Politechnika Wrocławska

Wydział Elektroniki
Katedra Informatyki Technicznej

PROJEKTOWANIE EFEKTYWNYCH ALGORYTMÓW
PROJEKT

Projekt nr 3 - Implementacja i analiza efektywności algorytmu genetycznego (ewolucyjnego) dla wybranego problemu optymalizacji.

Miron Oskroba, 236705

Prowadzący - dr inż. Jarosław Mierzwa
Termin zajęć: Pn 15^{15}

Wrocław, 25 stycznia 2021

Spis treści

1 Wstęp teoretyczny **2**

2 Opis najważniejszych klas **4**

3 Plan eksperymentu **6**

4 Wyniki eksperymentów **6**

4.1 Tabele 6

4.2 Wykres 13

4.3 Zestawienie najlepszych rozwiązań - GA z SW 14

4.3.1 ftv47.atsp 14

4.3.2 ftv170.atsp 15

4.3.3 rbg403.atsp 15

5 Wnioski **17**

6 Literatura **18**

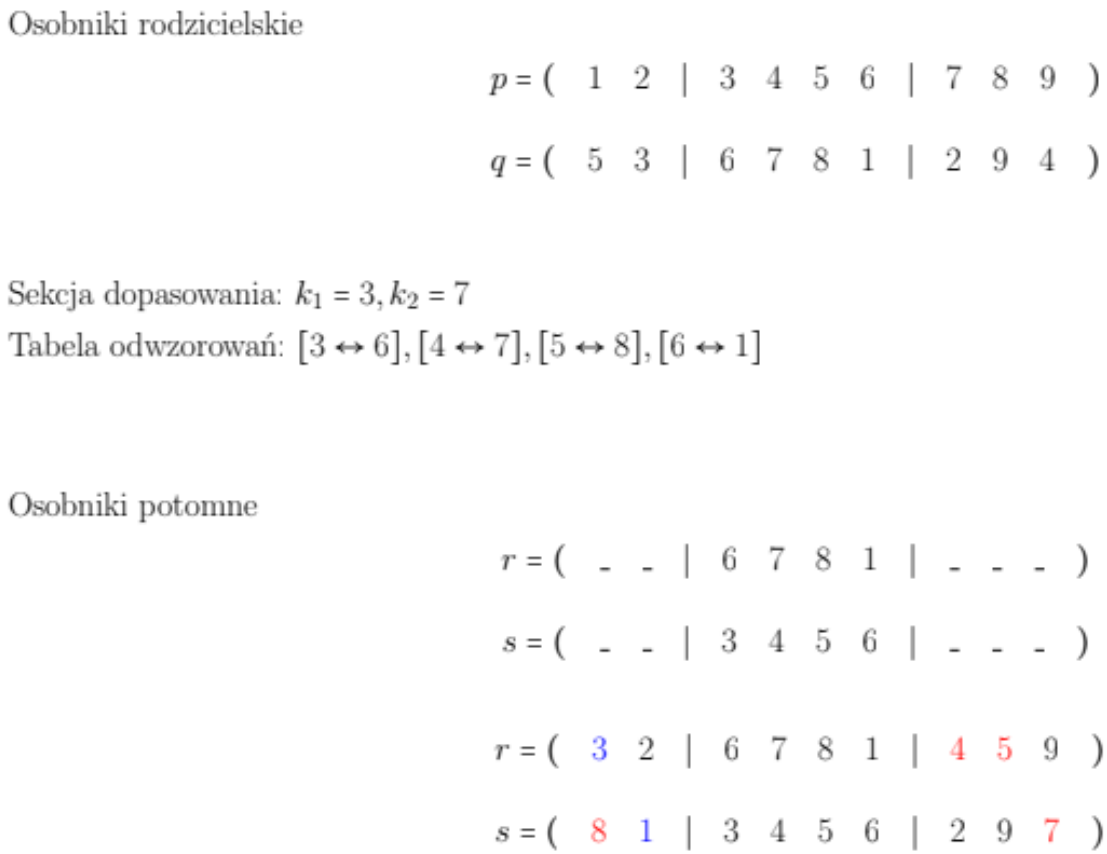
1 Wstęp teoretyczny

Celem wykonywanego zadania jest implementacja i analiza efektywności Algorytmu Genetycznego dla asymetrycznego problemu podróżującego komiwojażera oraz porównanie otrzymanych najlepszych wyników zwynikami algorytmu poprzedniego projektu - Symulowanego Wyżarzania. Zaimplementowane algorytmy:

- Algorytm Genetyczny dla ATSP dla dwóch rodzajów krzyżowania (PMX, OX).

Algorytm Genetyczny (z ang. 'Genetic Algorithm') jest algorytmem metaheurystycznym, który nie gwarantuje znalezienia rozwiązania optymalnego, jednak próbuje maksymalnie się do niego zbliżyć, działając w sposób niedeterministyczny. Algorytm posiada elementy:

- Współczynnik krzyżowania - Współczynnik określający prawdopodobieństwo na skrzyżowanie rodziców - ich przemiany w nowych osobników za pośrednictwem funkcji krzyżowania. Zaimplementowane zostały dwa rodzaje krzyżowania:
  - PMX - (z ang. PMX - 'Partially Mapped Crossover') - polega na wylosowaniu dwóch losowych indeksów oraz zamianie 'chromosomów' (miast) w tak powstałej sekcji dopasowania. Pozostałe pozycje uzupełnianie są na podstawie mapy odwzorowań powstałej podczas zamiany chromosoów na wylosowanych wcześniej pozycjach. Na rysunku 1 przedstawione zostało obrazowe działanie algorytmu na podstawie materiałów z prezentacji nr 7 - 'Algorytmy Genetyczne' prowadzącegoo wykład. Przykład PMX:



Rysunek 1: Przykład działania metody krzyżowania - PMX

  - OX - (z ang. 'Ordered Crossover') - krzyżowanie z częściowym odwzorowaniem, odmiana krzyżowania dwupuntkowego. Podobnie jak w PMX, w OX również występuje pojęcie *sekcji dopasowania*. Metody te różnią się natomiast sposobem dobierania pozostałych pozycji. Przykład OX - rysunki 2:, 3.

