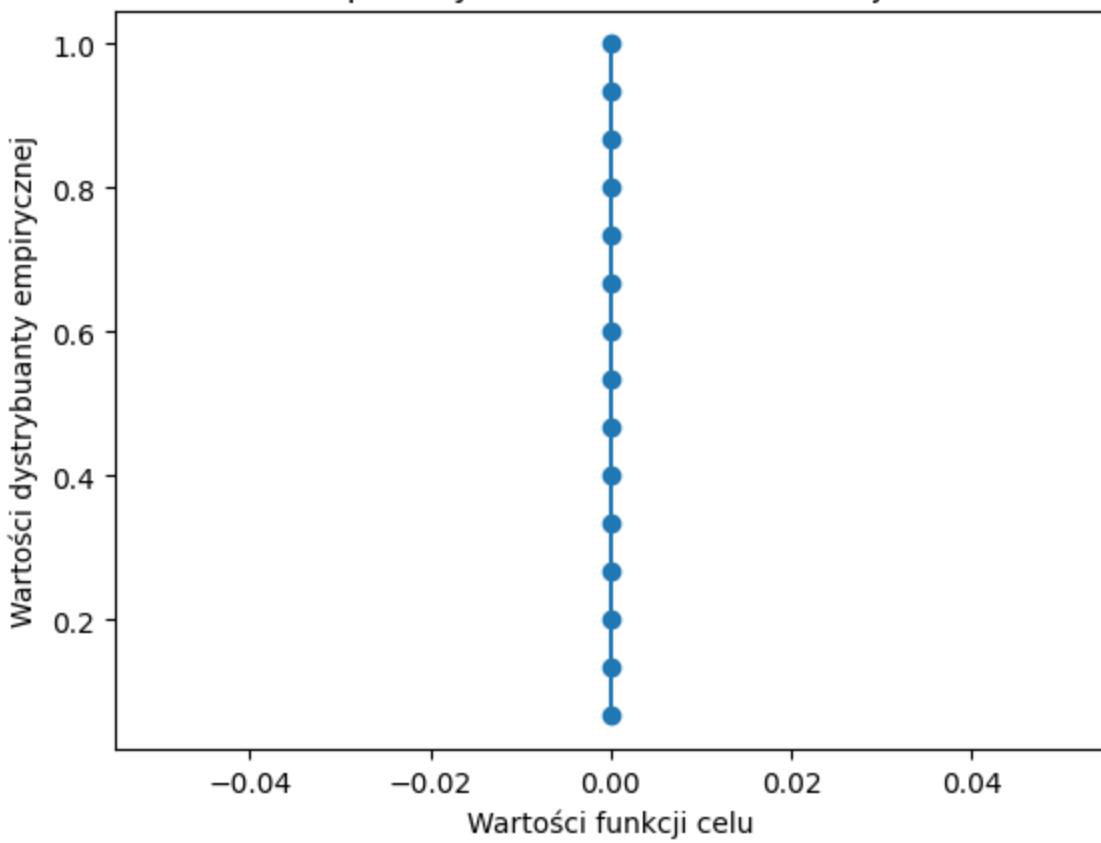
ACO ECDF dla planszy medium liczba ewaluacji f. celu 10000



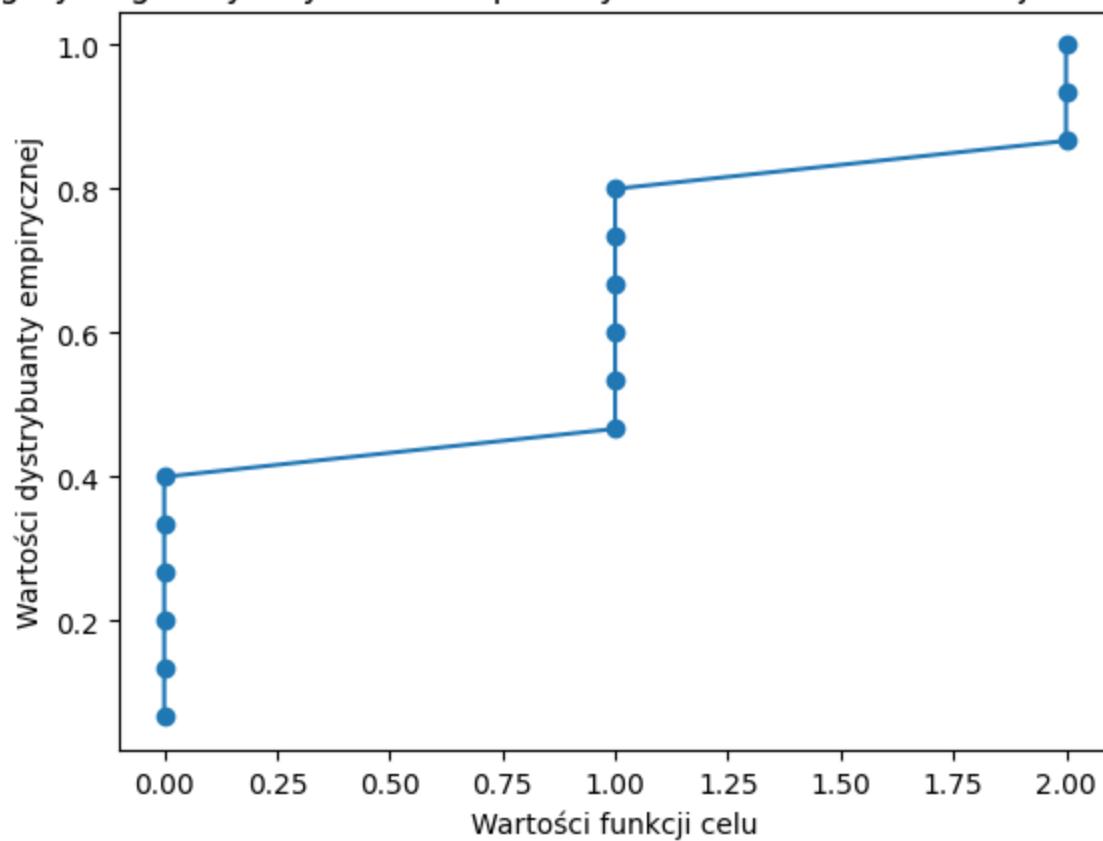
Algorytm genetyczny ECDF dla planszy medium liczba ewaluacji f. celu 15000



