Algorytm Genetyczny – Sztuczna inteligencja i inżynieria wiedzy

Monika Galińska

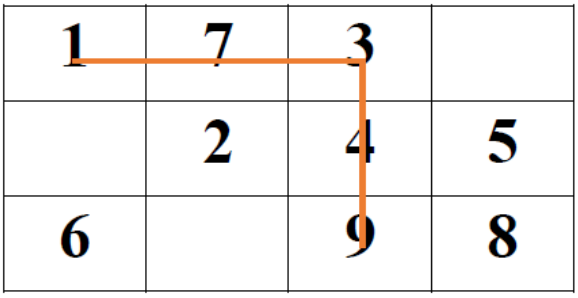
Wstęp

Algorytmy genetyczne często wykorzystywane są do rozwiązywania problemów projektowania. W naszym zadaniu zajmiemy się problemem Facility Layout Optimalization. By lepiej zwizualizować sobie przełożenie FLO na algorytm genetyczny należy dokonać przypisania odpowiednich dla nich pojęć. Jednymi z najważniejszych dla rozwiązania problemu pojęć są: genotyp, fenotyp i osobnik. Kluczowym dla poprawnego działania algorytmu jest poprawne zdefiniowanie osobnika, który w naszym przypadku będzie pojedynczą siatką rozstawienia wszystkich maszyn. W kontekście problemu FLO możemy przyjąć, że fenotyp to wizualne przedstawienie naszej siatki z rozstawionymi maszynami. Genotyp to odpowiednio programistyczne zakodowanie takiej siatki. Populację, na której będziemy działać, określimy jako zbiór siatek z rozstawieniem maszyn.

Problem

Jednym z typowych zastosowań metaheurystyk takich jak algorytmy genetyczne są problemy projektowania. W zadaniach projektowania musimy ustalić układ przestrzenny obiektów w taki sposób, aby zminimalizować pewien koszt związany z projektem i zmaksymalizować jego użyteczność. Projektuje się w ten sposób na przykład płytki PCB, anteny, czy rozkłady przestrzenne budynków.

Zajmiemy się tutaj konkretnym problemem – Facility Layout Optimization. Chcemy w ograniczonej przestrzeni rozplanować rozłożenie np. maszyn w fabryce, biorąc pod uwagę koszt związany z odległością między każdą parą maszyn, wynikający z przenoszenia materiałów między nimi. Najprostsze, dyskretne sformułowanie problemu wygląda następująco:



Rysunek 1. FLO – siatka 4 na 3, 9 maszyn do ustawienia, oznaczone połączenie 1 do 9

Mając siatkę m na n możliwych pozycji, chcemy ustawić pewną liczbę maszyn – czyli przypisać każdemu indeksowi i ze zbioru liczb całkowitych od 1 do zadanego k koordynaty (xi, yi). (Przy czym k ≤ mn.)

Dla każdej pary maszyn (i,j) są zdefiniowane: przepływ materiału między nimi oraz koszt obsługi tego przepływu. Celem przy wyborze ustawienia jest minimalizacja kosztu.

Ładowanie danych

Dostępne do zadania dane przechowywane są w formacie json. Do ich obsługi zaimplementowano dwie metody, oddzielnie dla kosztów i przepływu. Został także stworzony dodatkowy plik przechowujący dane o możliwych instancjach. Instancja składa się z nazwy, długości (dimX) i szerokości (dimY), liczby maszyn do rozstawienia oraz ścieżek do plików z danymi o kosztach i przepływach.

Screenshot 1 instances.json

Dane załadowane z plików json zwracane są w postaci listy odpowiednio kosztów, przepływów i instancji.

```python
def get_costs_list(instance):
    costs_list = []
    data = load_data(instance.cost_path)
    for entry in data:
        costs_list.append(Cost(entry['source'], entry['dest'], entry['cost']))
    return costs_list
```

Screenshot 2 Ładowanie danych kosztów

Metoda losowa

Przedstawiona poniżej funkcja umożliwia stworzenie jednego osobnika (Screenshot 3). W naszym przypadku jest to zbiór wszystkich maszyn z przypisanymi do nich współrzędnymi rozmieszczenia na siatce.

```python
def create_random_specimen(instance):
    machines = generate_clean_machines(instance)
    grid = generate_clean_grid(instance)
    for i in range(instance.machinesNumb):
        random_field = random.choice(grid)
        machines[i].x = random_field[0]
        machines[i].y = random_field[1]
        grid.remove(random_field)
    return machines
```

Screenshot 3 Metoda generująca osobnika

Maszyna posiada 3 pola: numer identyfikujący oraz współrzędne x i y.

```python
class Machine:
    def __init__(self, number, x, y):
        self.number = number
        self.x = x
        self.y = y
```

Screenshot 4 Model maszyny

Początkowo generowany jest zbiór maszyn, którego wielkość zależy od instancji, bez przypisanych współrzędnych.

```python
def generate_clean_machines(instance):
    machines = []
    for i in range(instance.machinesNumb):
        machines.append(Machine(i, -1, -1))
    return machines
```

Screenshot 5 Generowanie czystego zestawu maszyn

Tworzony jest także zbiór współrzędnych, w których potencjalnie może znaleźć się każda z maszyn.

```python
def generate_clean_grid(instance):
    fields = []
    for x in range(instance.dimX):
        for y in range(instance.dimY):
            fields.append((x, y))
    return fields
```

Screenshot 6 Generowanie możliwych dla danej instancji współrzędnych

Mając przygotowane metody pomocnicze można przejść do generowania populacji za pomocą metody losowej. Do każdej z maszyn przypisywany jest zestaw współrzędnych wylosowany ze zbioru wciąż dostępnych współrzędnych. Po przypisaniu do maszyny, użyte współrzędne usuwane są z listy dostępnych.

Funkcja przystosowania

Funkcja przystosowania jest reprezentacją przedstawionego poniżej wzoru.

$$\sum_{i,j} F_{ij} C_{ij} D_{ij}$$

Gdzie:

F_{ij} – jest to przepływ materiału między i a j

C_{ij} – jest to koszt obsługi materiałów między i a j

D_{ij} – odległość pomiędzy daną parą w ocenianym ustawieniu.