Osobniki rodzicielskie

$$p = (\quad p_1 \ldots \quad \overbrace{|p_{k_1} \ldots p_{k_2}|}^{\textit{sekcja dopasowania}} \quad \ldots \quad p_m \quad)$$

$$q = (\quad q_1 \ldots \quad \underbrace{|q_{k_1} \ldots q_{k_2}|}_{\textit{sekcja dopasowania}} \quad \ldots \quad q_m \quad)$$

Osobniki potomne

$$r = (\quad r_1 \ldots \quad \overbrace{|p_{k_1} \ldots p_{k_2}|}^{\textit{sekcja dopasowania}} \quad \ldots \quad r_m \quad)$$

$$s = (\quad s_1 \ldots \quad \underbrace{|q_{k_1} \ldots q_{k_2}|}_{\textit{sekcja dopasowania}} \quad \ldots \quad s_m \quad)$$

Indeksy k_1 i k_2 wybrane losowo z $[1 \ldots m]$. Kopiujemy do r te elementy z x, które nie występują w sekcji dopasowania skopiowanej z q - począwszy od pozycji $k_2 + 1$.

Rysunek 2: Przykład działania metody krzyżowania - OX cz. 1

Osobniki rodzicielskie

$$p = (\quad 1 \quad 2 \quad | \quad 3 \quad 4 \quad 5 \quad 6 \quad | \quad 7 \quad 8 \quad 9 \quad)$$

$$q = (\quad 5 \quad 3 \quad | \quad 6 \quad 7 \quad 8 \quad 1 \quad | \quad 2 \quad 9 \quad 4 \quad)$$

Osobniki potomne

$$r = (\quad 4 \quad 5 \quad | \quad 6 \quad 7 \quad 8 \quad 1 \quad | \quad 9 \quad 2 \quad 3 \quad)$$

$$s = (\quad 8 \quad 1 \quad | \quad 3 \quad 4 \quad 5 \quad 6 \quad | \quad 2 \quad 9 \quad 7 \quad)$$

Rysunek 3: Przykład działania metody krzyżowania - OX cz. 2

Współczynnik krzyżowania zgodnie z wytycznymi projektowymi został ustawiony na 0.8.

- Współczynnik mutacji - Po krzyżowaniu każdy osobnik może zmutować z określonym prawdopodobieństwem. W implementowanym algorytmie mutacja osobnika polega na wykonaniu losowo jednej z trzech akcji:
  - inwersję (z ang. 'inverse')
  - wstawianie (z ang. 'insert')
  - zamianę (z ang. 'swap')

korzystając z losowo wygenerowanych indeksów *i, j* - zamieniane są miasta. Zgodnie z wytycznymi projektowymi współczynnik mutacji został ustawiony na 0.01.

- Wielkość populacji - liczebność populacji określa liczbę osobników w danym pokoleniu. W każdym kolejnym pokoleniu jest ich tyle samo. Zostaną przeprowadzone badania sprawdzające wpływ trzech różnych wielkości populacji na efekt końcowy - znalezione najlepsze rozwiązania oraz czas jego wystąpienia.
- Selekcja - Polega na wybraniu z populacji osobników najbardziej przystosowanych do danego środowiska. Przykładowo w tym przypadku minimalizowana wartość to koszt ścieżki, dlatego selekcja zawiera definicję średniego kosztu ściezki dla danej operacji. W zależności od posiadanej przez osobnika ścieżki może zostać sklasyfikowany jako odpowiedni do przejścia 'selekcji'. Schemat, wg którego implementowano algorytm selekcji to 'selekcja elitarna' [1], czyli wybierająca najlepiej przystosowanych osobników w większym prawdopodobieństwem, oraz niektórych gorzej przystosowanych - analogicznie z mniejszym prawdopodobieństwem. Istotną rolę w algorytmie selekcji odgrywają współczynniki prawdopodobieństwa akceptacji danego osobnika w zależności od populacji i sprawiają trudność podczas dopasowywania.
- Mechanizmy wykrywające zdarzenia krytyczne (np. wejście w niechciane minimum lokalne) - Takie zdarzenie oznacza blokadę programu, znaczne obniżenie efektywności działania. Celem poprawienia wydajności zaimplementowano podstawowe akcje naprawcze. Przykładowo, po wykryciu braku poprawy wyniku przez okresloną liczbę iteracji następuje obowiązkowa mutacja całej populacji. Mechanizm ten w stopniu zauważalnym poprawia jakość wyniku w czasie.
- Warunek stopu - Czas w sekundach, po którym algorytm przerwie działanie i zwróci wyniki.