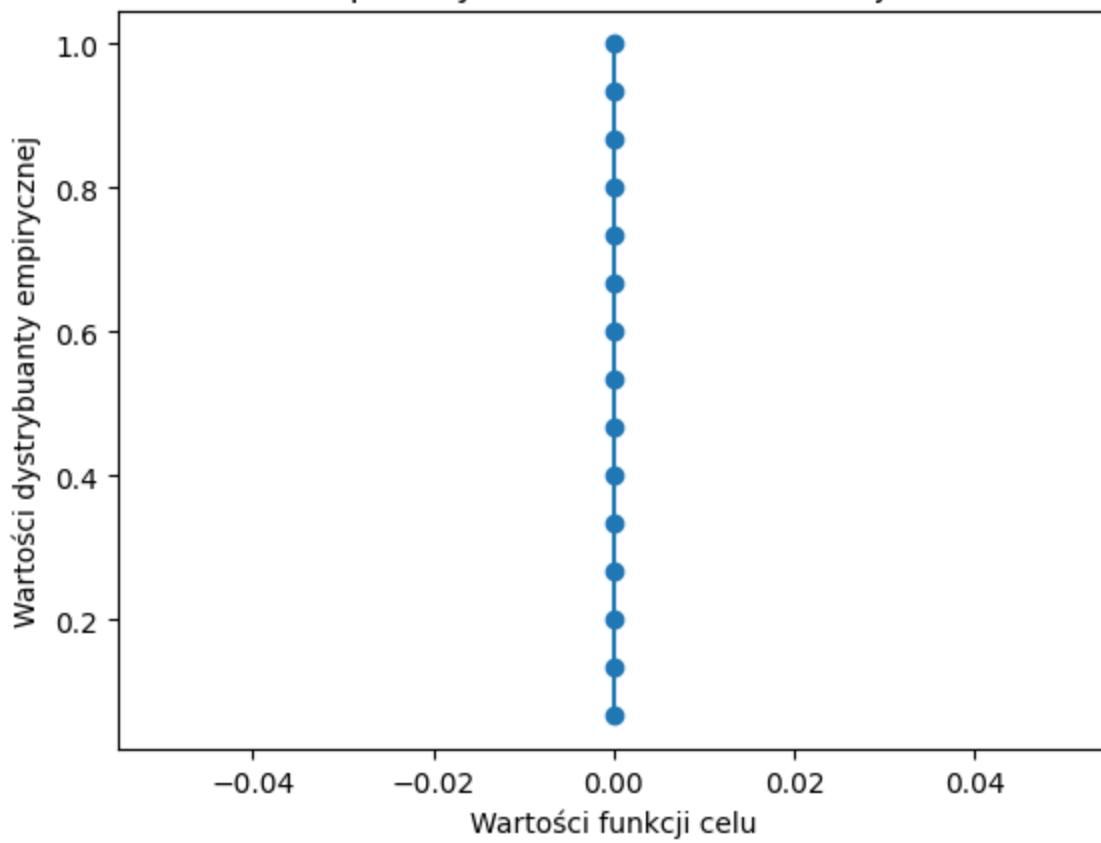
ACO ECDF dla planszy medium liczba ewaluacji f. celu 15000



