Arkadiusz Kozak 226122 **30.11.2017**

Prowadzący: **dr Inż. Dariusz Banasiak**

Algorytm komiwojażera oparty na metodzie symulowanego wyżarzania

1.Opis problemu

Zadanie polega na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Danych mamy ***N*** miast które komiwojażer ma odwiedzić oraz odległości między nimi. Celem jest znalezienie najkrótszej ścieżki łączącej wszystkie miasta i kończącej się w określonym punkcie.

2. Opis algorytmu

Algorytm symulowanego wyżarzania jest algorytmem wyszukującym przybliżone rozwiązanie konkretnej instancji problemu NP-trudnego. Jest to algorytm z dziedziny heurystyki, dzięki któremu jesteśmy w stanie znaleźć rozwiązanie które nie zawsze będzie najlepszym, jednak będzie w większości przypadków wystarczająco dobrym, a do tego rozwiązane rozstanie w rozsądnym czasie. Algorytmu metaheurystyczne działają za każdym razem inaczej, gdyż w swoim działaniu opierają się one na losowości. W symulowanym wyżarzaniu zezwalamy na zmianę rozwiązania z pewnym prawdopodobieństwem, nawet wtedy gdy jest ono gorsze, ponieważ możliwym jest że w sąsiedztwie gorszego będą lepsze niż te które mamy w aktualnym sąsiedztwie. Dzięki temu, możemy „unikać” problemów minimum lokalnego. Algorytm opiera się czterema głównymi parametrami :

- Temperaturą początkową **T**
- Temperaturą końcową T_{min}
- Funkcją prawdopodobieństwa **P**
- Funkcją obniżania temperatury **F**

Funkcja prawdopodobieństwa określa, z jakim prawdopodobieństwem będziemy zmieniać rozwiązania na gorsze. Jest ona zależna od aktualnego rozwiązania, wylosowanego nowego rozwiązania oraz temperatury. Jest to analogia do procesu metalurgicznego – dopóki temperatura jest wysoka, akceptujemy zmiany na rozwiązania początkowo gorsze, ale potencjalnie lepsze. Zakładamy że w miarę postępowania algorytmu i obniżania temperatury dojdziemy do rozwiązania „rozsądnego", więc wszelkie „ryzykowne" zamiany będą już zbędne. Z tego tez wynika funkcja obniżania temperatury. Przyjęte parametry poszczególnych funkcji:

$$P = e^{\frac{-(\Delta s)}{T}}$$

Gdzie:

e - stała Eulera

Δs – różnica drogi między aktualnym a wylosowanym rozwiązaniem

$$F = k * T_i$$

Gdzie:

k – pewna, ustalona odgórnie stała

T_i – aktualna temperatura

3. Przykład praktyczny

Tabela miast dla naszego przykładu:

\	0	1	2	3
0	∞	3	2	1
1	3	∞	5	2
2	3	1	∞	5
3	7	4	2	∞

Warunki poczatkowe:

Temperatura początkowa : 10

Temperatura końcowa : 2.5

Funkcja obniżania temperatury: 0.6 * T_i

K – aktualny najlepszy koszt

Losujemy rozwiązanie początkowe : 0-1-2-3-0 o koszcie 20.

*Podczas losowania nie zmieniamy położenia miasta początkowego

I iteracja programu: (T=10, K=20)

Losujemy nowe rozwiązanie, poprzez zamianę ze sobą dwóch losowych miast. Nasze nowe rozwiązanie: 0-1-3-2-0 koszt:10

Nowe rozwiązanie ma koszt mniejszy niż nasze aktualne rozwiązanie, zatem zamieniamy je. Zmniejszamy również temperaturę.

II iteracja programu (T=6, K=10)

Losujemy nowe rozwiązanie, poprzez zamianę ze sobą dwóch losowych miast. Nasze nowe rozwiązanie: 0-2-3-1-0 koszt:14

Nowe rozwiązanie ma większy koszt, sprawdzamy więc naszą funkcją prawdopodobieństwa. Wynosi ona w zaokrągleniu 0.51, sprawdzamy zatem czy wartość tej funkcji przewyższy losową liczbę z zakresu (0,1). Losowa liczba była mniejsza, a zatem przyjmujemy że rozwiązanie to jest perspektywiczne i przechodzimy do niego. Zmniejszamy temperaturę.