2 Opis najważniejszych klas

Projekt został napisany w IDE Intellij2020.3 CLion w języku C++. Najważniejsze klasy prezentują się następująco (Opis funkcji jest poglądowy, bez zwracanych wartości):

- **main** - Punkt wejściowy programu. Zawiera interfejs użytkownika:
  1. 1.Wczytanie danych z pliku i wyswietlenie
  2. 2.Kryterum stopu[s]
  3. 3.Wielkosc populacji poczatkowej
  4. 4.Wspolczynnik mutacji
  5. 5.Wspolczynnik krzyzowania
  6. 6. Wybor metody krzyzowania
  7. 7.Uruchom algorytm GA
  8. 8.ruchom algorytm SW(dla porownywania wynikow)
  9. 0.Wyjscie
- **Algorithm** - Klasa *Symulowanego Wyżarzania*. Aby uruchomić algorytm należy najpierw zainicjować instancję klasy podając w konstruktorze niezbędne parametry:
  - macierz kosztów
  - warunek stopu w sekundach
  - współczynnik krzyżowania
  - współczynnik mutacji
  - wielkość populacji
  - typ krzyżowania

  Funkcje w klasie:

– **GA()** - główne ciało algorytmu. W pierwszej kolejności przygotowuje niezbędne zmienne i dane do prawidłowej pracy nad aktualnie wczytaną instancją z pliku, następnie algorytm trwa do określonego momentu - do momentu warunku stopu wyrażonego w [s] przez użytkownika. Algorytm poglądowo działa w następujący sposób:

1. Inicjacja populacji początkowej - losowe rozwiązania ze zbioru rozwiązań
2. Powtarzaj dopóki kryt. stopu:
   (a) Ocena i selekcja populacji
   (b) Krzyżowanie populacji 0.8 szans na skrzyżowanie dwóch losowych osobników wybraną metodą krzyżowania
   (c) Mutacja populacji 0.01 szans na zmutowanie jednego z osobników
   (d) Porównanie najlepszego osobnika populacji z najlepszym rozwiązaniem
   (e) Sprawdzenie wystąpienia zdarzenia krytycznego - gdy wystąpi dokonać mutacji populacji.

– **selectPopulation(vector<Single> &population)** - Najpierw osobnicy są sortowani względem najmniejszego kosztu, następnie przebiega selekcja elitarna. Im gorzej dopasowany jest osobnik, tym inaczej dopasowywane jest prawdopodobieństwo wybrania go w zależności od wielkości populacji początkowej oraz rozmiaru badanej instancji - należy doświadczalnie wysterować parametry *fCut* oraz *fWeakProb*.

```
            struct Single // osobnik
    {
        Single() = default;
        int cost{};
        vector<int> path;
    };

            void selectPopulation(vector<Single> &population)
    {
        // SELEKCJA osobnikow - najlepsi sa na gorze
        sort(population.begin(), population.end(), SingleComparator());
        vector<Single> selectedPopulation;

        //selekcja elitarna
        int avgCost = avgPopulationCost(population);
        Single single;
        for(int i = 0; i < fPopulation * fCut ; i++)
        {
            single = population[i];
            if(single.cost < avgCost) // pozbywam sie gorszych osobnikow (wiekszy koszt)
                selectedPopulation.push_back(single);
            else if(rnd() <= fWeakProb) // akceptuje gorszych z pewnym prawdopodobienstwem
                selectedPopulation.push_back(single);
        }
        for(int i = fPopulation * fCut; i < fPopulation ; i++)
            if(rnd() <= fWeakProb*.2) // akceptuje gorszych z pewnym mniejszym prawdopodobienstwe
                selectedPopulation.push_back(population[i]);

        // kiedy zaden nie zostanie zaakceptowany
        if(selectedPopulation.empty())
        {
            mutate(lastPopulation, true);
            selectedPopulation = lastPopulation;
        }

        population = selectedPopulation;
    }
        }
```

- **cost(vector<int> &path)** - funkcja liczenia kosztu danej ścieżki.
- **crossPMX(vector<Single> &parentPopulation)** - funkcja odpowiedzialna za krzyżowanie PMX. 96% czasu pracy algorytmu dzieje się w tym miejscu w wybranym trybie krzyżowania. Funkcja, która wpływa najbardziej na czas działania algorytmu to 'inline void putCandidatePMX(vector<Item> &map, vector<bool> &tabu, int &candidate, int childNr)'.
- **static vector<vector<int> > read_from_file(const char* datafile)** - funkcja odpowiedzialna za wczytanie danych z pliku odpowiednio sformatowanym do pamięci programu.

- **Timer** - Umożliwia niezależne odmierzanie czasu(warunki stopu, czas wykrycia rozwiązania najlepszego)
  - **start()**
  - **stop()**
  - **info()**
  - **getTime()**

3 Plan eksperymentu

Plan eksperymentu odbędzie się zgodnie z wytycznymi projektowymi; dla plików zostane wykonaną testy jak w tabeli 1 dla trzech różnych wielkości populacji (800, 8.000, 20.000):

```
#define POP_SIZE_MIN 800
#define POP_SIZE_MID 8000
#define POP_SIZE_MAX 20000
```

Tablica 1: Plan ekserymentu.

nazwa pliku	ilość testów	czas testu[s]	wartość oczekiwana
ftv47.atsp	10	60	1776
ftv170.atsp	10	180	2755
rbg403.atsp	10	150	2465

DLa otrzymanych wyników policzone zostaną błędy względne wg wzoru:

$$|f_{zn} - f_{opt}|/f_{opt} * 100\%, gdzie$$

$$f_{zn} - \ warto\ obliczona\ przez\ algorytm$$

$$f_{opt} - warto\ oczekiwana(optymalna) - najlepsze\ znane\ rozwizanie$$

Dodatkowo, dla każdego uruchomienia wypisane zostaną:

- najlepszy wynik danego oraz uruchomienia
- jego czas wystąpienia .