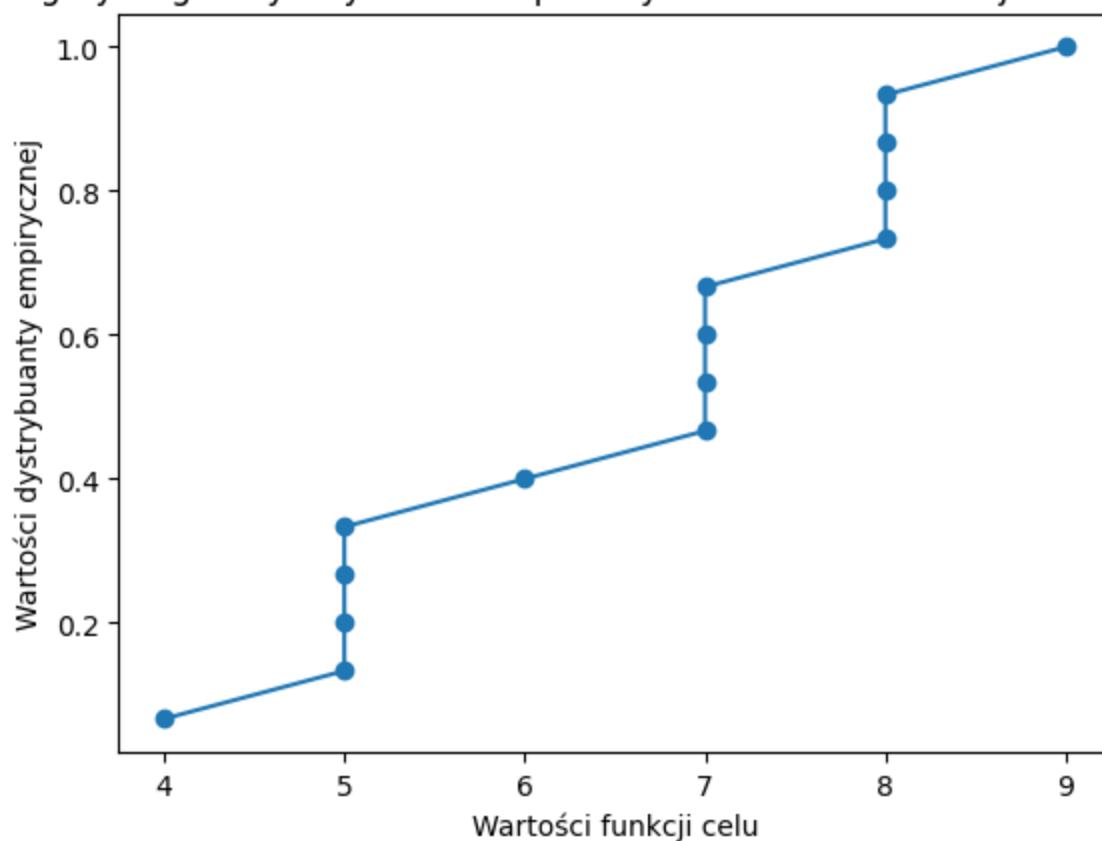
Algorytm genetyczny ECDF dla planszy medium liczba ewaluacji f. celu 20000



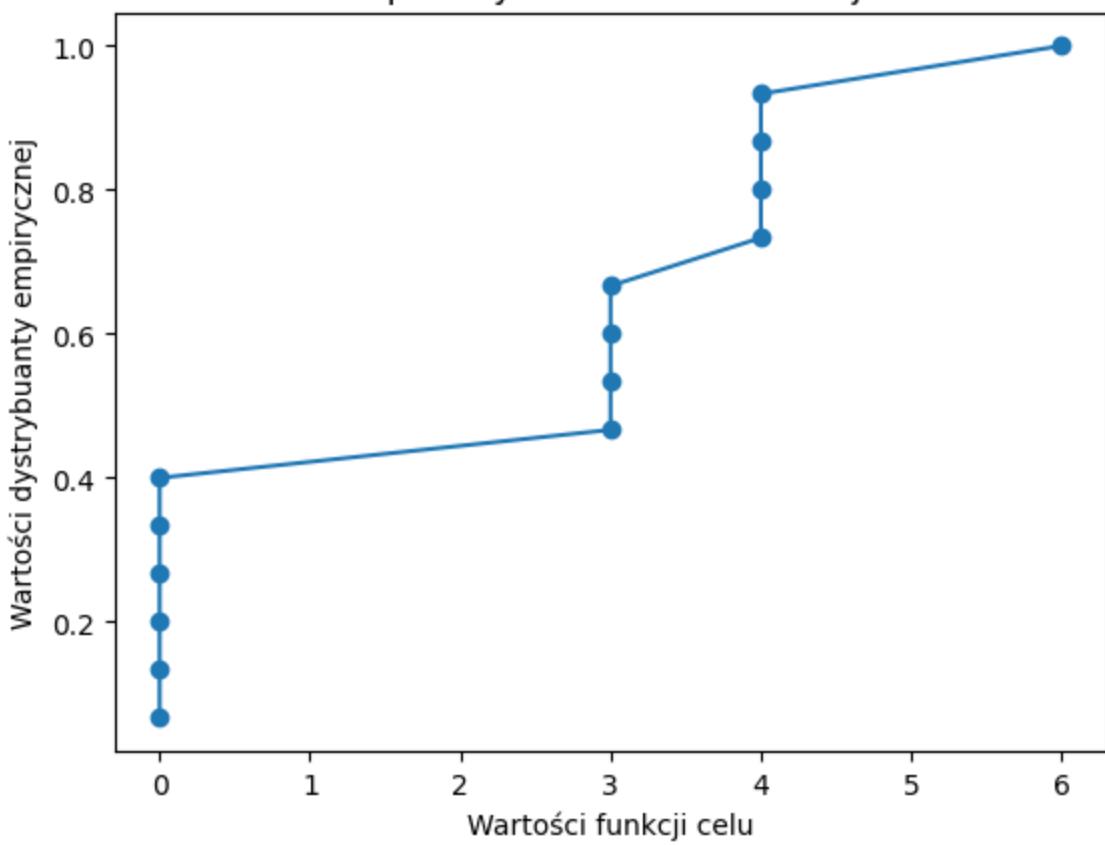
ACO ECDF dla planszy medium liczba ewaluacji f. celu 20000



