Projekt z Projektowania Efektywnych Algorytmów

Numer etapu: 3

Zadanie wykonał: Michał Wróbel (259132)

Grupa laboratoryjna: K00-490

Termin zajęć: WT 17.05

Data oddania: 24.01.2023

Prowadzący: Dr inż. Jarosław Mierzwa

1. **Cel:**

   Celem projektu było zaimplementowanie algorytmu genetycznego dla asymetrycznego problemu komiwojażera, z możliwością wyboru metody krzyżowania oraz metody mutacji.

2. **Projekt:**

   a) **Wstęp teoretyczny**

   Algorytm genetyczny – rodzaj heurystyki przeszukującej przestrzeń alternatywnych rozwiązań problemu w celu wyszukania najlepszych rozwiązań.

   Sposób działania algorytmów genetycznych nieprzypadkowo przypomina zjawisko ewolucji biologicznej, ponieważ ich twórca John Henry Holland właśnie z biologii czerpał inspiracje do swoich prac.

   Startową populację generujemy w taki sposób, że dla każdej ścieżki losujemy pierwsze trzy miasta a resztę obliczamy algorytmem zachłannym.

   W algorytmie mamy ograniczoną liczbę osobników (w mojej implementacji liczba ta jest stała i wynosi tyle co rozmiar populacji). Selekcja odbywa się poprzez posegregowanie ścieżek od najlepszej do najgorszej (kryterium jest koszt ścieżki) i usunięcie nadmiarowych (czyli w tym wypadku najgorszych) ścieżek.

   W czasie działania algorytmu wybieramy dwie ścieżki (losujemy je z puli). Następnie jest losowane to, czy zajdzie pomiędzy ścieżkami krzyżowanie (jest to krzyżowanie względem jednego punktu), jeżeli nie to jako potomka uznajemy pierwszą ścieżkę.

```python
# Funkcja krzyżowania dwóch ścieżek z jednym punktem podziału.
@staticmethod
def one_point_crossover(first_path, second_path):

    # Wylosowanie punktu podziału.
    crossover_city = random.randint(1, len(first_path) - 1)

    # Skopiowanie do nowej ścieżki części pierwszej ścieżki (od jej początku, aż do punktu podziału).
    new_route = first_path[:crossover_city]

    # Dla każdego miasta z drugiej ścieżki wykonujemy następująca operację.
    for current_city in second_path:

        # Jeżeli aktualne miasto nie znajduje się w nowej ścieżce, to dodajemy je na jej koniec.
        if current_city not in new_route:
            new_route.append(current_city)

    # Zwrócenie nowej ścieżki.
    return new_route
```

Następnie losujemy to, czy względem potomka zajdzie mutacja.

```python
# Funkcja mutacji typu swap (zamieniamy 2 miasta ze sobą).
@staticmethod
def swap_mutation(path):

    # Losowanie dwóch miast z listy.
    first_city = random.randint(0, len(path) - 1)
    second_city = random.randint(0, len(path) - 1)

    # Zamiana miast miejscami.
    path[first_city], path[second_city] = path[second_city], path[first_city]

    # Zwrócenie nowej ścieżki.
    return path
```

