Politechnika Wrocławska

Wydział informatyki i Telekomunikacji

Projektowanie Efektywnych Algorytmów

Projekt – zadanie 2.

Autor:

Stanisław Strauchold 259142

Prowadzący:

dr inż. Jarosław Mierzwa

Grupa:

Środa 11:15 – 13:00 – grupa wcześniejsza

Termin oddania:

8.12.2022

1. Wstęp teoretyczny

W napisanym przeze mnie programie znajdują się dwa algorytmy: algorytm zachłanny oraz algorytm symulowanego wyżarzania.

- Algorytm zachłanny – jest on najprostszym algorytmem przeszukiwania lokalnego. Jego idea polega na pełnym przeglądzie sąsiedztwa bieżącego rozwiązania. Algorytm zachłanny kończy pracę jeżeli w sąsiedztwie bieżącego rozwiązania nie istnieją rozwiązania o mniejszej wartości funkcji celu.
- Algorytm symulowanego wyżarzania – jest on modyfikacją algorytmu zachłannego polegającą na wprowadzeniu pewnych elementów losowych. Rezygnuje się z pełnego przeglądania sąsiedztwa. Zamiast tego kolejne rozwiązanie wybierane jest w sposób losowy.

Algorytm SW jest algorytmem heurystycznym służącym do określania przybliżonego optimum dla danego problemu, Działa on iteracyjnie, przybliżając nas do optymalnego rozwiązania. Wybór elementu z przestrzeni potencjalnych rozwiązań zależy od różnicy między starym rozwiązaniem, a propozycją nowego oraz temperatury T.

Propozycja nowego rozwiązania zostaje uznana jako nowe rozwiązanie jeżeli jest ona lepsza od starego. Może być również uznana za nowe rozwiązanie z pewnym prawdopodobieństwem.

Parametr T steruje wyborem kolejnych przybliżeń w przypadku gdy propozycja nowego rozwiązania jest gorsza od starego. Im wyższa wartość T, tym większe prawdopodobieństwo wyboru gorszego rozwiązania. W trakcie działania algorytmu, wartość parametru jest stale obniżana.

Prawdopodobieństwo przyjęcia nowego, gorszego rozwiązania określa się w następujący sposób:

$$P(\boldsymbol{x}_0, \boldsymbol{x}) = \min\{1, \exp\left(-\frac{f(\boldsymbol{x}) - f(\boldsymbol{x}_0)}{T_i}\right)\},$$

W powyższym wzorze T oznacza aktualną wartość temperatury, f(x) jest proponowanym rozwiązaniem, zaś f(x0) jest starym rozwiązaniem.

Istotnym elementem działania algorytmu jest dobór początkowej wartości parametru T. Powinna być ona wystarczająco wysoka, by zapewnić akceptację niemal wszystkich przejść. W moim programie początkowa wartość T liczona była w następujący sposób:

Przyjęta została wartość prawdopodobieństwa P na poziomie 0.98, następnie dla losowej próby wyliczona została średnia różnica f(x) – f(x0), która dalej figurować będzie pod nazwą delta_f. Następnie T zostało obliczone ze wzoru:

$$T_o = -\frac{\Delta f}{ln(P)}$$

Schładzanie, czyli stopniowe obniżanie parametru T odbywało się po każdej iteracji według następującego schematu: T(i+1) = a * T(i), gdzie „a" było parametrem wybieranym przez użytkownika.

2. Opis najważniejszych klas w projekcie

Za działanie programu odpowiedzialne są 3 klasy: AdjacencyMatrix.cpp, PEA_zad2.cpp oraz Interface.cpp. Klasa Interface.cpp zawiera jedynie menu, które wyświetlanie jest w trakcie działanie programu, stąd nie będzie ona omówiona.