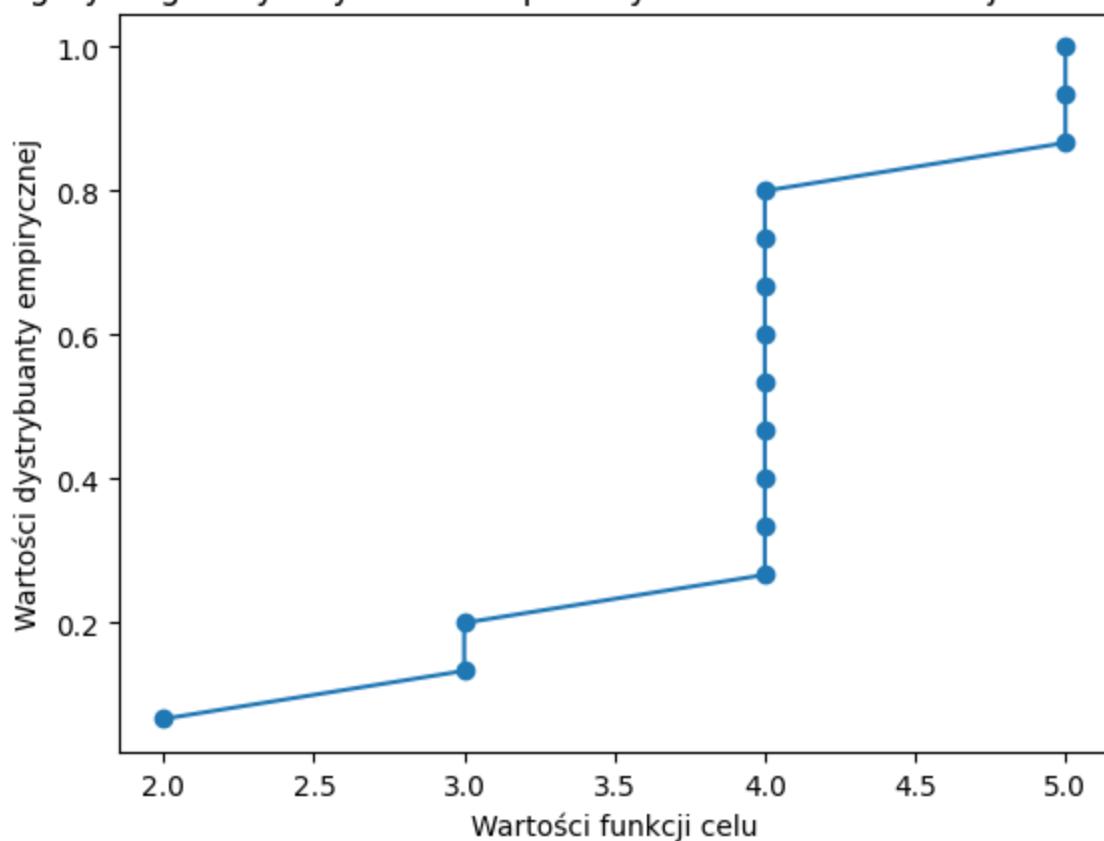
Algorytm genetyczny ECDF dla planszy hard liczba ewaluacji f. celu 5000