Przy czym F i C są zadane z góry, natomiast odległość D wynika z ocenianego przez nas ustawienia. W dyskretnej wersji problemu koordynaty (x_i, y_i) będą wyrażone liczbami całkowitymi i muszą się mieścić w zadanej siatce m na n. Korzystamy z odległości Manhattan – sumy bezwzględnych różnic w koordynatach x i y:

$$D_{ij} = |x_i - x_j| + |y_i - y_j|$$

Screenshot 7 Wzór reprezentujący liczenie kosztu dla jednego osobnika

W danych do zadania zamieszczono pliki z danymi określającymi przepływ materiałów i koszt ich obsługi dla niektórych połączeń pomiędzy maszynami. Implementacja funkcji przystosowania dla każdego połączenia najpierw liczy odległość. Następnie mnożone są koszt, przepływ i odległość. Wynik tego działania dodawany jest do wyniku.

```python
def check_quality(instance, specimen):
    costs = get_costs_list(instance)
    flows = get_flows_list(instance)
    costs_sum = 0
    for i in range(len(costs)):
        source = get_machine(specimen, costs[i].source)
        dest = get_machine(specimen, costs[i].dest)
        distance = abs(source.x - dest.x) + abs(source.y - dest.y)
        product = costs[i].cost * flows[i].amount * distance
        costs_sum += product
    return costs_sum
```

Screenshot 8 Funkcja przystosowania

Operator selekcji – turniej

Operator selekcji turniejowej przyjmuje dwa parametry, populację i rozmiar turnieju. Następnie losowany jest turniej wielkości wcześniej przyjętego rozmiaru. Funkcja zwraca osobnika z najmniejszą wartością funkcji przystosowania znalezioną w turnieju.

```
def tournament_selection(population, tournament_size):
    tournament = random.choices(population, k=tournament_size)
    best_specimen = population[0]
    for spec in tournament:
        if spec.fitness < best_specimen.fitness:
            best_specimen = spec
    return best_specimen
```

Screenshot 9 Operator selekcji - turniej

Operator selekcji – ruletka

Operator selekcji ruletkowej przyjmuje obecną populację i na jej podstawie przyporządkowuje każdemu osobnikowi populacji prawdopodobieństwo wylosowania. Ze względu na to, że lepsze osobniki mają niższe wartości przystosowania, funkcja odwraca stosunek prawdopodobieństwa, by faworyzować lepsze dla nas osobniki.

```
def roulette_wheel_selection(population):
    population_fitness = sum([specimen.fitness for specimen in population])
    probabilities = [specimen.fitness / population_fitness for specimen in population]
    inverse_probabilities = [(1 / x) for x in probabilities]
    inverse_probabilities = [x / sum(inverse_probabilities) for x in inverse_probabilities]
    chosen_index = np.random.choice(len(population), p=inverse_probabilities)
    return population[chosen_index]
```

Screenshot 10 Operator selekcji - ruletka

Operator krzyżowania – PMX

Dla problemu Facility Layout Optimalization wybrano krzyżowanie z częściowym odwzorowaniem (PMX). Zaimplementowany operator jest także operatorem krzyżowania jednopunktowego. Na początek losowany jest punkt przecięcia rodziców. Następnie Początek pierwszego rodzica do punktu przecięcia przypisywany jest do dziecka. Następnie dziecku przypisywane są możliwe do przepisania geny drugiego rodzica. Reszta maszyn rozmieszczana jest w pozostałych wolnych miejscach. Jeśli krzyżowanie nie zachodzi zwracany jest jeden z rodziców.

```python
def breed(instance, parent1, parent2, breed_rate):
    if random.random() < breed_rate:
        child_p1, child_p2, used_machines = [], [], []
        grid = generate_clean_grid(instance)
        cut_point = int(random.random() * len(parent1.machines))

        for i in range(2, cut_point):
            child_p1.append(parent1.machines[i])
            used_machines.append(parent1.machines[i].number)
            grid.remove((parent1.machines[i].x, parent1.machines[i].y))

        for m in parent2.machines:
            if m.number not in used_machines:
                if (m.x, m.y) in grid:
                    child_p2.append(m)
                    grid.remove((m.x, m.y))
                    used_machines.append(m.number)

        for m in parent2.machines:
            if m.number not in used_machines:
                place = int(random.random() * len(grid))
                child_p2.append(Machine(m.number, grid[place][0], grid[place][1]))
                grid.remove(grid[place])

        child = child_p1 + child_p2
        return Specimen(child, check_fitness(instance, child))
    else:
        return parent1
```

Screenshot 11 Operator krzyżowania - PMX

Operator mutacji

Mutacja polega na przypisaniu trzem maszynom nowej pozycji. Jeśli mutacja nie zachodzi, zwracany jest niezmieniony osobnik.

```python
def mutate(instance, specimen, mutation_rate):
    if random.random() < mutation_rate:
        child, used_machines = [], []
        grid = generate_clean_grid(instance)

        for i in range(len(specimen.machines) - 3):
            child.append(specimen.machines[i])
            used_machines.append(specimen.machines[i].number)
            grid.remove((specimen.machines[i].x, specimen.machines[i].y))

        for m in specimen.machines:
            if m.number not in used_machines:
                place = int(random.random() * len(grid))
                child.append(Machine(m.number, grid[place][0], grid[place][1]))
                grid.remove(grid[place])

        return Specimen(child, check_fitness(instance, child))
    return specimen
```

Screenshot 12 Operator mutacji

Porównanie wyników - TURNIEJ

instancja	Alg. Ewolucyjny [10x]				Metoda losowa N			
	best	worst	avg	std	best	worst	avg	std
EASY	4 822.4	7 919.8	4 928.31	52.22	5 067.8	11 302.8	7830.01	31.31
FLAT	11 209.5	19 347.0	11 452.84	250.58	14 174.0	31 864.0	24 330.6	66.04
HARD	15 927.3	24 787.7	16 356.94	1 494.02	26 809.2	50 916.0	38 559.66	121.29

Rezultaty dla metody losowej to uśrednione wyniki 10-krotnego przebiegu metody losowej dla każdej z instancji.

```
Instance - EASY
Random population
std: 31.30828062297044
best: 5067.8
avg: 7830.0007
worst: 11302.8
```

```
Instance - FLAT
Random population
std: 66.03900897701766
best: 14174.0
avg: 24330.596999999998
worst: 31864.0
```

```
Instance - HARD
Random population
std: 121.29145653829615
best: 26663.72
avg: 38572.612519999995
worst: 50916.43
```

Ustawienia parametrów:

Instancja – EASY

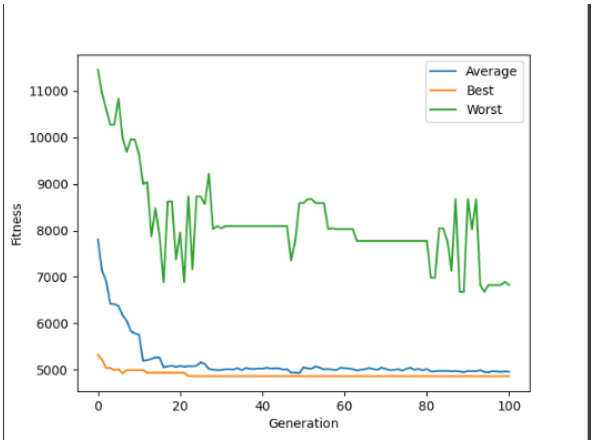
Prawdopodobieństwo krzyżowania – 0.8

Prawdopodobieństwo mutacji – 0.1

Liczba pokoleń – 100

Rozmiar turnieju – 3

Wielkość populacji – 1 000



Ustawienia parametrów:

Instancja – FLAT

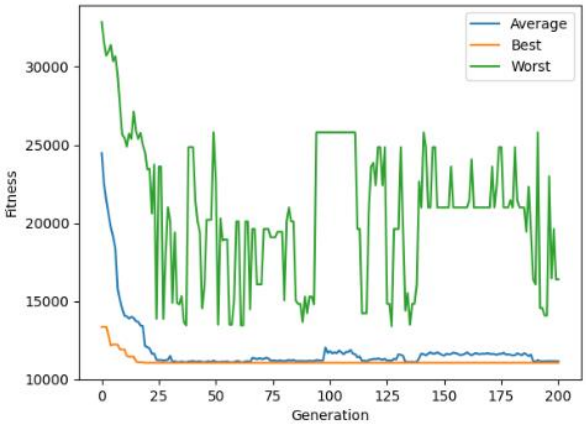
Prawdopodobieństwo krzyżowania – 0.8

Prawdopodobieństwo mutacji – 0.1

Liczba pokoleń – 200

Rozmiar turnieju – 3

Wielkość populacji – 1 000



Ustawienia parametrów:

Instancja – HARD

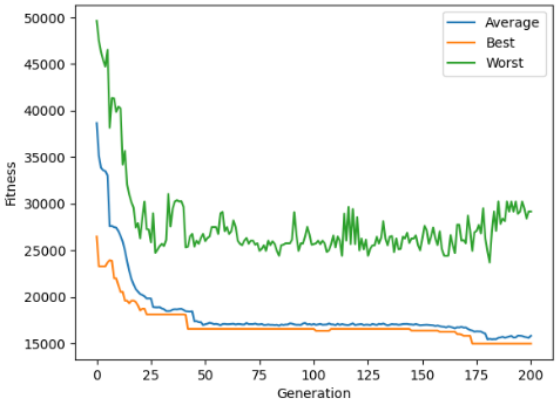
Prawdopodobieństwo krzyżowania – 0.8

Prawdopodobieństwo mutacji – 0.1

Liczba pokoleń – 200

Rozmiar turnieju – 3

Wielkość populacji – 1 000



Najlepsze wyniki – selekcja turniejowa

Najlepszy uzyskany wynik dla EASY selekcja turniejowa:

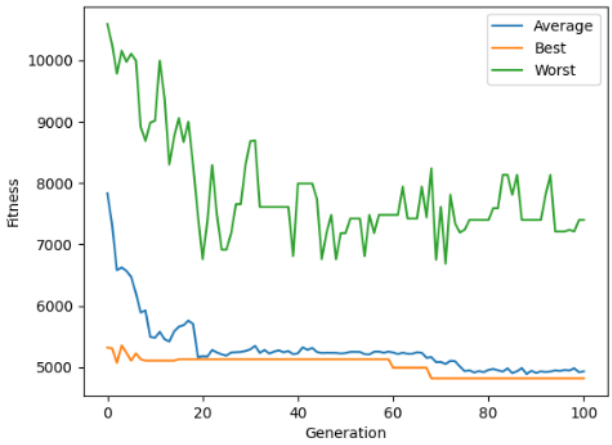
Ustawienia parametrów:

- Instancja – EASY
- Prawdopodobieństwo krzyżowania – 0.8
- Prawdopodobieństwo mutacji – 0.1
- Liczba pokoleń – 100
- Rozmiar turnieju – 3
- Wielkość populacji – 500

Best: 4 818

Average: 4 969.92

Worst: 8 818



Najlepszy uzyskany wynik dla FLAT selekcja turniejowa:

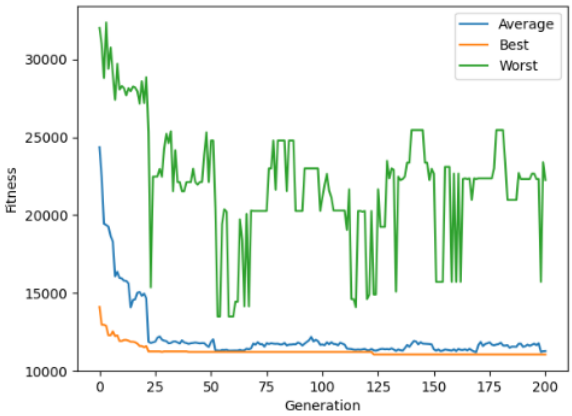
Ustawienia parametrów:

- Instancja – FLAT
- Prawdopodobieństwo krzyżowania – 0.8
- Prawdopodobieństwo mutacji – 0.1
- Liczba pokoleń – 100
- Rozmiar turnieju – 3
- Wielkość populacji – 1 000

Best: 11 055

Average: 11 278.06

Worst: 22 255



Najlepszy uzyskany wynik dla HARD selekcja turniejowa:

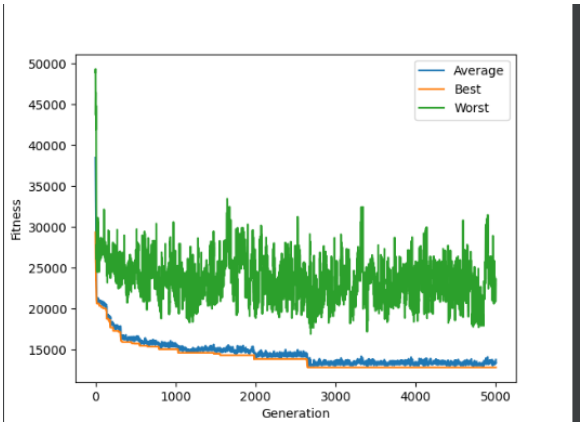
Ustawienia parametrów:

- Instancja – HARD
- Prawdopodobieństwo krzyżowania – 0.8
- Prawdopodobieństwo mutacji – 0.1
- Liczba pokoleń – 5 000
- Rozmiar turnieju – 3
- Wielkość populacji – 1 000

Best: 12 822

Average: 13 777.95

Worst: 23 597



Badanie zachowania algorytmu przy zmianie wielkości populacji

Ustawienia parametrów:

Instancja – HARD

Prawdopodobieństwo krzyżowania – 0.8

Prawdopodobieństwo mutacji – 0.1

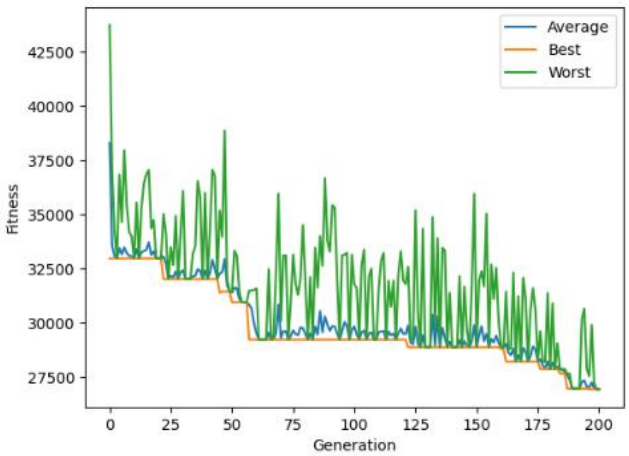
Liczba pokoleń – 200

Rozmiar turnieju – 3

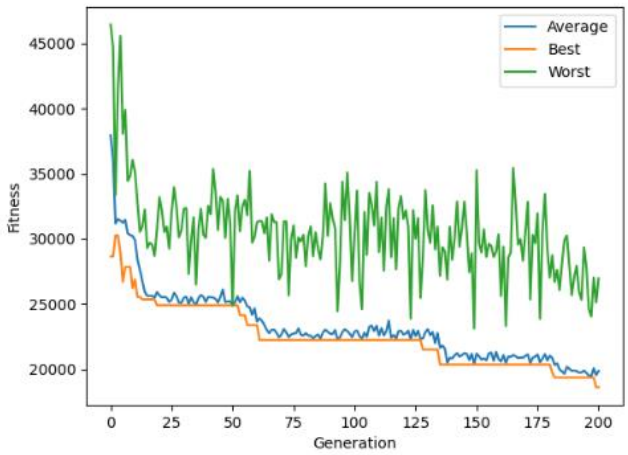
Wielkość populacji – 10 / 50 / 100 / 200 / 500 / 1 000

Wraz z wzrostem wielkości populacji możemy zaobserwować poprawę działania algorytmu, który otrzymuje wyniki lepsze średnio o ok. 4 000 porównując populacje 100 i 1 000 osobników. Przy populacji wielkości 1 000 możemy zaobserwować dla instancji hard wyniki w granicach 14 000, podczas gdy przy mniejszych populacjach zatrzymują się one ok. 18 000. Dla 10 osobników w populacji otrzymujemy średnio 24 000 co leży bardzo daleko od optymalnego wyniku.

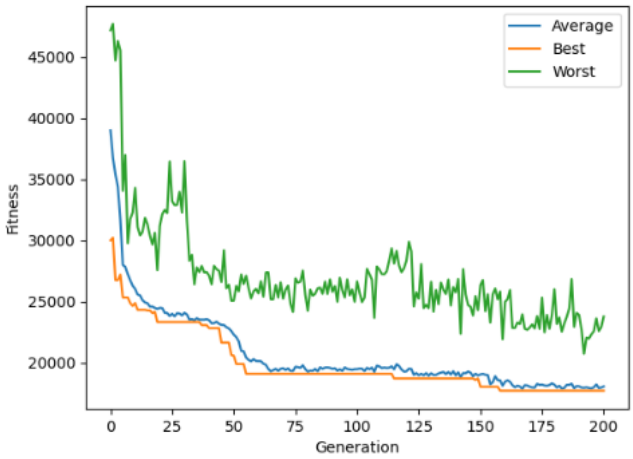
10 osobników - średnio 24 000



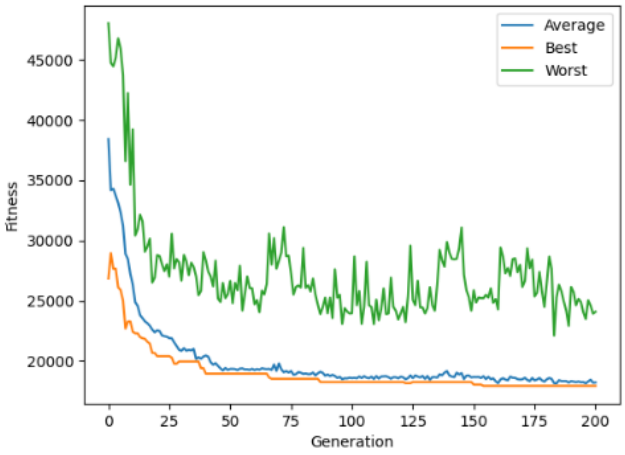
50 osobników – średnio 21 000



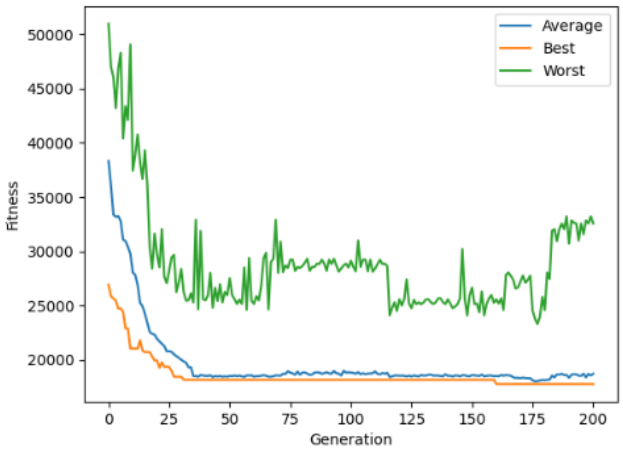
100 osobników turniej – 19 000



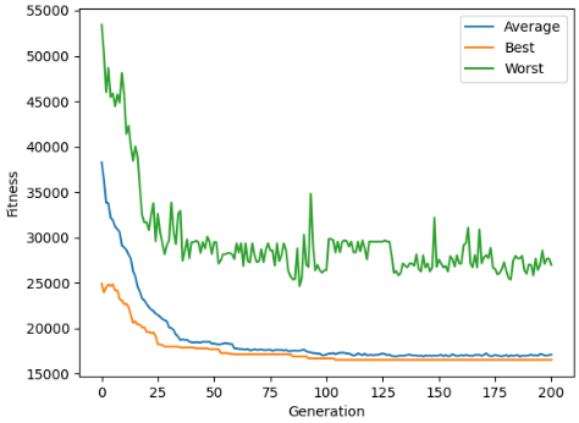
200 osobników turniej – 18 000



500 osobników – średnio 17 000



1000 osobników – średnio 15 000



Badanie zachowania algorytmu przy zmianie wielkości turnieju

Ustawienia parametrów:

Instancja – HARD

Prawdopodobieństwo krzyżowania – 0.8

Prawdopodobieństwo mutacji – 0.1

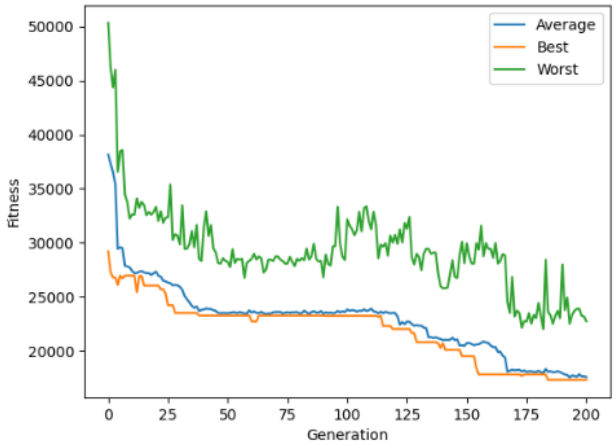
Liczba pokoleń – 200

Rozmiar turnieju – 2 / 5 / 10 / 50 / 100

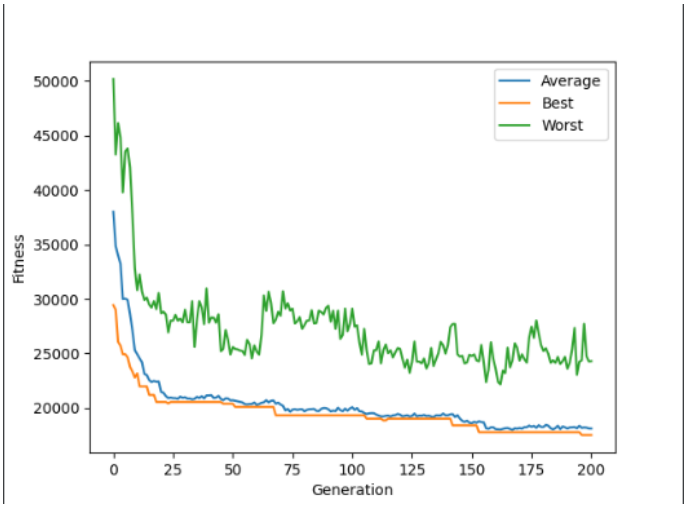
Wielkość populacji – 200

Zmiana wielkości turnieju nie ma znaczącego wpływu na osiągany wynik końcowy. Znacząco wpływa jednak na kształt wykresu, który przy większych wartościach staje się coraz bardziej schodkowy.

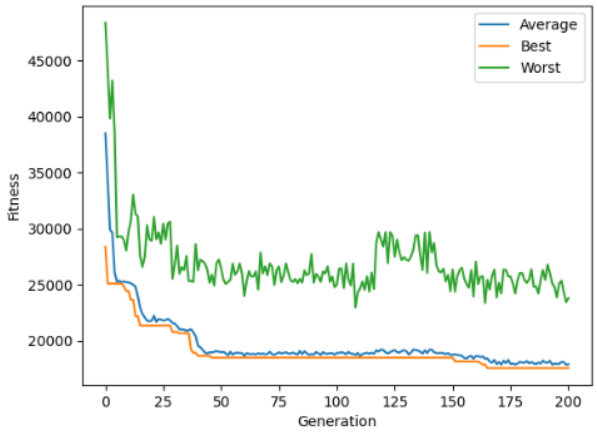
Wielkość turnieju 2 – średnio ok. 18 000



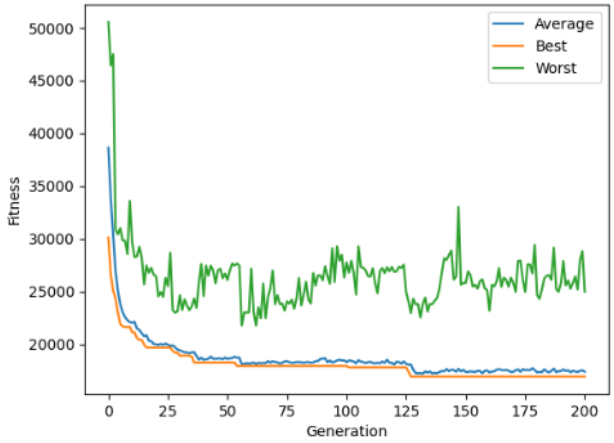
Wielkość turnieju 5 – średnio ok. 18 000



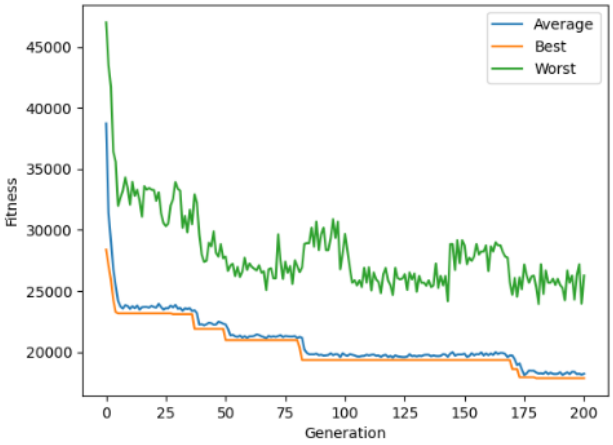
Wielkość turnieju 10 – średnio ok. 18 000



Wielkość turnieju 50 – średnio ok. 18 000



Wielkość turnieju 100 – średnio ok. 18 000



Badanie zachowania algorytmu przy zmianie prawdopodobieństwa krzyżowania

Ustawienia parametrów:

Instancja – HARD

Prawdopodobieństwo krzyżowania – 0.1 / 0.5 / 0.9

Prawdopodobieństwo mutacji – 0.1

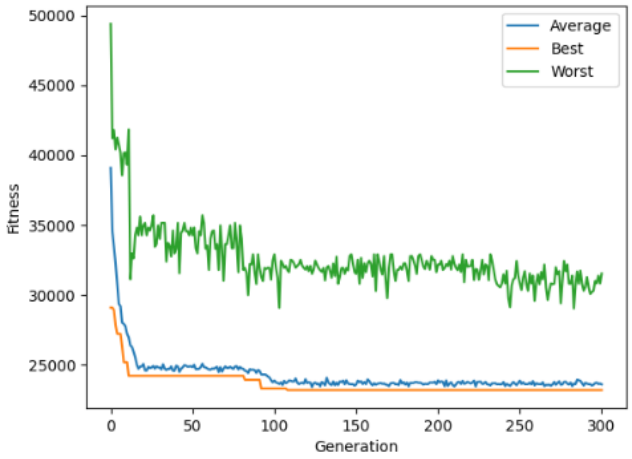
Liczba pokoleń – 300

Rozmiar turnieju – 3

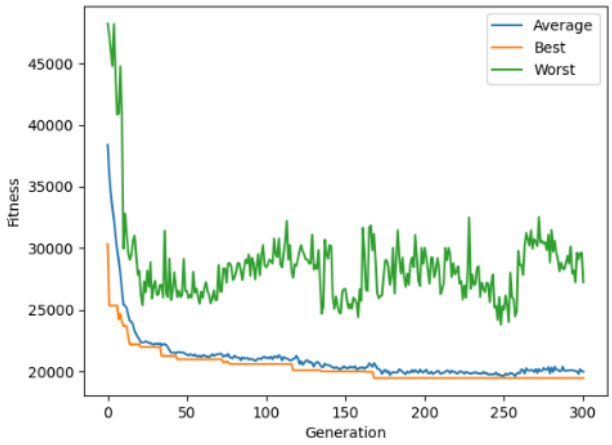
Wielkość populacji – 200

Zbyt niskie prawdopodobieństwo krzyżowania powoduje wczesną stagnację algorytmu i zatrzymanie postępu populacji przy bardzo nieoptymalnym wyniku. Już przy prawdopodobieństwie krzyżowania wynoszącym 50% możemy zaobserwować bliskie optymalnym dla danych parametrów wyniki dla selekcji turniejowej.