4 Wyniki eksperymentów

4.1 Tabele

Wyniki eksperymentów dla N 'wielkości populacji' opisanych w Plan eksperymentu zaprezentowane są w tabelach:

- PMX:
- 2
- 3
- 4
- OX:
- 5
- 6
- 7

Tablica 2: Wyniki eksperymentu dla typu krzyżowania PMX dla liczności populacji 800 osobników.

plik	czas wykonywania algorytmu[s]	l.p.	znalezione rozwiązanie	czas wykrycia [s]	błąd względny[%]
ftv47.atsp	30	1	1938	24	9.1
		2	2070	25	16.6
		3	1939	24	9.2
		4	2065	8	16.3
		5	1976	5	11.3
		6	2125	22	19.7
		7	1992	21	12.2
		8	1948	21	9.7
		9	2002	14	12.7
		10	2037	24	14.7
ftv170.atsp	180	1	5018	133	82.1
		2	4965	118	80.2
		3	4796	37	74.1
		4	4794	146	74.0
		5	4858	163	76.3
		6	4826	153	75.2
		7	4822	163	75.0
		8	4982	155	80.8
		9	4955	88	79.9
		10	5002	147	81.6
rbg403.atsp	150	1	2676	149	8.6
		2	2740	150	11.2
		3	2678	149	8.6
		4	2627	151	6.6
		5	2703	119	9.7
		6	2680	104	8.7
		7	2673	148	8.4
		8	2652	151	7.6
		9	2658	151	7.8
		10	2670	139	8.3

Tablica 3: Wyniki eksperymentu dla typu krzyżowania PMX dla liczności populacji 8.000 osobników.

plik	czas wykonywania algorytmu[s]	l.p.	znalezione rozwiązanie	czas wykrycia [s]	błąd względny[%]
ftv47.atsp	30	1	1983	19	11.7
		2	2017	4	13.6
		3	2031	30	14.4
		4	1923	18	8.3
		5	2065	2	16.3
		6	1910	30	7.5
		7	2013	4	13.3
		8	1972	27	11.0
		9	2045	11	15.1
		10	1892	15	6.5
ftv170.atsp	180	1	4642	178	68.5
		2	4376	72	58.8
		3	4519	173	64.0
		4	4438	177	61.1
		5	4295	166	55.9
		6	4075	167	47.9
		7	4210	163	52.8
		8	3886	147	41.1
		9	4283	1025	55.5
		10	4095	143	48.6
rbg403.atsp	150	1	2714	145	10.1
		2	2668	241	8.2
		3	2700	136	9.5
		4	2687	126	9.0
		5	2662	116	8.0
		6	2681	121	8.8
		7	2639	148	7.1
		8	2713	150	10.1
		9	2856	137	15.9
		10	2622	150	6.4

Tablica 4: Wyniki eksperymentu dla typu krzyżowania PMX dla liczności populacji 20.000 osobników.

plik	czas wykonywania algorytmu[s]	l.p.	znalezione rozwiązanie	czas wykrycia [s]	błąd względny[%]
ftv47.atsp	30	1	1986	7	11.8
		2	1942	7	9.3
		3	1898	17	6.9
		4	1872	12	5.4
		5	2037	5	14.7
		6	1844	55	3.8
		7	2027	18	14.1
		8	1919	18	8.1
		9	1991	24	12.1
		10	1909	16	7.5
ftv170.atsp	180	1	3899	145	41.5
		2	3983	130	44.6
		3	3883	155	40.9
		4	4313	141	56.6
		5	4042	179	46.7
		6	4159	167	51.0
		7	4127	172	49.8
		8	4541	147	64.8
		9	4203	179	52.6
		10	4385	145	59.2
rbg403.atsp	150	1	2707	148	9.8
		2	2622	145	6.4
		3	2730	147	10.8
		4	2685	146	8.9
		5	2705	144	9.7
		6	2739	147	11.1
		7	2685	144	8.9
		8	2697	147	9.4
		9	2691	141	9.2
		10	2699	141	9.5

Tablica 5: Wyniki eksperymentu dla typu krzyżowania OX dla liczności populacji 800 osobników.

plik	czas wykonywania algorytmu[s]	l.p.	znalezione rozwiązanie	czas	błąd względny[%]
ftv47.atsp	60	1	2168	14	22.1
		2	2137	21	20.3
		3	2076	50	16.9
		4	2049	17	15.4
		5	2110	9	18.8
		6	2191	13	23.4
		7	1910	32	7.5
		8	2176	37	22.5
		9	2173	34	22.4
		10	2126	12	19.7
ftv170.atsp	180	1	8976	178	225.8
		2	9847	174	257.4
		3	10270	175	272.8
		4	8784	172	218.8
		5	10726	142	289.3
		6	9490	179	244.5
		7	10537	68	282.5
		8	9883	64	258.7
		9	10181	120	269.5
		10	9950	80	261.2
rbg403.atsp	180	1	4680	96	89.9
		2	4837	148	96.2
		3	4326	128	75.5
		4	4740	160	92.3
		5	4510	160	83.0
		6	4807	39	95.0
		7	5007	134	103.1
		8	4439	119	80.1
		9	4709	133	91.0
		10	4595	118	86.4

Tablica 6: Wyniki eksperymentu dla typu krzyżowania OX dla liczności populacji 8.000 osobników.