2.1. Klasa PEA_zad2.cpp

Jest to klasa, w której znajduje się funkcja main(). Odpowiedzialna jest ona za sterowanie programem. Służy za komunikację z użytkownikiem oraz z jej poziomu wywoływane są metody zdefiniowane w innych klasach. Główną część klasy mian stanowi instrukcja typu switch – case, która steruje działaniem programu:

```
switch (choice)
{
	case 1:
		cout << "Podaj nazwe pliku" << endl;
		cin >> fileName;
		isRight = matrix->load_matrix(fileName);
		if (!isRight)
		cout << "Blad wczytania pliku" << endl;
		break;
	case 2:
		test = rand() % 100 * 0.01;
		cout << test << endl;
		break;
	case 3:
		break;
	case 4:
		matrix->algorytm_zachalnny();
		matrix->symulowane_wyzarzanie();
		break;
	case 5:
		//miejsce na tabu search
		break;
	case 6:
		isRight = matrix->show_matrix();
		if (!isRight)
			cout << "Macierz nie istnieje" << endl;
```

```
            break;
        case 7:
            cout << "Wprowadzi kryterium stopu:" << endl;
            cin>>matrix->kryterium_stopu;
            matrix->kryterium_stopu = matrix->kryterium_stopu * 1000000000; //w
nanosekundach
            break;
        case 8:
            cout << "Wprowadz wspolczynnik zmiany temperatury" << endl;
            cin >> matrix->a;
            break;
        case 9:
            return 0;
            break;
        default:
            cout << "Niepoprawny znak" << endl;
            break;
    }
```

Ważnym elementem klasy PEA_zad2.cpp są również dwie metody odpowiedzialne za mierzenie czasów działania poszczególnych algorytmów:

```
double PCFreq = 0.0;
__int64 CounterStart = 0;
void StartCounter()
{
    LARGE_INTEGER li;
    if (!QueryPerformanceFrequency(&li))
        cout << "QueryPerformanceFrequency failed!" << endl;

    PCFreq = double(li.QuadPart) / 1000000000.0;    //nanosekundy

    QueryPerformanceCounter(&li);
    CounterStart = li.QuadPart;
    }
double GetCounter()
{
    LARGE_INTEGER li;
    QueryPerformanceCounter(&li);
    return double(li.QuadPart - CounterStart) / PCFreq;
}
```

2.2. Klasa AdjacencyMatrix.cpp

Klasa, która stanowi najważniejszy element programu. Przechowuje ona macierz reprezentującą graf oraz posiada metody potrzebne do wykonania algorytmów.

Najważniejsze metody klasy AdjacencyMatrix.cpp to:

load_matrix() – metoda odpowiedzialna za wczytanie macierzy z pliku .atsp. Metoda ta wczytuje dane z pliku informujące o rozmiarze grafu, alokuje potrzebną pamięć, a następnie odpowiednim elementom macierzy przypisuje wagi krawędzi opisane w pliku. W przypadku jeżeli w pamięci istnieje już macierz reprezentująca jakiś graf, metoda najpierw tą pamięć zwolni.

```
bool AdjacencyMatrix::load_matrix(string fileName)
{
	if (n != 0)
	{
		for (int i = 0; i < n; i++)
			delete A[i];
		delete[] A;
	}
	else
	{
		fstream file;
		file.open(fileName.c_str(), ios::in);

		if (file.good() == false)
			return false;

		string help;
		for (int i = 0; i < 9; i++)
			file >> help;
		file >> n;
		for (int i = 0; i < 5; i++)
			file >> help;
		A = new long int* [n];
		for (int i = 0; i < n; i++)
		{
			A[i] = new long int[n];
		}
		for (int i = 0; i < n; i++)
		{
			for (int j = 0; j < n; j++)
			{
				file >> A[i][j];
			}
		}

	}
	return true;
}
```

show_matrix() – metoda odpowiedzialna za wyświetlenie macierzy sąsiedztwa na ekranie. Metoda ta nie jest konieczna w projekcie. Jednak została ona dodana w celu sprawdzenia poprawności wczytanych grafów.

```
bool AdjacencyMatrix::show_matrix()
{
	if (n == 0)
		return false;
	else
	{
		cout << "     ";
		for (int i = 1; i < n + 1; i++)
			printf("%5d", i);

		cout << endl;
		for (int i = 0; i < n; i++)
		{
			printf("%5d", i + 1);
			for (int j = 0; j < n; j++)
			{
				printf("%5d", A[i][j]);
			}
			cout << endl;
```

```
            }

        }
        return true;
}
```