b) Opis programu.

```python
# Algorytm genetyczny.
def genetic_algorithm(self, matrix, maximal_time, start_population_size, crossing_propagation, mutation_propagation):

    # Generowanie populacji początkowej.
    population = self.generate_population(matrix, start_population_size)

    # Wyznaczenie kryterium zakończenia algorytmu
    start_time = time.perf_counter_ns()
    end_time = start_time + int(maximal_time) * pow(10, 9)

    # Dopóki nie minie wyznaczony czas, wykonujemy następujące operacje.
    while time.perf_counter_ns() < end_time:

        # Posortowanie populacji wg. kosztu podróży dla danej ścieżki.
        population = sorted(population, key=lambda x: self.get_path_cost(matrix, x))

        # Metoda selekcji. Z populacji, którą aktualnie mamy zachowujemy połowę najlepszych ścieżek.
        population = population[:start_population_size // 2]

        # Wykonujemy operacje tyle razy, ile wynosi rozmiar początkowej populacji podzielony przez 2 z podłogą.
        for _ in range(start_population_size // 2):

            # Wybranie pierwszego i drugiego losowego rodzica z populacji
            first_parent = random.choice(population)
            second_parent = random.choice(population)
            child = first_parent

            # Wykonanie krzyżowania dwóch rodziców.
            crossing_random_number = random.randint(0, 100)
```

```python
            if crossing_random_number < crossing_propagation:
                child = self.one_point_crossover(first_parent, second_parent)

            # Wykonanie mutacji potomka.
            mutation_random_number = random.randint(0, 100)

            if mutation_random_number < mutation_propagation:
                child = self.swap_mutation(child)

            # Dodanie potomka do populacji
            population.append(child)

    # Wybranie najlepszej ścieżki, czyli pierwszej ścieżki, z populacji.
    best_path = population[0]
    # Obliczenie kosztu dla tej ścieżki.
    best_path_cost = self.get_path_cost(matrix, best_path)

    # Dodajemy na koniec ścieżki miasto startowe.
    best_path += (best_path[0],)

    # Zwrócenie kosztu, ścieżki oraz czasu wykonania algorytmu.
    return best_path_cost, best_path
```

c) Tabele i wykresy

Plik:	ftv47.atsp	Błąd
Wynik optymalny:	1776	
Prozmiar populacji	100	
Czas [s]	Wynik	
30	1997	12.44%
	2192	23.42%
	2184	22.97%
	2036	14.64%
	2146	20.83%
	średni błąd	18.86%
60	1983	11.66%
	2052	15.54%
	2124	19.59%
	2050	15.43%
	2037	14.70%
	średni błąd	15.38%
120	2141	20.55%
	2096	18.02%
	2059	15.93%
	2067	16.39%
	2155	21.34%
	średni błąd	18.45%
240	2032	14.41%
	2020	13.74%
	2078	17.00%
	2007	13.01%
	2039	14.81%
	średni błąd	14.59%

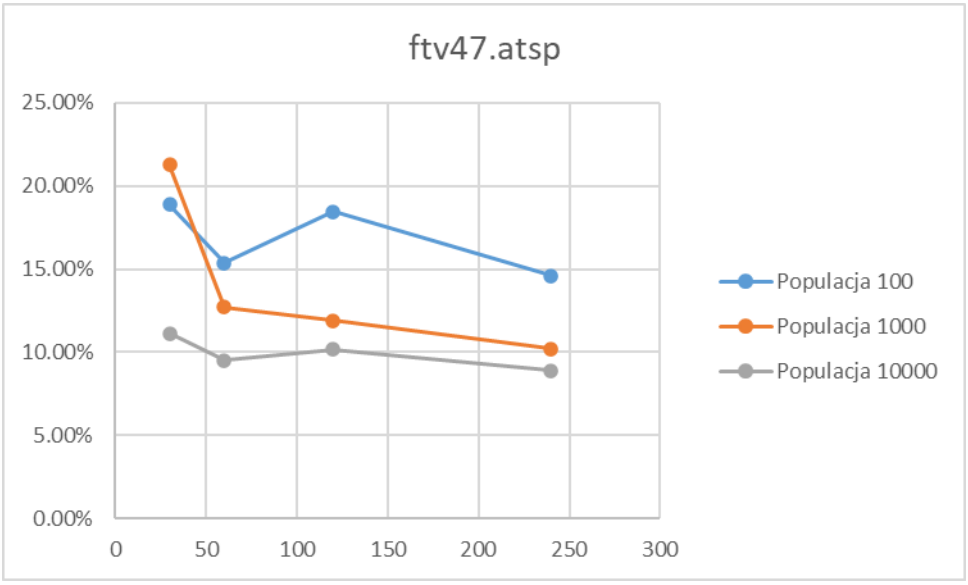
Czas [s]	Błąd[%]
30	18.86%
60	15.38%
120	18.45%
240	14.59%