plik	czas wykonywania algorytmu[s]	l.p.	znalezione rozwiązanie	czas	błąd względny[%]
ftv47.atsp	60	1	1919	48	8.1
		2	2000	60	12.6
		3	2017	42	13.6
		4	2018	44	13.6
		5	2051	55	15.5
		6	2166	58	22.0
		7	2133	42	20.1
		8	2106	52	18.6
		9	2466	60	38.9
		10	1915	60	7.8
ftv170.atsp	180	1	11201	167	306.6
		2	10664	179	287.1
		3	9969	180	261.9
		4	12450	170	351.9
		5	7705	180	179.7
		6	15026	153	445.4
		7	18282	174	563.6
		8	18651	180	577.0
		9	9165	161	232.7
		10	8675	165	214.9
rbg403.atsp	180	1	3050	102	23.7
		2	2970	110	20.5
		3	3040	114	23.3
		4	2866	163	16.3
		5	2891	147	17.3
		6	3055	90	23.9
		7	2985	117	21.1
		8	2987	104	21.2
		9	2962	120	20.2
		10	2921	153	18.5

Tablica 7: Wyniki eksperymentu dla typu krzyżowania OX dla liczności populacji 20.000 osobników.

plik	czas wykonywania algorytmu[s]	l.p.	znalezione rozwiązanie	czas	błąd względny[%]
ftv47.atsp	60	1	3348	58	88.5
		2	3920	58	120.7
		3	3604	60	102.9
		4	3895	60	119.3
		5	3585	54	101.9
		6	4045	48	127.8
		7	3920	56	120.7
		8	3718	59	109.3
		9	3434	53	93.4
		10	3288	52	85.1
ftv170.atsp	180	1	19238	149	598.3
		2	19316	155	601.1
		3	19371	149	603.1
		4	10460	180	279.7
		5	19249	176	598.7
		6	19112	126	593.7
		7	10100	177	266.6
		8	18795	179	582.2
		9	19829	153	619.7
		10	19212	180	597.4
rbg403.atsp	180	1	2951	18	19.7
		2	2920	178	18.5
		3	2907	178	17.9
		4	2992	180	21.4
		5	2963	178	20.2
		6	2941	179	19.3
		7	2925	179	18.7
		8	2927	179	18.7
		9	2955	178	19.9
		10	2981	180	20.9

Wyniki uśrednione dla poszczególnych uruchomień zaprezentowane są w tabelach:

- PMX:
- 8
- 9
- 10
- OX:
- 11
- 12
- 13

Tablica 8: Uśrednione wyniki eksperymentu dla typu krzyżowania PMX dla liczności populacji 800 osobników.

plik	czas wykonywania algorytmu[s]	uśrednione rozwiązanie	błąd względny[%]	oczekiwane	uśredniony czas[s]	Najlepsze rozwiązanie
ftv47.atsp	30	2009.2	13.1	1776	19	1938
ftv170.atsp	180	4901.8	77.9	2755	130	4794
rbg403.atsp	150	2675.7	8.5	2465	141	2627

Tablica 9: Uśrednione wyniki eksperymentu dla typu krzyżowania PMX dla liczności populacji 8.000 osobników.

plik	czas wykonywania algorytmu[s]	uśrednione rozwiązanie	błąd względny[%]	oczekiwane	uśredniony czas[s]	Najlepsze rozwiązanie
ftv47.atsp	30	1985.1	11.8	1776	16	1892
ftv170.atsp	180	4281.9	55.4	2755	241	3886
rbg403.atsp	150	2694.2	9.3	2465	147	2622

Tablica 10: Uśrednione wyniki eksperymentu dla typu krzyżowania PMX dla liczności populacji 20.000 osobników.

plik	czas wykonywania algorytmu[s]	uśrednione rozwiązanie	błąd względny[%]	oczekiwane	uśredniony czas[s]	Najlepsze rozwiązanie
ftv47.atsp	30	1942.5	9.4	1776	18	1844
ftv170.atsp	180	4153.5	50.8	2755	156	3883
rbg403.atsp	150	2696	9.4	2465	145	2622

Tablica 11: Uśrednione wyniki eksperymentu dla typu krzyżowania OX dla liczności populacji 800 osobników.

plik	czas wykonywania algorytmu[s]	uśrednione rozwiązanie	błąd względny[%]	oczekiwane	uśredniony czas[s]	Najlepsze rozwiązanie
ftv47.atsp	60	2111.6	18.9	1776	24	1910
ftv170.atsp	180	9864.4	258.1	2755	135	8784
rbg403.atsp	180	4665	89.2	2465	124	4326

Tablica 12: Uśrednione wyniki eksperymentu dla typu krzyżowania OX dla liczności populacji 8.000 osobników.

plik	czas wykonywania algorytmu[s]	uśrednione rozwiązanie	błąd względny[%]	oczekiwane	uśredniony czas[s]	Najlepsze rozwiązanie
ftv47.atsp	60	2079.1	17.1	1776	52	1915
ftv170.atsp	180	12178.8	342.1	2755	171	7705
rbg403.atsp	180	2972.7	20.6	2465	122	2866