algorytm_zachlanny() – metoda odpowiedzialna za algorytm zachłanny, który służył do wyznaczenia rozwiązania poczatkowanego, stosowanego dalej w algorytmie symulowanego wyżarzania. Metoda zachłanna za każdym razem wybiera wierzchołek najmniej oddalony od wierzchołka, w którym aktualnie się znajdujemy. Dodatkowo metoda inicjuje zmienne wykorzystywane następnie w algorytmie SW. Dzieje się tak, ponieważ metoda ta jest wywoływana za każdym razem przed wywołaniem metody odpowiedzialnej za symulowane wyżarzanie. Z racji na dużą ilość kodu, nie zostanie on w całości wklejony do sprawozdania.

Część metody odpowiedzialna za wybór ścieżki

```
while (ilosc != n)
{
        for (int i = 1; i < n; i++)
        {
                if (A[v][i] < min && odwiedzone[i] == false && v != i)
                {
                        min = A[v][i];
                        wierzcholek = i;
                }
        }
        odwiedzone[wierzcholek] = true;
        suma += A[v][wierzcholek];
        sciezka[indeks] = wierzcholek;
        indeks++;
        ilosc++;
        min = 2147483647;
        v = wierzcholek;

}

suma += A[v][0]
```

sumulowane_wyzarzanie() – najważniejsza metoda w projekcie. Odpowiedzialna za wykonanie algorytmu symulowanego wyżarzania. Metoda ta jest podzielona na kilka części:

```
xo = new int[n];
x = new int[n];
int v1;
int v2;
for (int i = 0; i < n; i++)
{
        xo[i] = sciezka1[i];
        x[i] = sciezka1[i];
}
sumaxo = suma1;
sumax = 0;
```

Powyższa część metody odpowiedzialna jest za inicjalizacje oraz alokacje zmiennych koniecznych do działania algorytmu.

```
for (int i = 0; i < 100; i++)
	{
		v1 = rand() % (n - 1) + 1;
		do
		{
			v2 = rand() % (n - 1) + 1;
		} while (v1 == v2);

		swap(pomocx[v1], pomocx[v2]);
		for (int i = 0; i < n - 1; i++)
		{
			suma_pomocx += A[pomocx[i]][pomocx[i + 1]];
		}
		srednia += (suma_pomocxo - suma_pomocx);
		suma_pomocx = 0;
		swap(pomocx[v1], pomocx[v2]);
	}
	srednia = srednia / 100;
	T = (srednia / log(0.98));
	delete[] pomocx;
```

Powyższa część odpowiedzialna jest za obliczenie temperatury początkowej zgodnie ze wzorem podanym w 1. punkcie sprawozdania.

```
StartCounter();
while (GetCounter()  < kryterium_stopu) // warunek zatrzymaniaw
{
	v1 = rand() % (n-1) + 1; //wybierz w sposób losowy
	do
	{
		v2 = rand() % (n-1) + 1;
	} while (v1 == v2);
	swap(x[v1], x[v2]);
	for (int i = 0; i < (n - 1); i++)
	{
		sumax += A[x[i]][x[i + 1]];
	}
	sumax += A[x[n - 1]][0]; //dlugosc nowego rozwiazania
	if (rand() % 101 * 0.01 < probability())
	{
		sumaxo = sumax;
		for (int i = 0; i < n; i++)
			xo[i] = x[i];
			if (sumaxo < suma1)
			{
				suma1 = sumaxo;
				for (int i = 0; i < n; i++)
				{
					sciezka1[i] = xo[i];
				}
			}
	}
	T = T * a;
	sumax = 0;
	for (int i = 0; i < n; i++)
		x[i] = xo[i];
}
```

Powyższa część odpowiedzialna jest za wykonanie SW. Na początku w warunku pętli while sprawdzane jest czy nie przekroczyliśmy zadanego wcześniej czasu. Następnie wybierane są losowo dwa wierzchołki, które zamieniamy w ścieżce. Obliczana jest suma drogi rozważanego rozwiązania, a później na podstawie obliczonego prawdopodobieństwa algorytm wybiera czy nowe rozwiązanie zostanie przyjęte, czy nie. Jeżeli tak, to następnie sprawdzane jest czy nowe rozwiązanie jest lepsze od najlepszego dotychczas znanego. W przypadku spełnienia warunku, najlepsze rozwiązanie jest aktualizowane. Na koniec pętli while następuje „schładzanie" temperatury.