Plik:	ftv47.atsp	Błąd
Wynik optymalny:	1776	
Prozmiar populacji	1000	
Czas [s]	Wynik	
30	1979	11.43%
	2010	13.18%
	2057	15.82%
	2657	49.61%
	2066	16.33%
	średni błąd	21.27%
60	1982	11.60%
	1968	10.81%
	2019	13.68%
	2022	13.85%
	2019	13.68%
	średni błąd	12.73%
120	2004	12.84%
	1997	12.44%
	2040	14.86%
	2019	13.68%
	1878	5.74%
	średni błąd	11.91%
240	1968	10.81%
	1932	8.78%
	1932	8.78%
	1954	10.02%
	1999	12.56%
	średni błąd	10.19%

Czas [s]	Błąd[%]
30	21.27%
60	12.73%
120	11.91%
240	10.19%

Plik:	ftv47.atsp	Błąd
Wynik optymalny:	1776	
Prozmiar populacji	10000	
Czas [s]	Wynik	
30	1983	11.66%
	1987	11.88%
	1942	9.35%
	2000	12.61%
	1957	10.19%
	średni błąd	11.14%
60	1997	12.44%
	1983	11.66%
	1916	7.88%
	1932	8.78%
	1897	6.81%
	średni błąd	9.52%
120	1902	7.09%
	2025	14.02%
	1982	11.60%
	1959	10.30%
	1916	7.88%
	średni błąd	10.18%
240	1895	6.70%
	1970	10.92%
	1916	7.88%
	1950	9.80%
	1938	9.12%
	średni błąd	8.89%

Czas [s]	Błąd[%]
30	11.14%
60	9.52%
120	10.18%
240	8.89%



Plik:	ftv170.atsp	Błąd
Wynik optymalny:	1776	
Prozmiar populacji	100	
Czas [s]	Wynik	
30	3701	108.39%
	3786	113.18%
	3844	116.44%
	3692	107.88%
	3904	119.82%
	średni błąd	113.14%
60	3716	109.23%
	3707	108.73%
	3686	107.55%
	3811	114.58%
	3800	113.96%
	średni błąd	110.81%
120	3734	110.25%
	3507	97.47%
	3696	108.11%
	3709	108.84%
	3896	119.37%
	średni błąd	108.81%
240	3605	102.98%
	3867	117.74%
	3702	108.45%
	3764	111.94%
	3798	113.85%
	średni błąd	110.99%

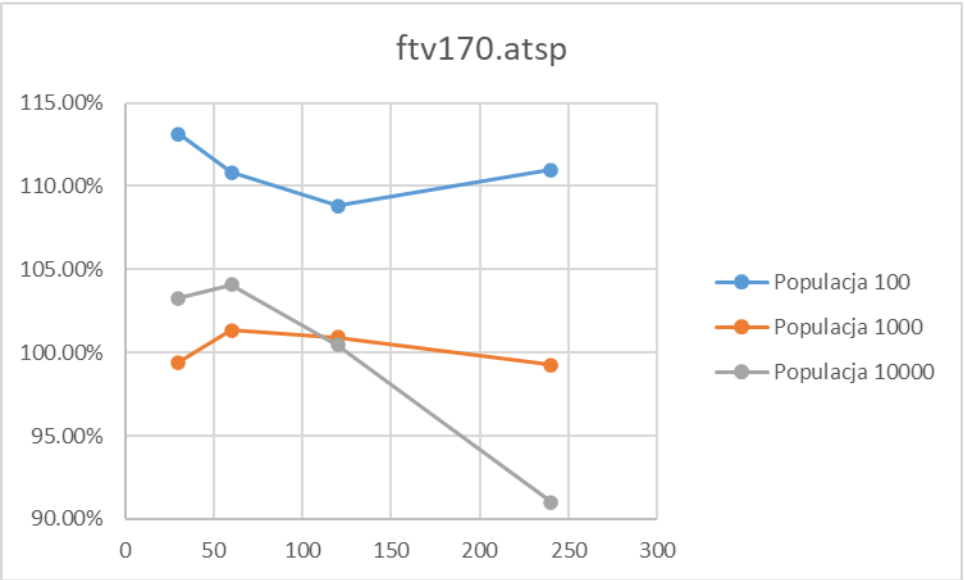
Plik:	ftv170.atsp	Błąd
Wynik optymalny:	1776	
Prozmiar populacji	1000	
Czas [s]	Wynik	
30	3548	99.77%
	3377	90.15%
	3653	105.69%
	3597	102.53%
	3533	98.93%
	średni błąd	99.41%
60	3602	102.82%
	3464	95.05%
	3624	104.05%
	3649	105.46%
	3539	99.27%
	średni błąd	101.33%
120	3605	102.98%
	3439	93.64%
	3672	106.76%
	3638	104.84%
	3488	96.40%
	średni błąd	100.92%
240	3492	96.62%
	3556	100.23%
	3494	96.73%
	3527	98.59%
	3627	104.22%
	średni błąd	99.28%