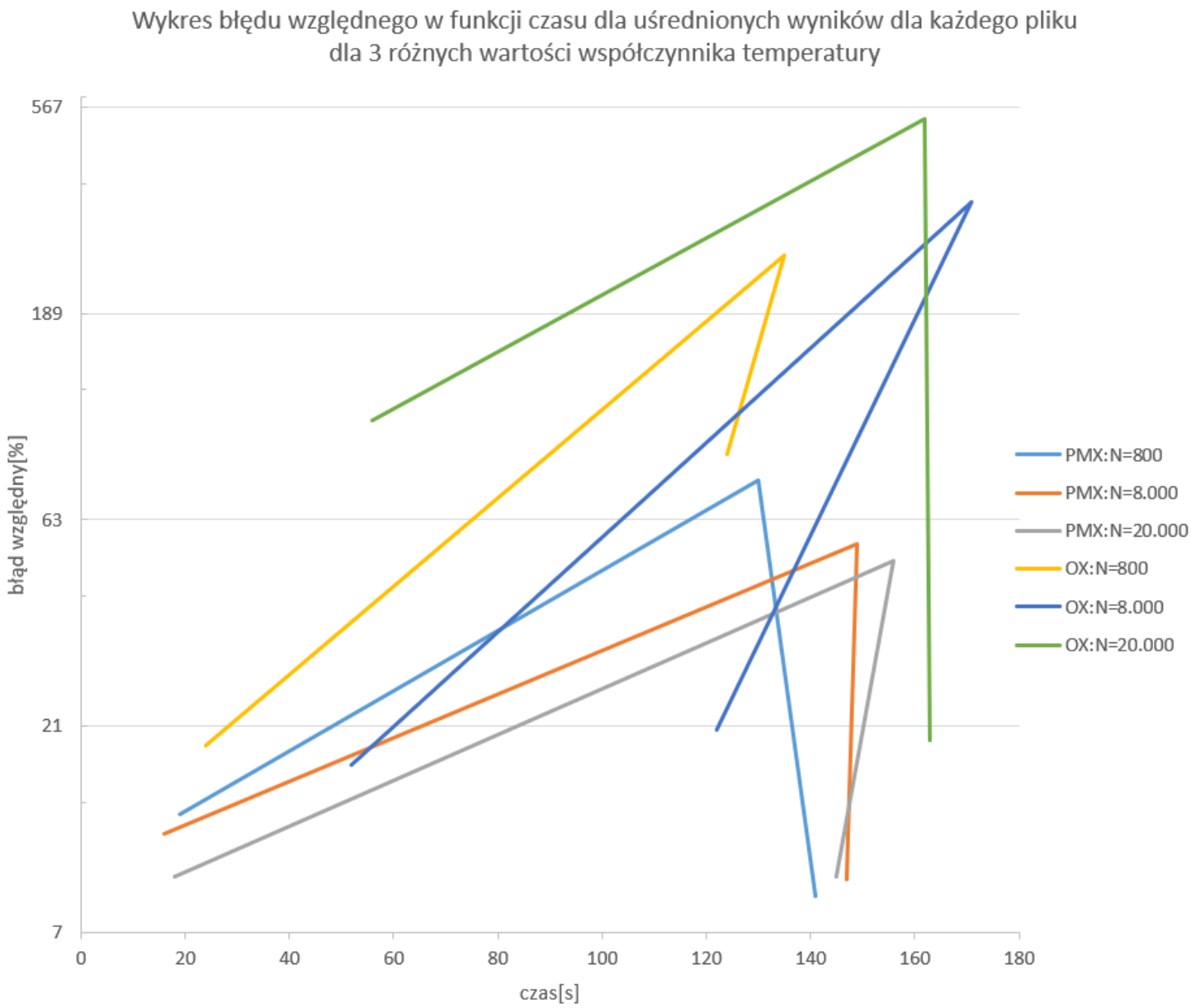
Tablica 13: Uśrednione wyniki eksperymentu dla typu krzyżowania OX dla liczności populacji 20.000 osobników.

plik	czas wykonywania algorytmu[s]	uśrednione rozwiązanie	błąd względny[%]	oczekiwane	uśredniony czas[s]	Najlepsze rozwiązanie
ftv47.atsp	60	3675.7	107.0	1776	56	3288
ftv170.atsp	180	17468.2	534.1	2755	162	10100
rbg403.atsp	180	2946.2	19.5	2465	163	2907

4.2 Wykres

Na podstawie danych z ww. tabel na podstawie wartości z kolumn czasu średniego oraz uśrednionego błędu względnego stworzono wykres jak na (Rys. 4). Zgdnie z wytycznymi projektowymi współczynniki zostały ustawione następująco:

- krzyżowania - 0.8
- mutacji = 0.01



Rysunek 4: Wykres błędu względnego w funkcji czasu dla uśrednionych wyników dla każdego pliku dla 3 różnych wartości wielkości populacji.

4.3 Zestawienie najlepszych rozwiązań - GA z SW

Poniżej zaprezentowane są ścieżki najlepszych rozwiązań dla algorytmów: SW oraz GA.

4.3.1 ftv47.atsp

(SW): Koszt: 1828
Ścieżka:

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

(GA): Koszt: 1844
Ścieżka:

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

4.3.2 ftv170.atsp

(SW): Koszt: 2971
Ścieżka:

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

(GA): Koszt: 3883
Ścieżka:

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

4.3.3 rbg403.atsp

(SW): Koszt: 2487
Ścieżka:

0 − 92 − 347 − 352 − 361 − 51 − 259 − 184 − 265 − 326 − 390 − 207 − 45 − 340 − 231 − 205 − 204 − 222 − 250 − 388 − 348 − 329 − 294 − 272 − 356 − 391 − 210 − 182 − 149 − 146 − 381 − 183 − 17 − 12 − 242 − 70 − 295 − 201 − 89 − 316 − 120 − 116 − 366 − 321 − 289 − 236 − 227 − 320 − 264 − 105 − 19 − 57 − 64 − 15 − 245 − 248 − 235 − 121 − 169 − 334 − 385 − 67 − 189 − 88 − 96 − 66 − 273 − 395 − 373 − 30 − 302 − 118 − 252 − 371 − 83 − 230 − 226 − 325 − 111 − 369 − 237 − 135 − 270 − 171 − 63 − 18 − 97 − 276 − 49 − 296 − 133 − 176 − 31 − 400 − 221 − 199 − 136 − 142 − 301 − 351 − 293 − 81 − 22 − 21 − 181 − 346 − 94 − 314 − 313 − 323 − 168 − 213 − 28 − 318 − 179 − 368 − 193 − 104 − 349 − 344 − 178 − 100 − 44 − 24 − 13 − 288 − 254 − 223 − 309 − 164 − 25 − 202 − 125 − 74 − 354 − 342 − 198 − 128 − 186 − 53