probability() – jest to metoda pomocnicza wykorzystywana w metodzie symulowane_wyzarzanie(). Jest ona odpowiedzialna za implementację wzoru:

$$P(\boldsymbol{x}_0, \boldsymbol{x}) = \min\{1, \exp\,(-\frac{f(\boldsymbol{x}) - f(\boldsymbol{x}_0)}{T_i})\},$$

3. Pomiary i wyniki

Poniższa tabela zawiera najlepsze rozwiązanie oraz jego znalezienia dla różnych czasów dla różnych plików. Wartość współczynnika alfa dla każdego uruchomienia wynosiła 0,9999999.

Nazwa pliku	Czas algorytmu[s]	Najlepszy wynik	Moment znalezienia[s]
ftv47.atsp	120	1807	10
ftv47.atsp	240	1890	20
ftv47.atsp	540	1899	30
ftv170.atsp	120	3701	50
ftv170.atsp	240	3625	80
ftv170.atsp	540	3721	50
rbg403.atsp	120	2481	110
rbg403.atsp	240	2473	110
rbg403.atsp	540	2467	120

Tabela 1. Najlepsze rozwiązanie i moment jego znalezienia dla różnych plików

Poniżej zamieszczone są wykresu błędu funkcji czasu dla uśrednionych wyników dla każdego z plików. W każdym przypadku współczynnik alfa jest równy 0.9999999.

Błąd względny liczony jest ze wzoru: $|f_{zn} - f_{opt}|/f_{opt}$, gdzie fzn oznacza wartość obliczoną przez nasz algorytm, natomiast fopt najlepsze znane rozwiązanie.