III iteracja programu (T=3.6, K=14)

Losujemy nowe rozwiązanie, poprzez zamianę ze sobą dwóch losowych miast. Nasze nowe rozwiązanie: 0-2-1-3-0 koszt:12

Nowe rozwiązanie ma koszt mniejszy niż nasze aktualne rozwiązanie, zatem zamieniamy je. Zmniejszamy również temperaturę.

W tym momencie aktualna temperatura spadła poniżej temperatury końcowej, a zatem kończymy algorytm. Finalnie naszą najlepszą ścieżką jest ścieżka 0-2-1-3-0 z kosztem 12. Nie jest to jednak optymalne rozwiązanie, gdyż najlepsze możliwe to ścieżka: 0-3-2-1-0 o koszcie 7.

4. Opis implementacji programu

Cały algorytm opiera się na przedstawionych poniższych polach:

```
private string filePath; // ścieżka do pliku
private List<int> currentOrder = new List<int>(); // aktualne rozwiązanie
private List<int> nextOrder = new List<int>(); // nowe rozwiązanie
private double[,] distances; // tabela miast
private Random random = new Random();
private double shortestDistance = 0; // najlepsza ścieżka
```

Oraz metodach:

```
public bool Included(int[] tab, int k)
```

Powyższa funkcja służy do sprawdzenia czy w danej tablicy ***tab*** znajduje się już wyraz o wartości ***k***.

```
private void LoadCities()
```

Powyższa funkcja służy do załadowania wszystkich miast z pliku do tablicy *distances* oraz wylosowania pierwszego rozwiązania(losowego) *currentOrder*.

```
private double GetTotalDistance(List<int> order)
```

Powyższa funkcja służy do zliczenia długości ścieżki w danym rozwiązaniu.

```
private List<int> GetNextArrangement(List<int> order)
```

Powyższa funkcja służy do zamiany dwóch losowych miast w rozwiązaniu *order* i zwróceniu nowej listy.

```
public void Anneal()
```

Powyższa funkcja, to główny punkt programu. Służy do wyżarzania przez określony czas danej instancji tablicy.

5. Testy:

Testy zostały przeprowadzone dla 3 instancji miast, pobranych z biblioteki TSPLIB :

gr17 - bays29 - berlin52

Dla następujących temperatur początkowych:

100 (kolor niebieski na wykresie)

$10 * 10^5$ (kolor pomarańczowy na wykresie)

$10 * 10^{10}$ (kolor szary na wykresie)

Oraz dla następujących funkcji obniżania temperatury:

$$F_1 = 0.99 * T_i$$

$$F_2 = 0.9999 * T_i$$

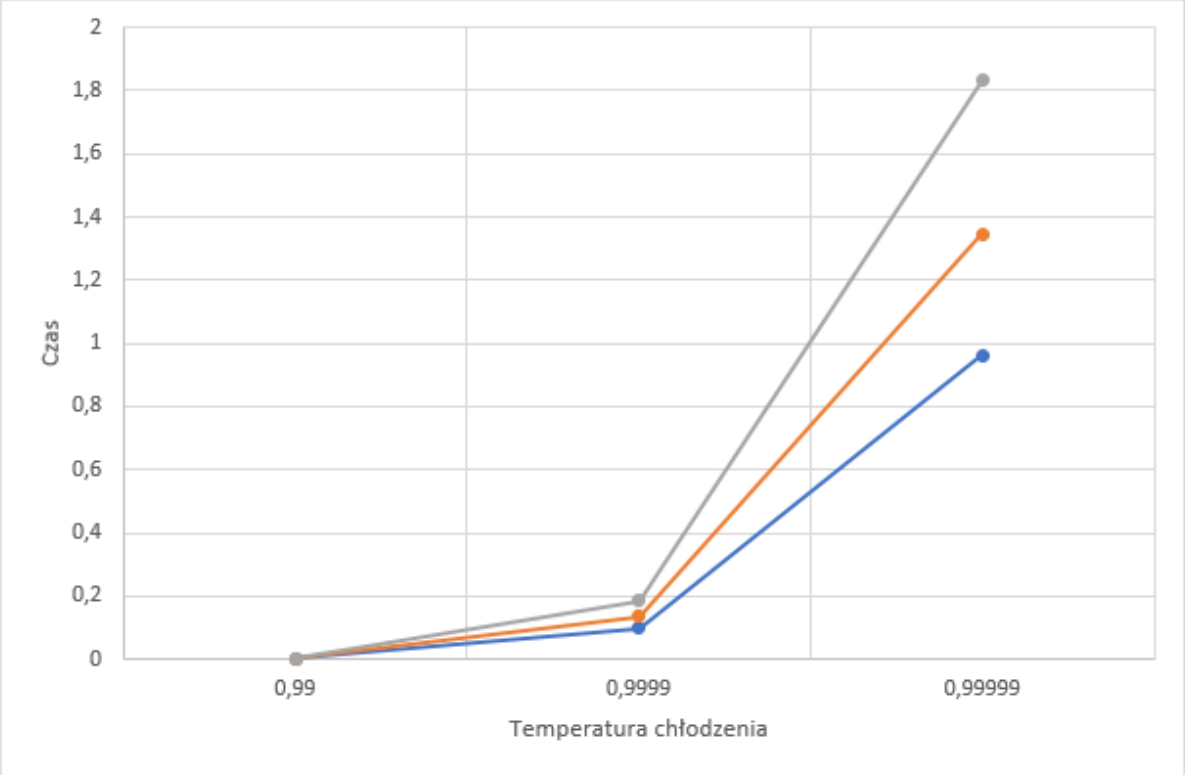
$$F_3 = 0.99999 * T_i$$

Temperatura minimalna jest stała i wynosi 0.000000001.

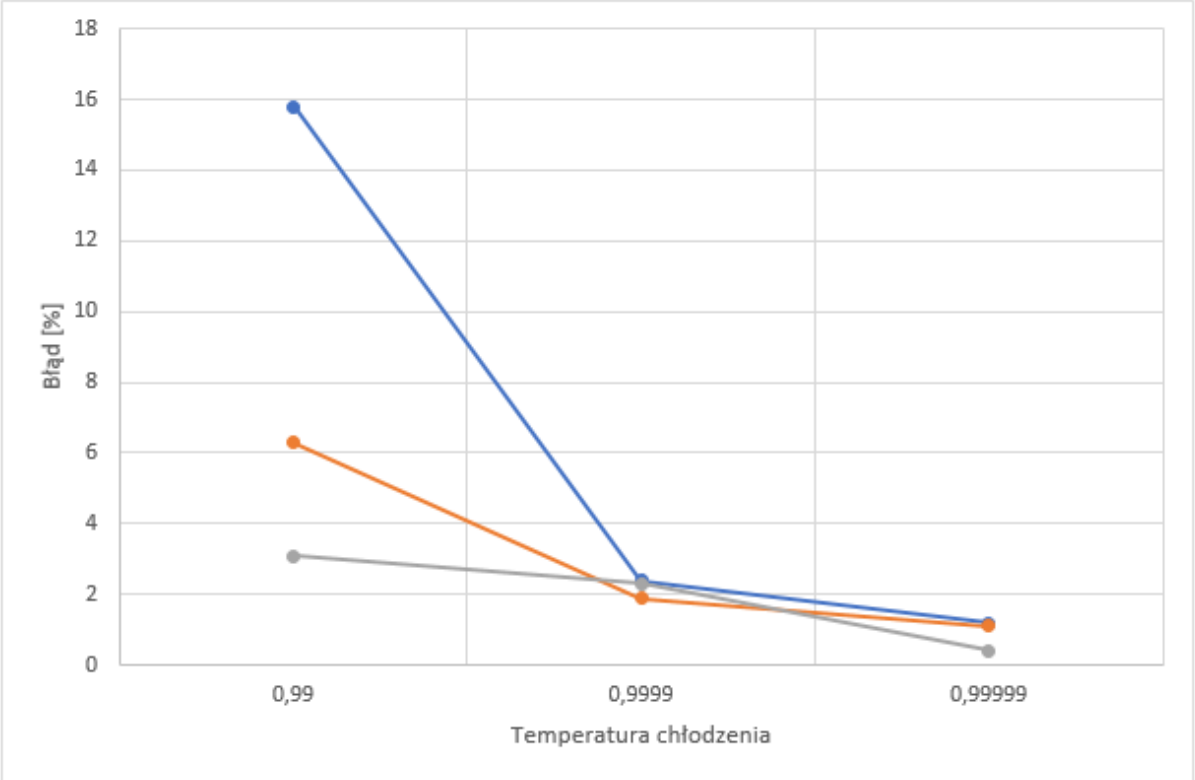
Każdy test został powtórzony 20 razy. A następnie wyciągnięta została z niego średnia oraz mediana.