Plik:	ftv170.atsp	Błąd
Wynik optymalny:	1776	
Prozmiar populacji	10000	
Czas [s]	Wynik	
30	3645	105.24%
	3558	100.34%
	3625	104.11%
	3576	101.35%
	3647	105.35%
	średni błąd	103.28%
60	3559	100.39%
	3636	104.73%
	3642	105.07%
	3632	104.50%
	3652	105.63%
	średni błąd	104.07%
120	3575	101.30%
	3593	102.31%
	3550	99.89%
	3590	102.14%
	3494	96.73%
	średni błąd	100.47%
240	3316	86.71%
	3436	93.47%
	3324	87.16%
	3548	99.77%
	3339	88.01%
	średni błąd	91.02%

Czas [s]	Błąd[%]
30	113.14%
60	110.81%
120	108.81%
240	110.99%

Czas [s]	Błąd[%]
30	99.41%
60	101.33%
120	100.92%
240	99.28%

Czas [s]	Błąd[%]
30	103.28%
60	104.07%
120	100.47%
240	91.02%



Plik:	rbg403.atsp	Błąd
Wynik optymalny:	1776	
Prozmiar populacji	100	
Czas [s]	Wynik	
30	3426	92.91%
	3470	95.38%
	3454	94.48%
	3456	94.59%
	3443	93.86%
	średni błąd	94.25%
60	3451	94.31%
	3419	92.51%
	3449	94.20%
	3444	93.92%
	3447	94.09%
	średni błąd	93.81%
120	3418	92.45%
	3388	90.77%
	3407	91.84%
	3403	91.61%
	3378	90.20%
	średni błąd	91.37%
240	3420	92.57%
	3366	89.53%
	3410	92.00%
	3326	87.27%
	3388	90.77%
	średni błąd	90.43%

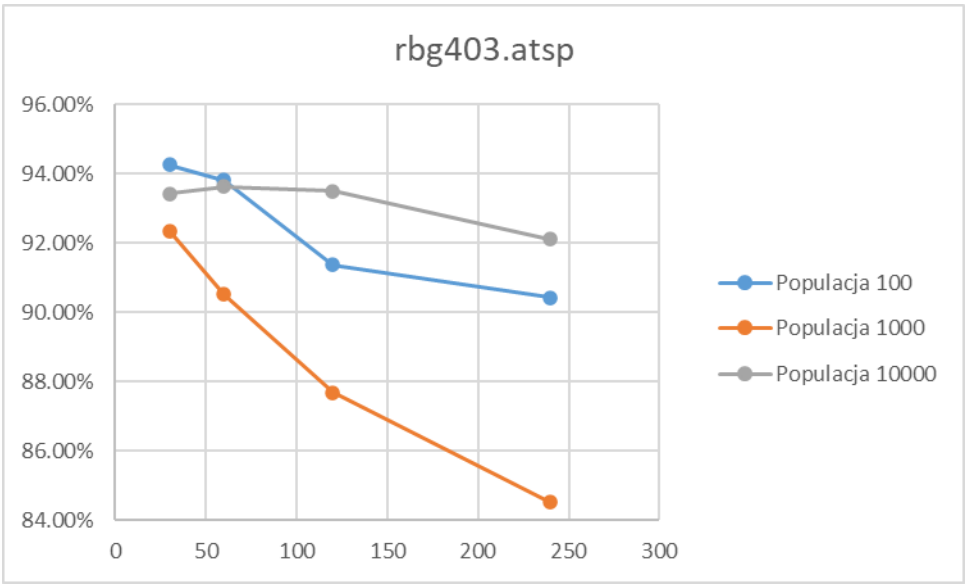
Plik:	rbg403.atsp	Błąd
Wynik optymalny:	1776	
Prozmiar populacji	1000	
Czas [s]	Wynik	
30	3421	92.62%
	3399	91.39%
	3413	92.17%
	3408	91.89%
	3440	93.69%
	średni błąd	92.35%
60	3365	89.47%
	3376	90.09%
	3406	91.78%
	3383	90.48%
	3388	90.77%
	średni błąd	90.52%
120	3306	86.15%
	3399	91.39%
	3314	86.60%
	3321	86.99%
	3327	87.33%
	średni błąd	87.69%
240	3280	84.68%
	3235	82.15%
	3271	84.18%
	3326	87.27%
	3274	84.35%
	średni błąd	84.53%