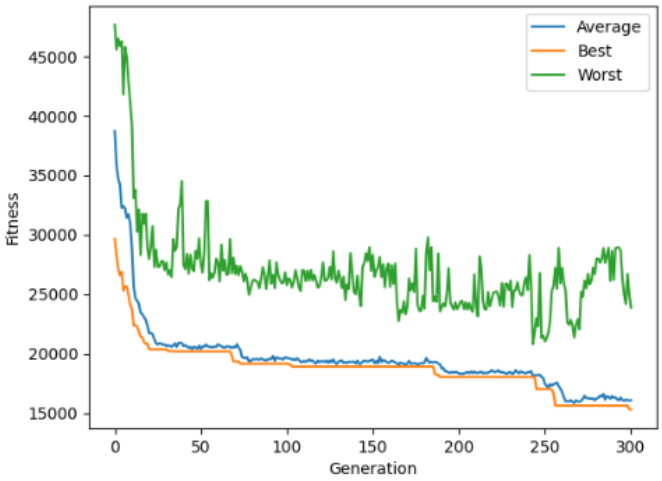
Prawdopodobieństwo krzyżowania 0.1 – średnio ok. 22 000



Prawdopodobieństwo krzyżowania 0.5 – średnio ok. 18 000



Prawdopodobieństwo krzyżowania 0.9 – średnio ok. 18 000



Badanie zachowania algorytmu przy zmianie prawdopodobieństwa mutacji

Ustawienia parametrów:

Instancja – HARD

Prawdopodobieństwo krzyżowania – 0.8

Prawdopodobieństwo mutacji – 0.1 / 0.5 / 0.9

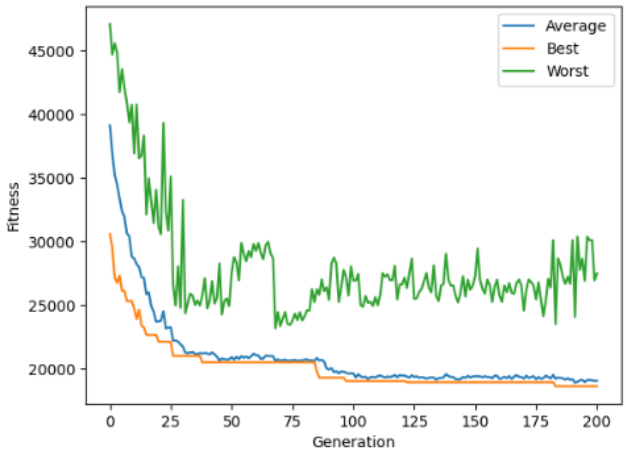
Liczba pokoleń – 200

Rozmiar turnieju – 3

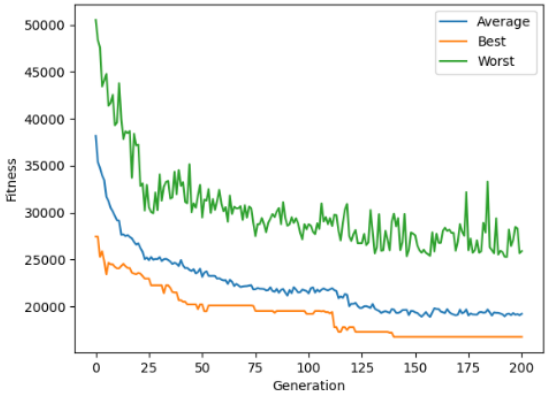
Wielkość populacji – 200

Zmiana prawdopodobieństwa mutacji ma duży wpływ na średnie i najgorsze wyniki osiągane przez algorytm. Jak możemy zaobserwować na wykresach niezależnie od prawdopodobieństwa mutacji najlepszy osiągany wynik jest praktycznie taki sam. Wraz ze zwiększaniem prawdopodobieństwa pojawia się natomiast coraz więcej słabych osobników, a średnia wyników rośnie. Dodatkowo warto zauważyć zmiany zachodzące w wykresie najlepszego wyniku, przy prawdopodobieństwie 90% zaczyna „drgać", podczas gdy przy mniejszych wartościach jest odcinkami linią prostą.

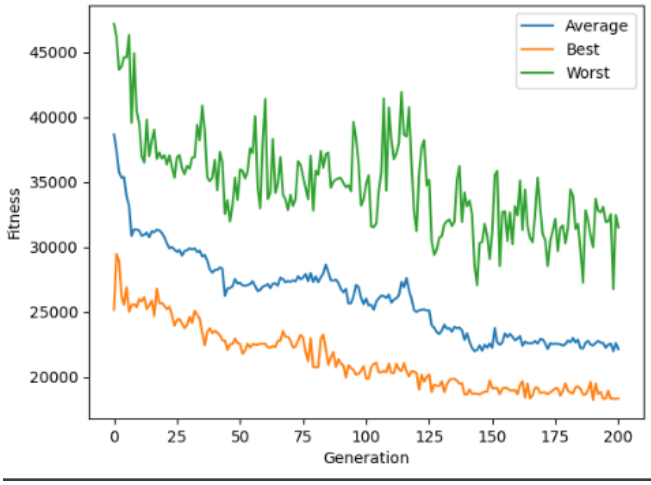
Prawdopodobieństwo mutacji 0.1 – średnio ok. 17 500



Prawdopodobieństwo mutacji 0.5 – średnio ok. 19 500



Prawdopodobieństwo mutacji 0.9 – średnio ok. 23 000



Porównanie wyników – RULETKA

instancja	Alg. Ewolucyjny [10x]				Metoda losowa N			
	best	worst	avg	std	best	worst	avg	std
EASY	4 855.3	8 502.7	5 053.06	47.6	5 067.8	11 302.8	7830.01	31.31
FLAT	11 567.5	22 677.0	12 208.41	395.9	14 174.0	31 864.0	24 330.6	66.04
HARD	18 730.1	29 044.2	20 098.08	1 568.67	26 809.2	50 916.0	38 559.66	121.29

Rezultaty dla metody losowej to uśrednione wyniki 10-krotnego przebiegu metody losowej dla każdej z instancji.