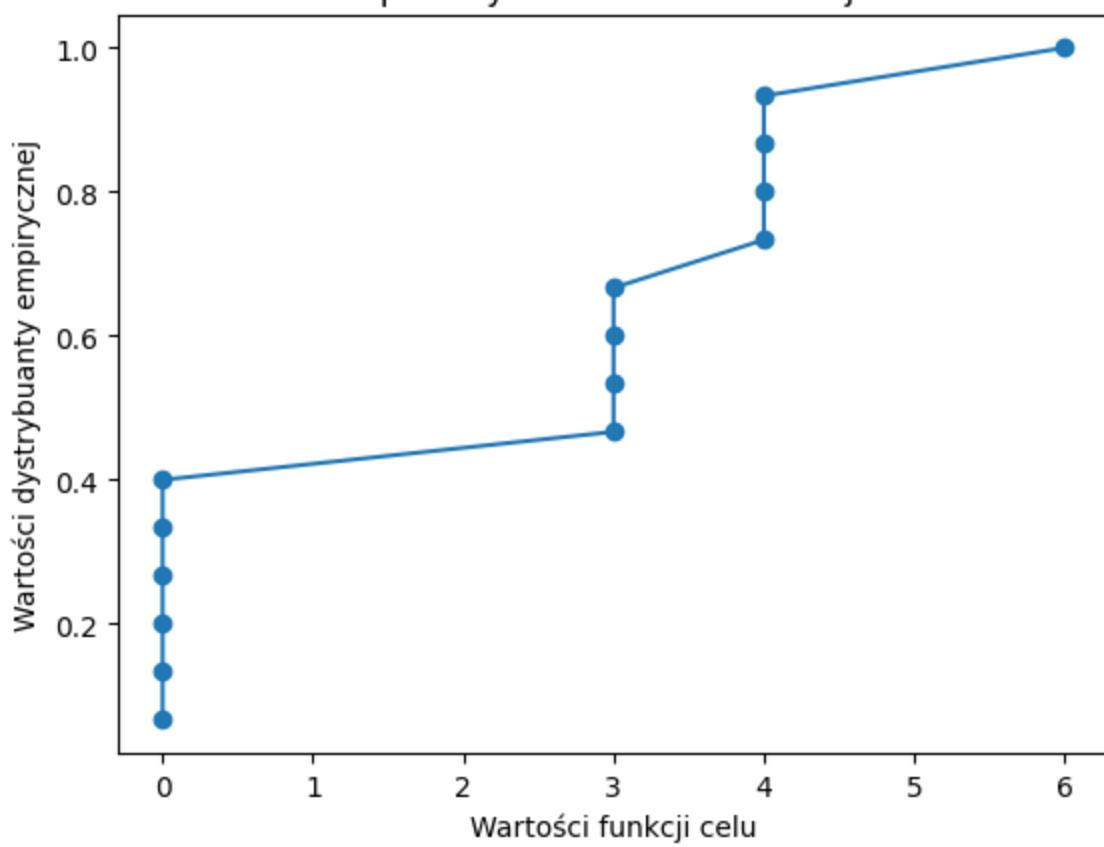
ACO ECDF dla planszy hard liczba ewaluacji f. celu 5000



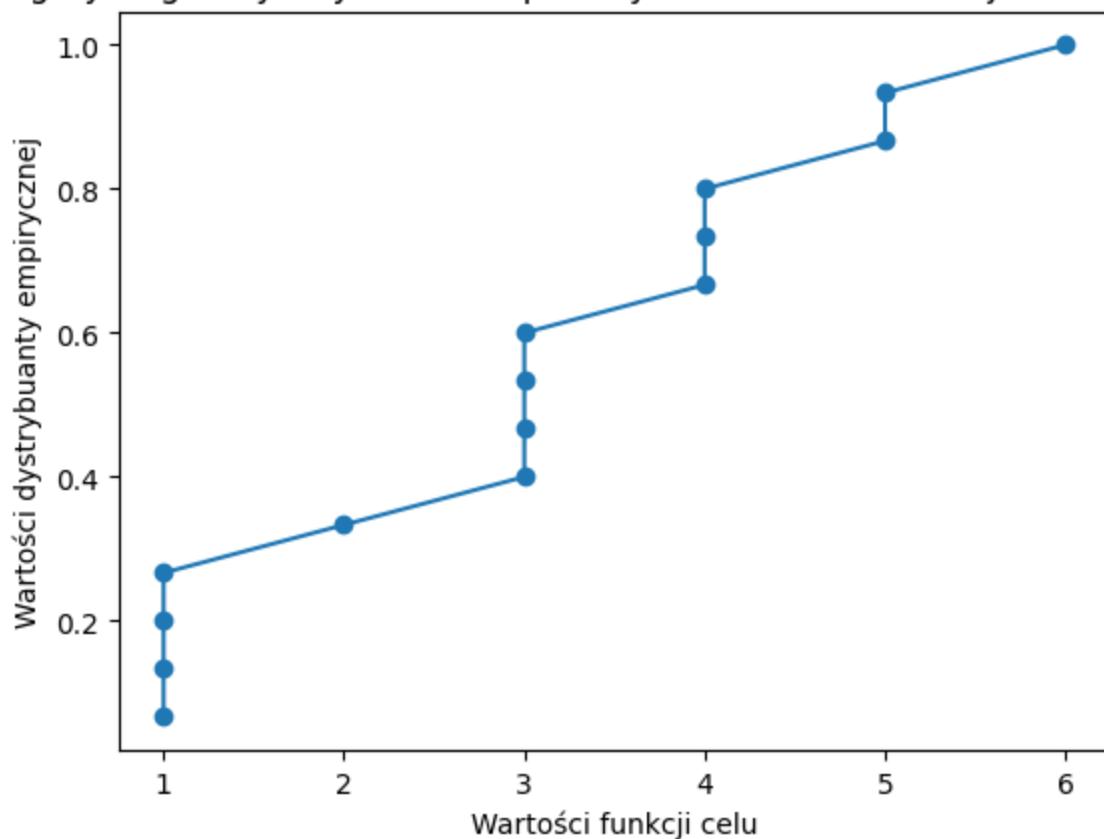
Algorytm genetyczny ECDF dla planszy hard liczba ewaluacji f. celu 10000