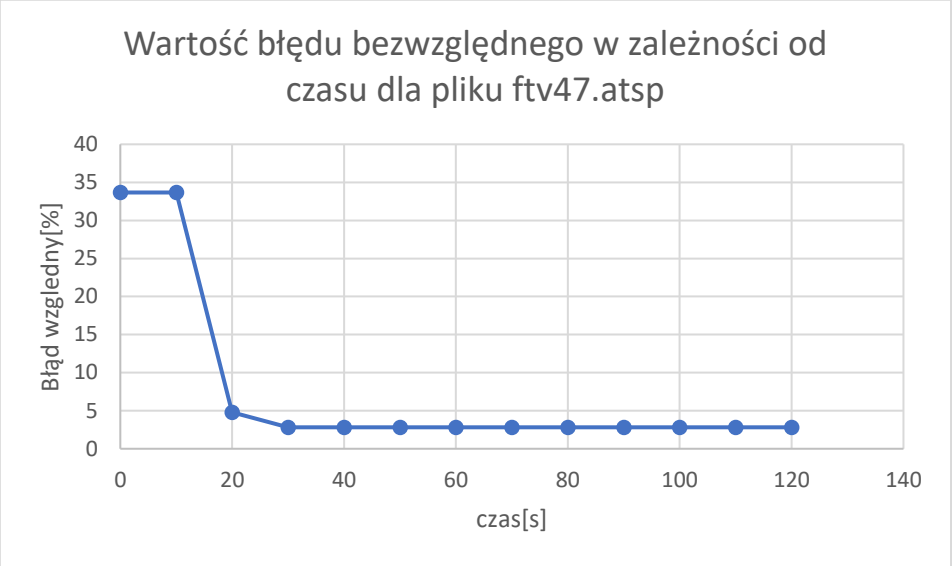

Tabela 2. Wykres błędu bezwzględnego od czasu dla pliku ftv47.atsp

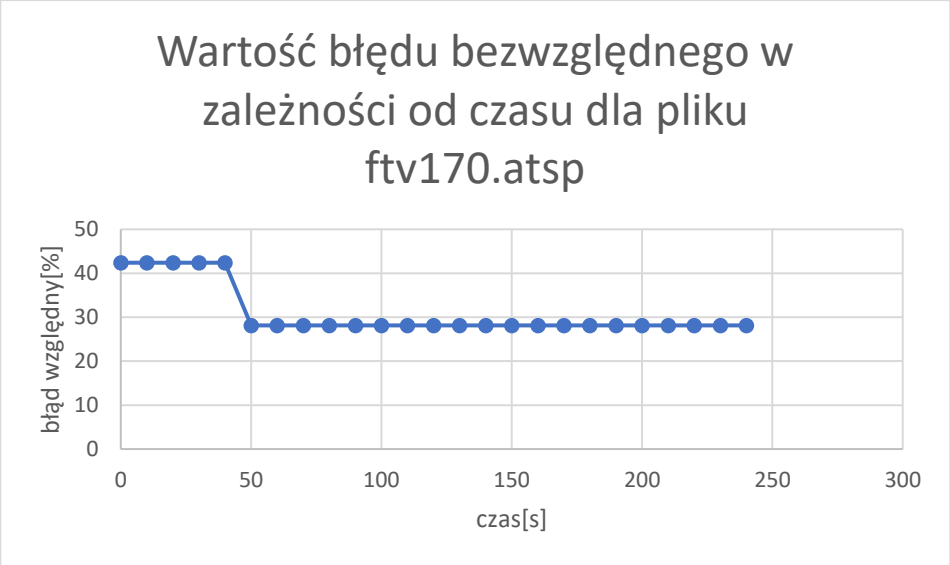

Tabela 3. Wykres błędu bezwzględnego od czasu dla pliku ftv170.atsp

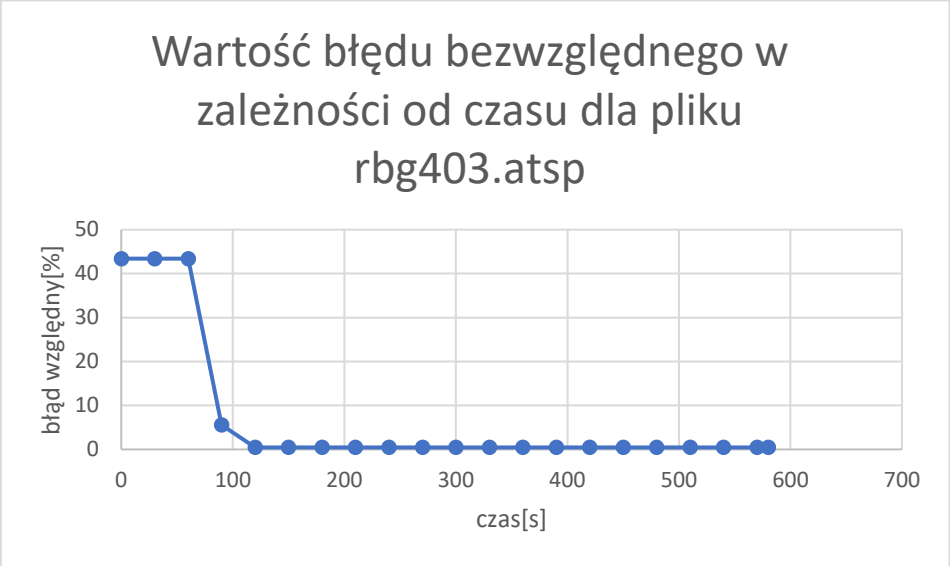

Tabela 4. Wykres błędu bezwzględnego w zależności od czasu dla pliku rbg403.atsp

4. Najlepsze rozwiązania

ftv47.atsp:

```
temperatura poczatkowa: 11137.1
czas: 1e+10, rozwiazanie: 2374
czas: 2e+10, rozwiazanie: 1864
czas: 3e+10, rozwiazanie: 1837
czas: 4e+10, rozwiazanie: 1837
Temperatura ko˜cowa: 1.09494e-06
Suma: 1837
Sciezka: 0 -> 25 -> 47 -> 41 -> 43 -> 42 -> 22 -> 40 -> 20 -> 37 -> 38 -> 18 -> 17 -> 12 -> 32 -> 7 -> 23 -> 34 -> 13 ->
46 -> 36 -> 35 -> 14 -> 15 -> 16 -> 45 -> 39 -> 19 -> 44 -> 21 -> 26 -> 1 -> 10 -> 11 -> 8 -> 9 -> 33 -> 27 -> 2 -> 28
-> 3 -> 24 -> 4 -> 29 -> 30 -> 31 -> 5 -> 6 ->
Wybierz co chcesz zrobic:
```