Ustawienia parametrów:

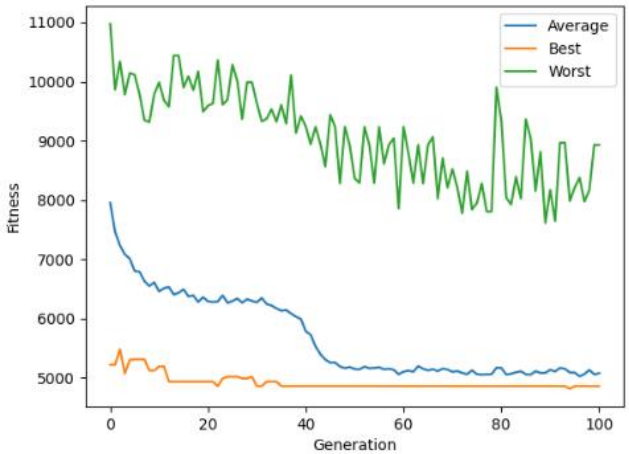
Instancja – EASY

Prawdopodobieństwo krzyżowania – 0.8

Prawdopodobieństwo mutacji – 0.1

Liczba pokoleń – 100

Wielkość populacji – 300



Ustawienia parametrów:

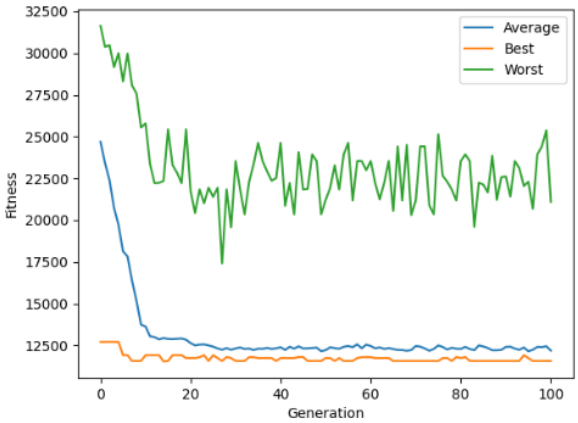
Instancja – FLAT

Prawdopodobieństwo krzyżowania – 0.8

Prawdopodobieństwo mutacji – 0.1

Liczba pokoleń – 100

Wielkość populacji – 300



Ustawienia parametrów:

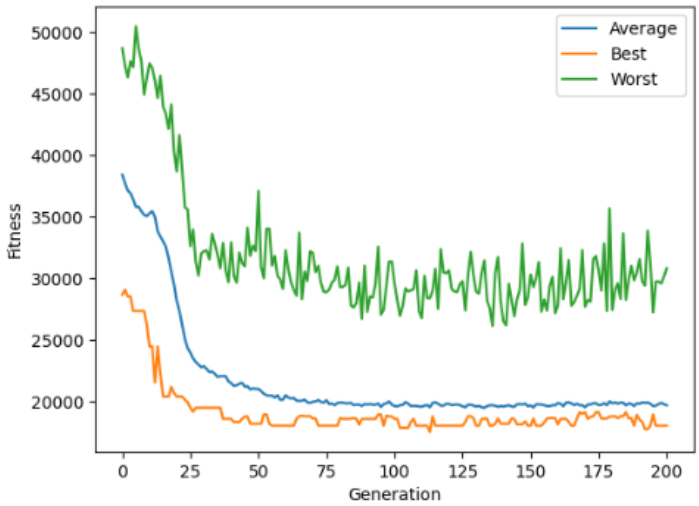
Instancja – HARD

Prawdopodobieństwo krzyżowania – 0.8

Prawdopodobieństwo mutacji – 0.1

Liczba pokoleń – 200

Wielkość populacji – 300



Najlepsze wyniki – selekcja ruletkowa

Najlepszy uzyskany wynik dla EASY selekcja ruletkowa:

Ustawienia parametrów:

Instancja – EASY

Prawdopodobieństwo krzyżowania – 0.8

Prawdopodobieństwo mutacji – 0.1

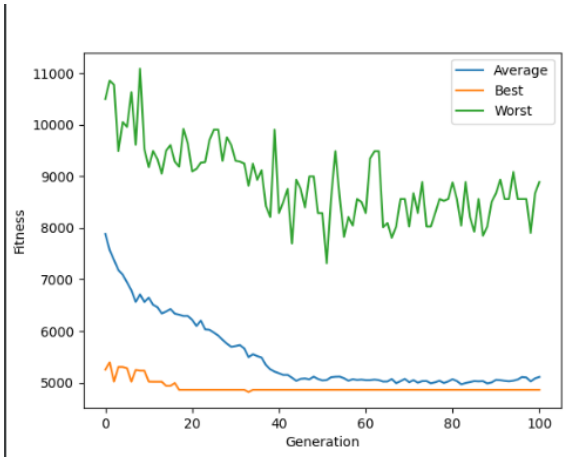
Liczba pokoleń – 100

Wielkość populacji – 300

Best: 4 818

Average: 5 046.28

Worst: 8 439



Najlepszy uzyskany wynik dla FLAT selekcja ruletkowa:

Ustawienia parametrów:

Instancja – FLAT

Prawdopodobieństwo krzyżowania – 0.8

Prawdopodobieństwo mutacji – 0.1

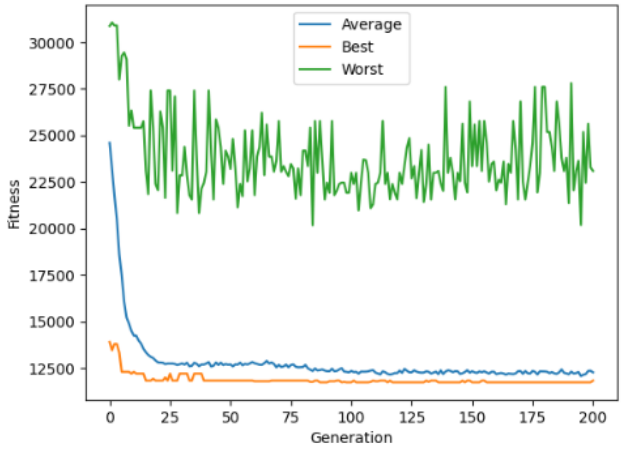
Liczba pokoleń – 100

Wielkość populacji – 300

Best: 11 825

Average: 12 266.96

Worst: 23 095



Najlepszy uzyskany wynik dla HARD selekcja ruletkowa:

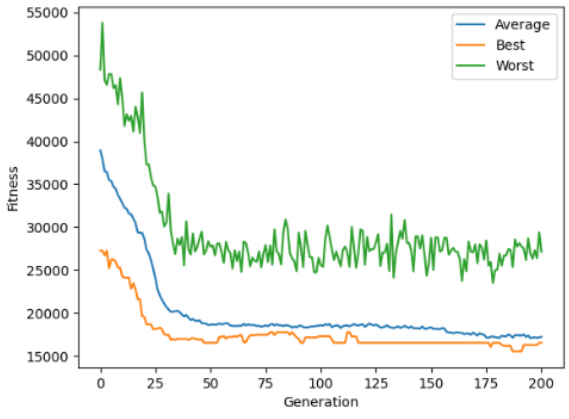
Ustawienia parametrów:

- Instancja – HARD
- Prawdopodobieństwo krzyżowania – 0.8
- Prawdopodobieństwo mutacji – 0.1
- Liczba pokoleń – 200
- Wielkość populacji – 300

Best: 16 545

Average: 17 224.71

Worst: 27 135



Badanie zachowania algorytmu przy zmianie wielkości populacji

Ustawienia parametrów:

Instancja – HARD

Prawdopodobieństwo krzyżowania – 0.8

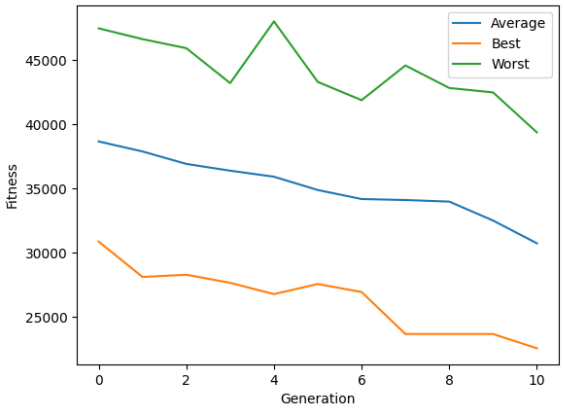
Prawdopodobieństwo mutacji – 0.1

Liczba pokoleń – 10 / 50 / 100 / 300

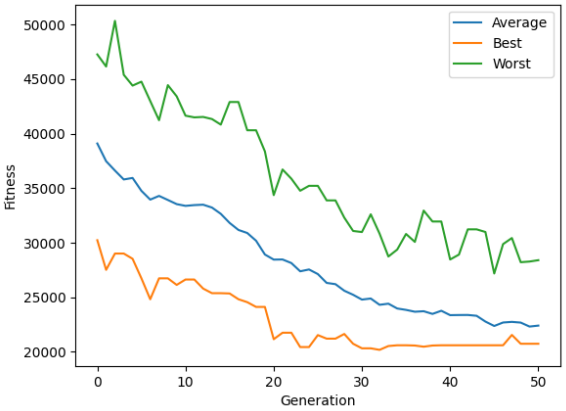
Wielkość populacji – 100

Przeprowadzone testy pokazują, że większa liczba pokoleń wpływa w większości przypadków na lepszy wynik. Nieoptymalna jest jednak zbyt duża liczba pokoleń, ponieważ w pewnym momencie wykres się wypłaszcza, a kolejne iteracje algorytmu przestają mieć znaczący wpływ na wynik. Moment wypłaszczenia wykresu zależy od przyjętych parametrów i typu selekcji, dla ruletki następuje on typowo później niż dla turnieju.

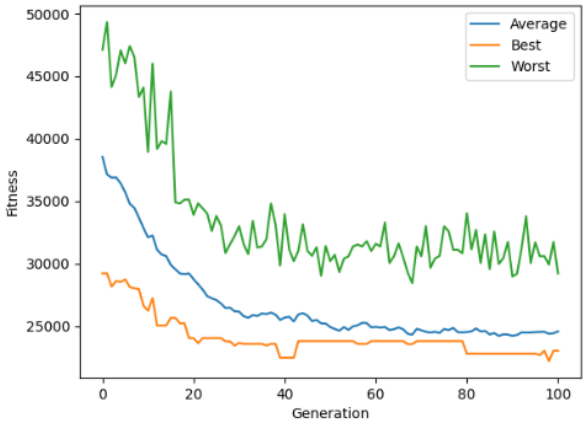
10 pokoleń - średnio 31 439.47



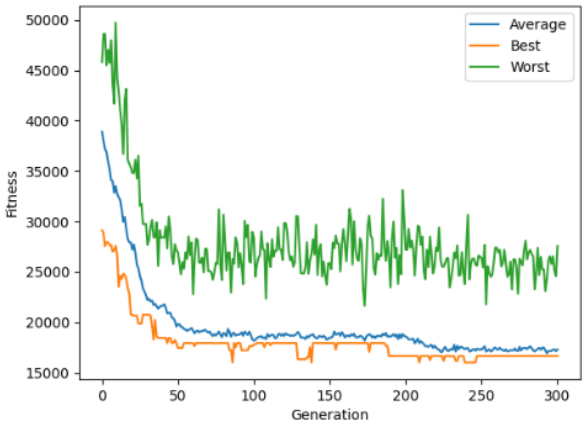
50 pokoleń - średnio 23 928.89



100 pokoleń - średnio 22 092.13



300 pokoleń - średnio 20 771.81



Porównanie selekcji turniejowej i ruletki

Selekcja turniejowa daje generalnie lepsze wyniki od ruletki. Jest mniej wrażliwa na zmiany parametrów i niezależnie od wielkości turnieju przynosi regularnie podobnie dobre wyniki. Ruletka daje mniej konsekwentne i stałe wyniki, pokazuje to odchylnie standardowe zanotowane w tabelach dla obu selekcji. Ponad to przy selekcji ruletkowej nawet małe zmiany parametrów potrafią rozstroić cały mechanizm i pogorszyć otrzymywane wyniki. Największy wpływ na poprawę działania selekcji ruletkowej miało zwiększenie ciśnienia selekcyjnego. Przy prawdopodobieństwie wylosowania osobnika definiowanym jako *1/x*, prawie niemożliwym było uzyskanie wyników rywalizujących z selekcją turniejową. Zwiększenie ciśnienia za pomocą zmiany wzoru na *1/(x^4)* znacznie poprawiło otrzymywane wyniki, ponieważ dobre jakościowo osobniki zaczęły być losowane wystarczająco często.

W przypadku obu selekcji pozytywny wpływ na poprawę wyniku ma zwiększanie wielkości populacji. Dodatkowo zarówno prawdopodobieństwo krzyżowania, jak i prawdopodobieństwo mutacji nie mogą być zbyt niskie, ponieważ powoduje to szybką stagnację algorytmu, który przestaje ewoluować.

Porównując wykresy otrzymywane przy wykorzystaniu obu selekcji możemy zauważyć znacznie gorsze wyniki średniego najgorszego osobnika. Dodatkowo ruletka sprawia, że wykres jest bardziej dynamiczny, rzadko pojawiają się równoległe do osi X linie dla najlepszego osobnika, które są zjawiskiem charakterystycznym dla selekcji turniejowej. Selekcja turniejowa ma także tendencję do bardzo szybkiego znajdowania optymalnych dla danej populacji osobników i nieodbiegania od nich.

Porównanie z metodami losowymi

Metoda losowa generuje wyniki bez porównania gorsze od algorytmu genetycznego. Największe pogorszenie widać w zakresie średniego wyniku. Także odchylenie standardowe jest większe dla metody losowej, co sugeruje dużą nieprzewidywalność otrzymywanych rezultatów.