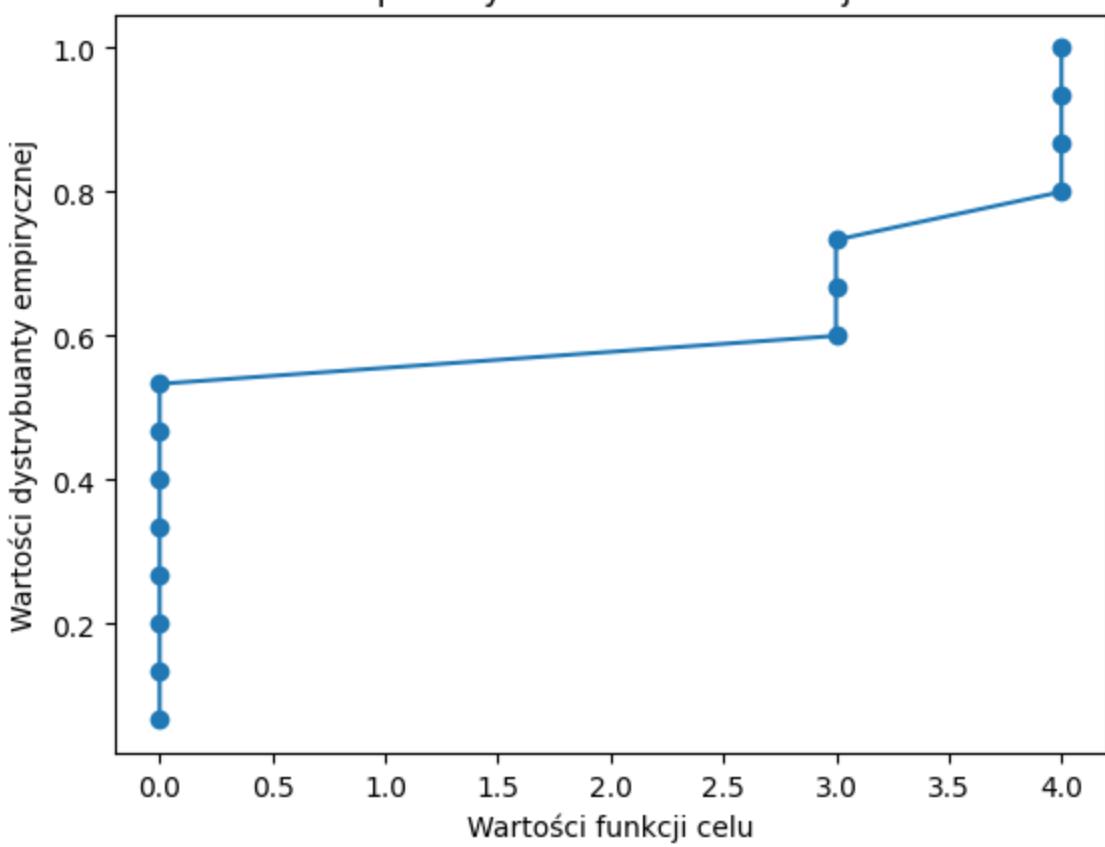
ACO ECDF dla planszy hard liczba ewaluacji f. celu 10000