Plik:	rbg403.atsp	Błąd
Wynik optymalny:	1776	
Prozmiar populacji	10000	
Czas [s]	Wynik	
30	3430	93.13%
	3439	93.64%
	3443	93.86%
	3426	92.91%
	3438	93.58%
	średni błąd	93.42%
60	3439	93.64%
	3439	93.64%
	3433	93.30%
	3446	94.03%
	3436	93.47%
	średni błąd	93.61%
120	3441	93.75%
	3443	93.86%
	3433	93.30%
	3429	93.07%
	3436	93.47%
	średni błąd	93.49%
240	3396	91.22%
	3426	92.91%
	3409	91.95%
	3415	92.29%
	3413	92.17%
	średni błąd	92.11%

Czas [s]	Błąd[%]
30	94.25%
60	93.81%
120	91.37%
240	90.43%

Czas [s]	Błąd[%]
30	92.35%
60	90.52%
120	87.69%
240	84.53%

Czas [s]	Błąd[%]
30	93.42%
60	93.61%
120	93.49%
240	92.11%



d) Wnioski

Dla ftv47.atsp najlepszy wynik to:

Genetyczny: koszt: 1878, ścieżka: [11, 10, 9, 33, 27, 2, 28, 3, 24, 4, 29, 30, 5, 31, 6, 8, 32, 7, 23, 12, 13, 46, 36, 14, 16, 45, 20, 38, 37, 0, 25, 18, 17, 34, 35, 15, 19, 44, 39, 21, 40, 47, 26, 22, 41, 43, 42, 1, 11]

Tabu search: koszt: 2052, ścieżka: [34, 13, 46, 36, 14, 16, 45, 20, 38, 18, 17, 12, 32, 6, 24, 4, 3, 33, 27, 28, 2, 41, 43, 22, 40, 47, 26, 42, 1, 0, 25, 10, 11, 8, 9, 29, 30, 5, 31, 7, 23, 35, 15, 19, 44, 39, 21, 37, 34]

Dla ftv170.atsp najlepszy wynik to:

Genetyczny: koszt: 3316 ścieżka: [151, 21, 25, 150, 160, 152, 142, 141, 134, 131, 113, 164, 127, 126, 125, 124, 121, 120, 122, 123, 162, 102, 103, 117, 118, 119, 129, 128, 130, 135, 138, 139, 140, 6, 7, 8, 9, 10, 76, 74, 75, 11, 12, 18, 19, 20, 158, 32, 36, 157, 33, 31, 30, 28, 27, 26, 23, 22, 16, 17, 29, 24, 15, 159, 13, 37, 38, 39, 40, 156, 34, 35, 41, 155, 42, 45, 44, 46, 47, 48, 51, 52, 53, 43, 55, 54, 58, 59, 60, 50, 49, 170, 73, 77, 1, 2, 0, 81, 80, 79, 82, 78, 72, 71, 61, 68, 67, 167, 70, 87, 85, 86, 83, 84, 69, 66, 65, 64, 56, 57, 62, 63, 88, 153, 154, 89, 90, 91, 94, 96, 97, 165, 163, 99, 98, 95, 92, 93, 166, 108, 168, 3, 4, 5, 133, 169, 111, 112, 132, 110, 109, 107, 106, 105, 100, 101, 104, 114, 115, 116, 136, 137, 146, 145, 144, 143, 147, 148, 149, 161, 14, 151]