− 305 − 215 − 6 − 123 − 399 − 266 − 90 − 72 − 122 − 119 − 115 − 114 − 62 − 345 − 173 − 319 − 78 − 382 − 144 − 159 − 7 − 177 − 387 − 34 − 174 − 185 − 232 − 244 − 80 − 131 − 355 − 73 − 172 − 26 − 32 − 274 − 188 − 299 − 298 − 291 − 379 − 217 − 203 − 153 − 143 − 271 − 370 − 322 − 93 − 180 − 359 − 166 − 300 − 206 − 267 − 317 − 234 − 228 − 398 − 396 − 327 − 262 − 261 − 68 − 335 − 10 − 402 − 287 − 281 − 246 − 163 − 3 − 42 − 360 − 353 − 341 − 151 − 11 − 397 − 35 − 124 − 269 − 211 − 233 − 324 − 212 − 38 − 86 − 59 − 157 − 1 − 87 − 137 − 307 − 75 − 79 − 69 − 304 − 5 − 56 − 150 − 145 − 190 − 138 − 224 − 77 − 156 − 258 − 372 − 278 − 297 − 127 − 192 − 338 − 82 − 247 − 308 − 394 − 393 − 99 − 389 − 110 − 275 − 154 − 238 − 229 − 14 − 375 − 191 − 187 − 141 − 392 − 312 − 194 − 380 − 362 − 241 − 113 − 126 − 55 − 333 − 158 − 339 − 195 − 280 − 279 − 106 − 165 − 285 − 282 − 357 − 328 − 257 − 109 − 134 − 243 − 208 − 162 − 48 − 47 − 112 − 91 − 167 − 54 − 108 − 256 − 249 − 160 − 365 − 170 − 284 − 139 − 4 − 239 − 117 − 95 − 268 − 16 − 367 − 310 − 52 − 40 − 263 − 50 − 220 − 197 − 152 − 76 − 401 − 29 − 65 − 364 − 303 − 377 − 311 − 27 − 253 − 140 − 101 − 129 − 255 − 161 − 148 − 147 − 374 − 363 − 315 − 175 − 103 − 209 − 41 − 102 − 240 − 219 − 358 − 46 − 33 − 376 − 58 − 8 − 214 − 155 − 20 − 331 − 378 − 337 − 23 − 225 − 332 − 37 − 36 − 283 − 277 − 39 − 85 − 260 − 43 − 84 − 107 − 386 − 2 − 61 − 350 − 218 − 384 − 383 − 9 − 286 − 60 − 98 − 251 − 330 − 343 − 130 − 132 − 306 − 216 − 200 − 71 − 336 − 292 − 290 − 196 − 0

(GA): Koszt: 2622
Ścieżka:

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

Porównanie najlepszych wyników algorytmów Symulowanego Wyżarzania oraz Genetycznego:

Tablica 14: Zestawienie najlepszych znalezionych rozwiązań algorytmów: Genetycznego(GA) oraz Symulowanego Wyżarzania(SW) wraz z różnicą ich błędów względnych.

plik	najlepsze znane rozwiązanie	best GA	best SW	różnica błędów względnych[%]
ftv47.atsp	1776	1844	1828	0.9
ftv170.atsp	2755	3883	2971	33.1
rbg403.atsp	2465	2622	2487	5.5

5 Wnioski

Algorytm Genetyczny(GA) dla asymetrycznego problemu komiwojażera(ATSP) pomaga przybliżyć rozwiązanie dla dużych instancji macierzy kosztów wejściowych w nieporównywalnie krótszym czasie niż algorytmy: BruteForce, Branch&Bound. W zależności od instancji wyniki przybliżeń są różne. Np. dla instancji wejściowej: macierzy kosztów o rozmiarze 170 przybliżenie było średnio około 10 razy gorsze niż w przypadku instancji o rozmiarze 403. Może to być spowodowane faktem, że postacie macierzy są zbudowane inaczej, a generator liczb pseudolosowych działa w pewien określony sposób. Korzystano z *std::uniform_real_distribution* oraz z generatora *std::mt19937*, który korzystał z *std::random_device*, który pojawił się po raz pierwszy w edycji języka C++11.

Pierwszym faktem najbardziej rzucającym się w oczy jest fakt, że w tabeli 14 wyniki pomiarów pokazują, że algorytm genetyczny poradził sobie w każdym przypadku gorzej niż algorytm symulowanego wyżarzania. W przypadku najmniejszej instancji wyniki są bardzo zbliżone, niemal identyczne, ponieważ błąd względny wynosi 0.9%. W przypadku instancji o wielkości 170 miast różnica jest największa i wynosi 33.1%, a dla największego rozmiaru instancji błąd względny wynosi zaledwie 5.5%. Biorąc pod uwagę czasy wykonywania oraz rozwiązania średnie dla danych problemów można stwierdzić, że statystycznie algorytm Symulowanego Wyżarzania zachowuje się lepiej dla dużych rozmiarów instancji (170+).