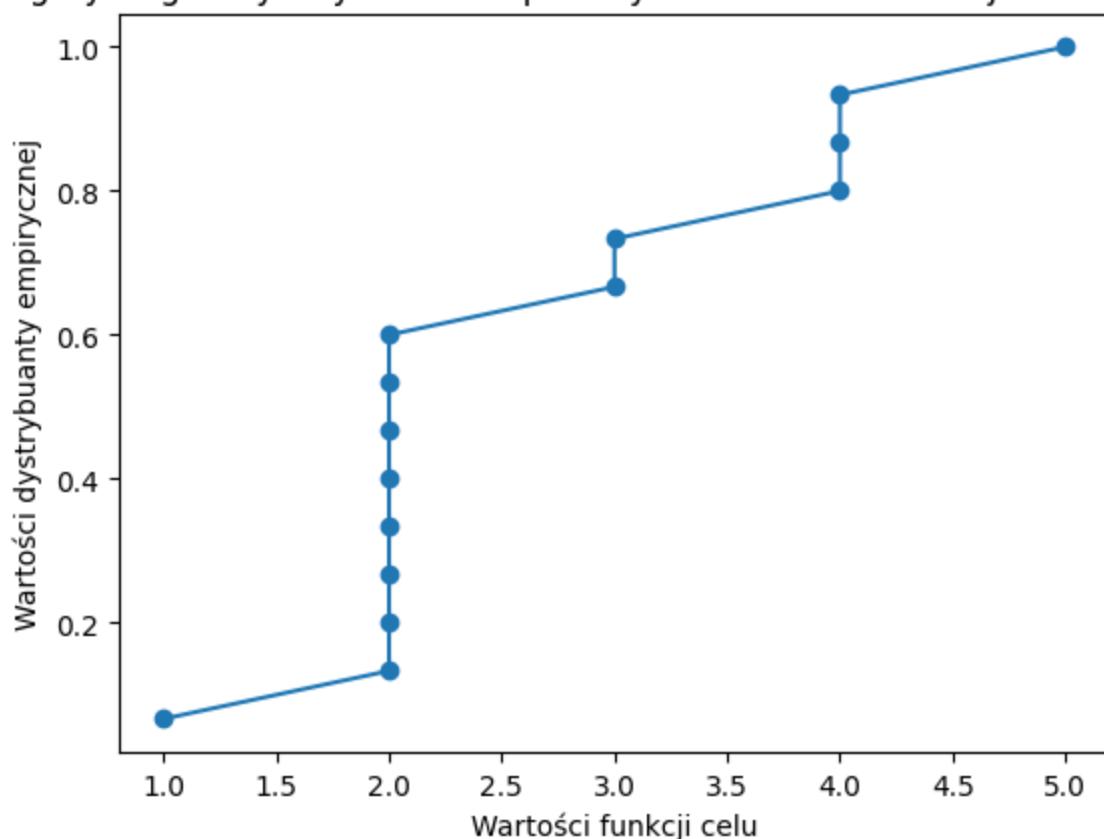
Algorytm genetyczny ECDF dla planszy hard liczba ewaluacji f. celu 15000



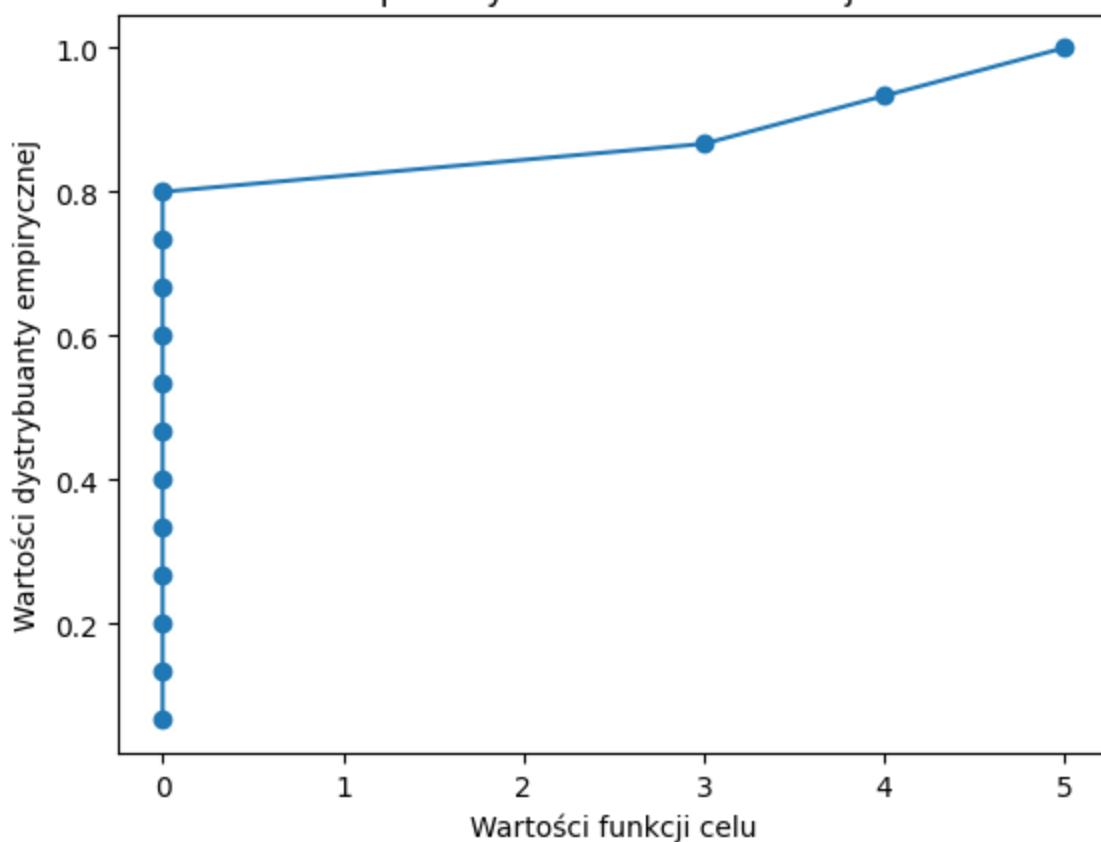
ACO ECDF dla planszy hard liczba ewaluacji f. celu 15000