Tabu search: koszt: 3440 , ścieżka: [144, 143, 141, 134, 131, 113, 164, 127, 126, 125, 124, 121, 120, 122, 123, 162, 102, 103, 117, 118, 119, 129, 128, 130, 135, 138, 139, 140, 6, 7, 8, 9, 10, 76, 74, 75, 11, 12, 18, 19, 20, 21, 22, 23, 27, 28, 29, 30, 31, 33, 35, 34, 156, 40, 39, 38, 37, 49, 170, 73, 77, 1, 2, 0, 81, 80, 79, 82, 78, 72, 71, 60, 50, 51, 52, 53, 43, 55, 54, 58, 59, 61, 68, 67, 167, 70, 87, 85, 86, 83, 84, 69, 66, 63, 64, 56, 57, 62, 65, 88, 153, 154, 89, 90, 91, 94, 96, 97, 99, 98, 95, 92, 93, 166, 108, 107, 106, 105, 165, 163, 100, 101, 104, 110, 109, 114, 115, 116, 136, 137, 147, 148, 149, 161, 152, 151, 14, 13, 17, 32, 158, 36, 157, 41, 155, 42, 45, 44, 46, 47, 48, 168, 3, 4, 5, 133, 169, 111, 112, 132, 142, 160, 150, 26, 24, 15, 159, 16, 25, 146, 145, 144]

Dla rbg403.atsp najlepszy wynik to:

Genetyczny: koszt: 3235 ścieżka: [170, 339, 369, 232, 159, 7, 11, 101, 10, 55, 360, 222, 185, 56, 394, 14, 62, 13, 205, 204, 23, 225, 24, 36, 32, 274, 46, 33, 376, 68, 38, 270, 19, 18, 42, 287, 83, 43, 34, 295, 50, 5, 260, 35, 9, 84, 107, 61, 257, 58, 8, 6, 64, 3, 2, 386, 47, 112, 322, 272, 28, 59, 76, 310, 29, 65, 12, 77, 31, 52, 40, 39, 78, 226, 147, 79, 69, 245, 81, 22, 21, 82, 247, 54, 85, 26, 67, 66, 60, 44, 86, 75, 27, 15, 1, 267, 281, 87, 70, 349, 73, 88, 96, 94, 90, 72, 57, 91, 25,

116, 92, 51, 49, 37, 249, 256, 120, 392, 351, 293, 93, 145, 143, 334, 240, 219, 328, 317, 312, 235, 121, 41, 365, 190, 100, 98, 95, 303, 71, 97, 99, 389, 187, 102, 80, 165, 359, 302, 89, 20, 0, 30, 269, 355, 115, 114, 353, 263, 167, 201, 118, 261, 393, 103, 117, 278, 122, 119, 168, 104, 203, 384, 383, 197, 146, 144, 105, 391, 154, 106, 340, 373, 4, 356, 358, 327, 108, 255, 212, 109, 275, 210, 110, 326, 258, 111, 45, 363, 289, 123, 192, 189, 176, 156, 325, 124, 284, 290, 125, 74, 214, 401, 216, 200, 198, 126, 323, 305, 215, 309, 127, 48, 128, 228, 224, 129, 350, 130, 131, 282, 357, 370, 395, 132, 243, 288, 291, 315, 133, 148, 390, 134, 306, 319, 135, 280, 279, 136, 137, 320, 139, 223, 344, 140, 239, 173, 354, 342, 266, 141, 259, 142, 199, 229, 338, 149, 347, 175, 388, 348, 283, 277, 341, 361, 375, 191, 233, 324, 352, 150, 300, 273, 227, 381, 151, 292, 329, 294, 138, 297, 152, 16, 153, 346, 400, 155, 378, 337, 157, 377, 158, 271, 362, 262, 276, 298, 63, 398, 396, 330, 343, 264, 113, 194, 380, 379, 265, 382, 296, 242, 299, 321, 368, 268, 177, 250, 252, 371, 399, 285, 181, 183, 17, 286, 53, 301, 202, 304, 307, 308, 311, 313, 314, 316, 318, 331, 332, 333, 248, 335, 336, 345, 364, 366, 367, 372, 374, 385, 387, 397, 402, 160, 161, 162, 221, 163, 164, 166, 169, 171, 172, 174, 178, 179, 180, 182, 184, 186, 188, 193, 195, 196, 206, 207, 208, 209, 211, 213, 217, 218, 220, 230, 231, 234, 236, 237, 238, 241, 244, 246, 251, 253, 254, 170]