Na wykresie 4 widać, że wielkość populacji ma wpływ na najlepsze znalezione rozwiązanie. Wyszczególniono (N oznacza wielkość populacji):

1. PMX:N=800 - Dla plików ftv47 oraz rbg403 dobre wyniki, aczkolwiek dla bardziej licznych populacji wyniki są lepsze. Średnia wyników dla instancji rbg403 jest najlepsza ze wszystkich.
2. PMX:N=8000 - Podobnie do populacji 20.000, aczkolwiek dla mniejszego rozmiaru instancji błąd jest większy o ok. 5%.
3. PMX:N=20000 - prawdopodobnie najlepsze dopasowanie - najlepsze wyniki dla wszystkich plików oraz średnia rozwiązań najlepsza dla plików ftv47 oraz ftv170.
4. OX:N=800 - Dla pliku ftv47 wynik nieco gorszy niż w PMX, dla pozostałych błędy zauważalnie gorsze niż w PMX
5. OX:N=8000 - Dla plików ftv47 oraz rbg403 wyniki porównywalne do PMX, chociaż nieco gorsze. Dla pliku ftv170 błąd rzędu 400% jest nieakceptowalny.
6. OX:N=20000 - Ze względu na wielkość populacji oraz czasochłonność algorytmu krzyżowania OX zostaje sklasyfikowany jako najgorszy. Populacja zbyt duża jak na czas rozwiązywania.

Dla wybranego lepszego krzyżowania - PMX zwrócono szczególną uwagę na następujące wyniki:

- Plik *ftv47.atsp* najlepiej pracuje z dużą licznością populacji
- Plik *ftv170.atsp* pracuje gorzej z mniejszą licznością populacji
- Plik *rbg403.atsp* pracuje najlepiej z małą wielkością populacji

Na tej podstawie można wyciągnąć wniosek, że im więcej osobników znajduje się w populacjii, tym więcej jest informacji i tym lepiej można dopasować rozwiązanie. Początkowo jest lepsza dywersyfikacja, a intensyfikacja oddala się w czasie, mając większy wachlarz możliwości.
Należy mieć na uwadze parametry występujące w selekcji osobników do następnego pokolenia. Doświadczalne ich dopasowywanie zajmuje sporo czasu, ponieważ jest to problem NP-trudny.

Następnie zauważono, że w algorytmie genetycznym metoda krzyżowania PMX wypadła lepiej od OX. Widać to na wykresie 4, gdzie wszystkie serie danych odpowiedzialne za wyniki działania algorytmu OX znajdują się wyżej na wykresie względem rozwiązań z PMX. Zestawiając otrzymane wyniki stwierdzono, że największa różnica w wyszczególnionych metodach krzyżowania nastąpiła dla instancji o wielkości 170 miast, gdzie różnica błędu względnego średnio wynosi około 400-500%. Być może gdyby dać więcej czasu dla OX, to wynik mógłby się polepszyć. Nie mniej jednak PMX wygląda czasowo lepiej. Może też tak być dlatego, że 'dostrajanie' Algorytmu Genetycznego nie należy do najłatwiejszych zadań. Podczas selekcji elitarnej akceptuje się część najlepszych rozwiązań oraz z pewnym prawdopodobieństwem akceptuje pozostałą część gorszych. Algorytm został nastrojony pod krzyżowanie PMX.

Podczas pisania algorytmu zadbano o jakość i moc wykonywania algorytmu:

1. Testowana wersja programu została zbudowana w wersji 'Release'
2. W środowisku VisualStudio2019 użyto Analizera Profilowania, celem zebrania informacji, w których miejscach w kodzie CPU jest wykorzystywany najbardziej, co pozwoliło na napoisanie lepszego kodu usprawniającego pracę.
   (a) W algorytmie z krzyżowaniem PMX(podobnie jak w wersji z OX) najwięcej czasu (ok. 98%) zajmowała funkcja sprawdzania 'chromosomów'(miast), tak aby włożyć dany chromosom(kandydata) w odpowiednie miejsce 'DNA'(ścieżka kolejnych miast) potomka. Przed optymalizacją funkcja wstawiania kandydata zajmowała czasowo ok. 67% czasu trwania programu. Po stworzeniu tablicy boolowskiej zamiast iterowania do momentu znalezienia powtórzeń czas wykonywania spadł do 35%.
   Polepszenie czasowe 67% —> 35%.
   (b) W przypadku OX analogicznie jak w PMX najwięcej czasu zajmowała funkcja wstawiania odpowiedniego kandydata na dane miejsce. Czasowo krzyżowanie OX zajmowało ok. 98% czasu trwania programu, z czego funkcja wstawiania ok.95%. Po implementacji ulepszonego mechanizmu iterującego zużycie czasowe spadło do ok. 65%.
   Polepszenie czasowe 95% —> 65%.
3. W kodzie końcowym podczas testów ograniczono wyświetlanie do konsoli do wystarczającego minimum - niezbędnych informacji nt. wyników poszczególnych pomiarów
4. Przed uruchomieniem testu zmieniono priorytet wykonywanego procesu z 'Normal' na 'Realtime', co w OS Windows oznacza największy priorytet
5. W flagach pliku CMake ustawiono flagi przyspieszające liczenie matematyki '-ffast-math'

Tak zoptymalizowane środowisko, w którym pracuje algorytm srpawiło, że czas wykonywania algorytmu uległ znacznemu polepszeniu, przez co algorytm jest efektywniejszy w działaniu.

6 Literatura

[1] 'Comparison of Parents Selection Methods of Genetic Algorithm for TSP', Chetan Chudasama, Mahesh Panchal
[data-dostępu : 25 stycznia 2021]
https://www.researchgate.net/profile/Chetan_Chudasama/publication/285690217_Comparison_of_parents_selection_m
of-parents-selection-methods-of-genetic-algorithm-for-TSP.pdf?origin=publication$_detail$