Algorytm genetyczny ECDF dla planszy hard liczba ewaluacji f. celu 20000



ACO ECDF dla planszy hard liczba ewaluacji f. celu 20000



Wyniki dla algorytmu genetycznego

Znaleziono najlepsze rozwiązania	Plansza	Liczba ewaluacji f. celu
8	easy	5000
10	easy	10000

14	easy	15000
11	easy	20000
1	medium	5000
3	medium	10000
6	medium	15000
6	medium	20000
0	hard	5000
0	hard	10000
0	hard	15000
0	hard	20000

Średnia znalezionych rozwiązań: 4.916666666666667

Odchylenie standardowe znalezionych rozwiązań: 4.768967975941498

Średnia znalezionych rozwiązań dla prostej planszy: 10.75

Odchylenie standardowe znalezionych rozwiązań dla prostej planszy: 2.165063509461097

Średnia znalezionych rozwiązań dla średniej planszy: 4.0

Odchylenie standardowe znalezionych rozwiązań dla średniej: 2.1213203435596424

Średnia znalezionych rozwiązań dla trudnej planszy: 0.0

Odchylenie standardowe znalezionych rozwiązań dla trudnej planszy: 0.0

Wyniki dla algorytmu ACO

Znaleziono najlepsze rozwiązania	Plansza	Liczba ewaluacji f. celu
15	easy	5000
15	easy	10000
14	easy	15000
14	easy	20000
15	medium	5000
15	medium	10000
15	medium	15000
15	medium	20000
6	hard	5000
6	hard	10000
8	hard	15000
12	hard	20000

Średnia znalezionych rozwiązań: 12.5

Odchylenie standardowe znalezionych rozwiązań: 3.5

Średnia znalezionych rozwiązań dla prostej planszy: 14.5

Odchylenie standardowe znalezionych rozwiązań dla prostej planszy: 0.5

Średnia znalezionych rozwiązań dla średniej planszy: 15.0

Odchylenie standardowe znalezionych rozwiązań dla średniej: 0.0

Średnia znalezionych rozwiązań dla trudnej planszy: 8.0
Odchylenie standardowe znalezionych rozwiązań dla trudnej planszy: 2.449489742783178

Z badań ECDF dla maksymalnych iteracji wynika, że dla plansz easy i hard algorytm genetyczny jest lepszy (szybciej znajduje rozwiązanie itd). Jednakże nie jest to do końca prawda. Ze względu na wybrane hiperparametry liczba iteracji to dla algorytmu ACO mniej więcej liczba ewaluacji, a jednej mrówce trudno jakoś szybko znaleźć rozwiązanie. Algorytm genetyczny zaś miał do wykorzystania dużo większą liczbę ewaluacji, co się wiąże z większym kosztem obliczeniowym oraz czasowym. Dokładniejsze wyniki daje nam badanie wykresów ECDF (i innych statystyk) z ograniczeniem liczb ewaluacji. Dla każdej planszy i maksymalnej liczby ewaluacji algorytm ACO częściej i szybciej znajdował rozwiązania od algorytmu genetycznego, miał duży procent sukcesu.

Wnioski

Zdecydowanie możemy powiedzieć, że algorytm ACO jest sporo lepszy od algorytmu genetycznego, co wynika ze specyfiki problemu. Sudoku raczej rozwiązuje się "na logikę" wypełniając odpowiednie komórki możliwymi liczbami i wykreślając nieodpowiednie po pewnym czasie, powoli tworząc rozwiązanie, co próbuje symulować algorytm ACO. Dużo też pomaga możliwość propagacji "założen" po każdym ustawnieniu wartości. Zaś losowe wypełnianie planszy i potem wymienianie liczb miejscami, jak w algorytmie genetycznym, po prostu nie jest dobrym pomysłem, w rzeczywistości się też nie sprawdza.

Bibliografia

- Genetyczny: wykłady WSI i POP
- ACO: https://pl.wikipedia.org/wiki/Algorytm_mr%C3%B3wkowy, <https://arxiv.org/pdf/1805.03545.pdf>, <https://www.ii.pwr.edu.pl/~kwasnicka/lindaabrichwww/description.html>