ftv170.atsp:

```
Suma: 3530
Sciezka: 0 -> 110 -> 108 -> 166 -> 83 -> 84 -> 69 -> 65 -> 88 -> 153 -> 87 -> 67 -> 167 -> 70 -> 85 -> 86 -> 93 -> 92 -> 154 -> 89 -> 90 -> 91 -> 94 -> 96 -> 165 -> 104 -> 114 -> 109 -> 107 -> 106 -> 105 -> 97 -
> 100 -> 101 -> 123 -> 162 -> 102 -> 103 -> 163 -> 99 -> 98 -> 95 -> 127 -> 126 -> 125 -> 129 -> 146 -> 145 -> 144 -> 143 -> 149 -> 148 -> 147 -> 137 -> 136 -> 138 -> 139 -> 140 -> 141 -> 113 -> 164 -> 135 -> 13
4 -> 6 -> 142 -> 152 -> 14 -> 13 -> 21 -> 24 -> 15 -> 159 -> 16 -> 17 -> 20 -> 158 -> 32 -> 36 -> 37 -> 75 -> 10 -> 76 -> 74 -> 11 -> 12 -> 18 -> 19 -> 38 -> 39 -> 40 -> 46 -> 47 -> 48 -> 49 -> 73 -> 170 -> 168
-> 78 -> 82 -> 79 -> 80 -> 3 -> 5 -> 133 -> 169 -> 112 -> 115 -> 116 -> 117 -> 118 -> 119 -> 120 -> 122 -> 121 -> 124 -> 128 -> 130 -> 131 -> 132 -> 111 -> 77 -> 72 -> 71 -> 68 -> 63 -> 56 -> 57 -> 62 -> 61 -> 6
6 -> 64 -> 55 -> 54 -> 58 -> 59 -> 60 -> 50 -> 51 -> 52 -> 53 -> 43 -> 44 -> 45 -> 42 -> 41 -> 155 -> 156 -> 34 -> 35 -> 157 -> 33 -> 31 -> 30 -> 28 -> 27 -> 29 -> 22 -> 23 -> 26 -> 25 -> 150 -> 161 -> 160 -> 15
1 -> 7 -> 8 -> 4 -> 9 -> 2 -> 1 -> 81 ->
```