Tabu search: koszt: 2574 , ścieżka: [23, 14, 62, 13, 205, 204, 24, 36, 32, 274, 46, 33, 376, 68, 38, 270, 19, 18, 402, 287, 83, 43, 34, 295, 50, 304, 394, 225, 58, 8, 6, 163, 3, 2, 61, 107, 386, 47, 112, 322, 257, 308, 5, 312, 35, 217, 84, 272, 28, 129, 360, 110, 314, 59, 153, 310, 29, 77, 31, 52, 40, 39, 78, 226, 147, 79, 69, 245, 81, 22, 21, 82, 247, 249, 172, 26, 246, 367, 75, 160, 15, 333, 267, 281, 221, 67, 66, 60, 44, 88, 96, 94, 90, 72, 57, 164, 25, 91, 92, 51, 49, 37, 251, 301, 120, 392, 351, 293, 93, 145, 101, 180, 165, 316, 364, 303, 71, 97, 387, 349, 73, 99, 389, 111, 369, 237, 124, 269, 355, 115, 114, 353, 263, 244, 80, 209, 341, 313, 20, 340, 169, 27, 239, 117, 278, 122, 119, 168, 142, 150, 384, 383, 361, 146, 144, 105, 391, 154, 155, 7, 393, 372, 356, 328, 317, 108, 256, 182, 109, 275, 210, 265, 326, 258, 140, 374, 363, 289, 123, 192, 189, 170, 284, 290, 125, 74, 214, 401, 216, 200, 198, 159, 323, 305, 215, 309, 162, 48, 358, 327, 234, 228, 224, 151, 271, 350, 307, 143, 285, 282, 357, 241, 113, 395, 132, 243, 175, 291, 315, 161, 148, 325, 134, 306, 319, 135, 280, 279, 264, 268, 16, 388, 254, 223, 344, 345, 173, 354, 342, 266, 141, 116, 385, 300, 302, 118, 149, 347, 352, 104, 9, 236, 227, 381, 188, 299, 298, 63, 297, 130, 178, 100, 138, 337, 181, 346, 338, 331, 378, 398, 396, 156, 390, 157, 1, 176, 318, 195, 179, 368, 366, 292, 329, 294, 330, 343, 136, 370, 380, 379, 230, 382, 212, 103, 336, 321, 362, 208, 85, 252, 371, 177, 283, 277, 375, 191, 187, 186, 53, 400, 262, 261, 193, 190, 86, 11, 106, 359, 133, 201, 89, 158, 339, 126, 55, 399, 377, 286, 273, 335, 207, 45, 184, 4, 131, 242, 70, 397, 260, 56, 232, 127, 213, 41, 64, 365, 166, 199, 373, 311, 334, 102, 276, 259, 128, 255, 194, 324, 320, 95, 10, 183, 17, 12, 332, 296, 174, 185, 167, 196, 0, 202, 206, 30, 222, 250, 42, 152, 76, 220, 197, 87, 137, 54, 231, 288, 233, 235, 121, 211, 98, 238, 229, 240, 219, 65, 218, 203, 248, 348, 171, 253, 139, 23]

Moja implementacja algorytmu genetycznego wydaje się dawać stosunkowo małe błędy dla mniejszych macierzy (mniejsze niż tabu search). Przy większych błąd jest stosunkowo duży (większy niż w tabu search).
Przeważnie wraz ze wzrostem czasu działania algorytm daje wyniki bardziej zbliżone do optimum.
Dla ftv47.atsp większy rozmiar populacji daje lepsze wyniki. W przypadku ftv170.atsp również wydaje się być podobnie. Tylko dla rbg403.atsp najbardziej efektywna jest populacja mająca 1000 osobników, czyli środkowa pod względem liczebności.