rbg403.atsp:

```
Suma: 2476
Sciezka: 0 -> 159 -> 7 -> 134 -> 288 -> 179 -> 368 -> 170 -> 284 -> 343 -> 81 -> 22 -> 21 -> 253 -> 206 -> 30 -> 68 -> 233 -> 324 -> 212 -> 64 -> 15 -> 397 -> 5 -> 190 -> 100 -> 44 -> 194 -> 380 -> 362 -> 307 ->
20 -> 133 -> 265 -> 275 -> 147 -> 193 -> 161 -> 148 -> 154 -> 331 -> 378 -> 18 -> 217 -> 84 -> 293 -> 124 -> 269 -> 162 -> 48 -> 402 -> 287 -> 46 -> 211 -> 183 -> 17 -> 12 -> 333 -> 267 -> 357 -> 358 -> 257 ->
106 -> 359 -> 286 -> 74 -> 52 -> 40 -> 263 -> 139 -> 4 -> 240 -> 219 -> 330 -> 290 -> 169 -> 27 -> 111 -> 369 -> 38 -> 131 -> 355 -> 216 -> 200 -> 198 -> 246 -> 360 -> 96 -> 66 -> 60 -> 98 -> 99 -> 389 -> 105 ->
19 -> 57 -> 182 -> 361 -> 146 -> 144 -> 174 -> 185 -> 375 -> 191 -> 187 -> 130 -> 172 -> 26 -> 186 -> 53 -> 115 -> 94 -> 128 -> 255 -> 239 -> 117 -> 151 -> 11 -> 316 -> 256 -> 374 -> 363 -> 379 -> 79 -> 62 -> 1
3 -> 243 -> 176 -> 156 -> 210 -> 308 -> 394 -> 225 -> 283 -> 277 -> 352 -> 209 -> 41 -> 137 -> 332 -> 37 -> 262 -> 261 -> 171 -> 63 -> 297 -> 136 -> 388 -> 348 -> 329 -> 294 -> 370 -> 328 -> 327 -> 33 -> 376 ->
318 -> 218 -> 203 -> 336 -> 292 -> 108 -> 120 -> 116 -> 109 -> 301 -> 249 -> 82 -> 247 -> 268 -> 16 -> 47 -> 112 -> 391 -> 258 -> 232 -> 132 -> 306 -> 319 -> 264 -> 274 -> 58 -> 8 -> 344 -> 43 -> 349 -> 305 -> 2
96 -> 295 -> 201 -> 118 -> 149 -> 347 -> 341 -> 23 -> 393 -> 178 -> 138 -> 236 -> 227 -> 320 -> 207 -> 45 -> 241 -> 304 -> 260 -> 160 -> 365 -> 87 -> 56 -> 93 -> 145 -> 285 -> 282 -> 281 -> 180 -> 165 -> 387 ->
181 -> 346 -> 338 -> 102 -> 88 -> 69 -> 245 -> 166 -> 199 -> 95 -> 10 -> 221 -> 300 -> 86 -> 101 -> 71 -> 153 -> 75 -> 385 -> 67 -> 189 -> 345 -> 173 -> 400 -> 141 -> 392 -> 351 -> 125 -> 273 -> 395 -> 177 -> 31
3 -> 323 -> 354 -> 342 -> 122 -> 119 -> 29 -> 91 -> 235 -> 121 -> 2 -> 61 -> 350 -> 372 -> 356 -> 152 -> 76 -> 222 -> 250 -> 312 -> 35 -> 367 -> 310 -> 6 -> 208 -> 276 -> 259 -> 78 -> 226 -> 390 -> 175 -> 104 ->
9 -> 50 -> 311 -> 271 -> 272 -> 278 -> 377 -> 202 -> 32 -> 322 -> 317 -> 373 -> 163 -> 25 -> 135 -> 270 -> 238 -> 229 -> 14 -> 353 -> 39 -> 103 -> 158 -> 339 -> 77 -> 31 -> 73 -> 251 -> 113 -> 107 -> 386 -> 184
-> 129 -> 237 -> 90 -> 72 -> 337 -> 213 -> 28 -> 401 -> 214 -> 192 -> 114 -> 65 -> 254 -> 223 -> 309 -> 242 -> 70 -> 188 -> 299 -> 298 -> 291 -> 315 -> 85 -> 252 -> 371 -> 364 -> 303 -> 224 -> 248 -> 127 -> 399
-> 398 -> 396 -> 83 -> 167 -> 36 -> 234 -> 228 -> 266 -> 97 -> 195 -> 280 -> 279 -> 24 -> 54 -> 110 -> 326 -> 325 -> 157 -> 1 -> 230 -> 382 -> 381 -> 244 -> 80 -> 366 -> 321 -> 289 -> 302 -> 89 -> 231 -> 205 ->
204 -> 142 -> 314 -> 59 -> 42 -> 92 -> 51 -> 49 -> 215 -> 168 -> 126 -> 55 -> 340 -> 164 -> 3 -> 220 -> 197 -> 123 -> 150 -> 384 -> 383 -> 34 -> 140 -> 334 -> 155 -> 143 -> 335 -> 196 ->
```

5. Wnioski

Eksperyment można uznać za udany, ponieważ udało się znaleźć rozwiązania zbliżone do optymalnego(poza grafem z pliku ftv170.atsp). Najgorsze wyniki zostały uzyskane dla pliku średniego, gdzie błąd znalezionego rozwiązania wynosił około 28%. W porównaniu do algorytmów z poprzedniego zadania, nowe implementacje są znacznie mniej złożone, co pozwala na ich znacznie szybszą pracę, z tą różnicą, że znalezienie wyniku optymalnego jest niemal niemożliwe.