1) Plik gr17, 17 miast, najkrótsza ścieżka 2085

Temp \ Cooling	F_1	F_2	F_3
CZASY[s]			
$10 * 10^1$	0.002	0.097	0.963
$10 * 10^5$	0.002	0.135	1.346
$10 * 10^{10}$	0.003	0.185	1.831
Średnia długość ścieżki			
$10 * 10^1$	2358	2136	2111
$10 * 10^5$	2213	2126	2109
$10 * 10^{10}$	2164	2135	2095
Mediana ze wszystkich ścieżek			
$10 * 10^1$	2398	2123	2103
$10 * 10^5$	2184	2123	2103
$10 * 10^{10}$	2167	2120	2090
Błąd %			
$10 * 10^1$	15.8	2.4	1.2
$10 * 10^5$	6.3	1.9	1.1
$10 * 10^{10}$	3.1	2.3	0.4



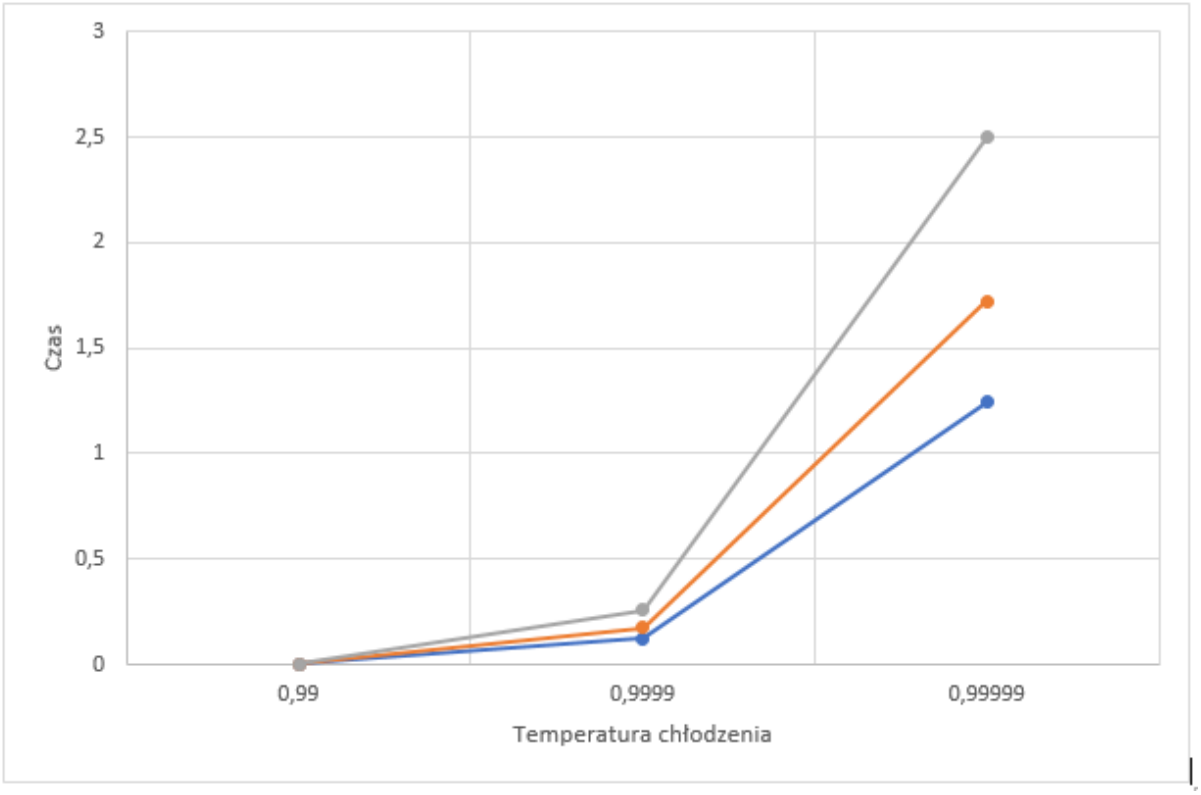
Wykres zależności czasu od temperatury chłodzenia



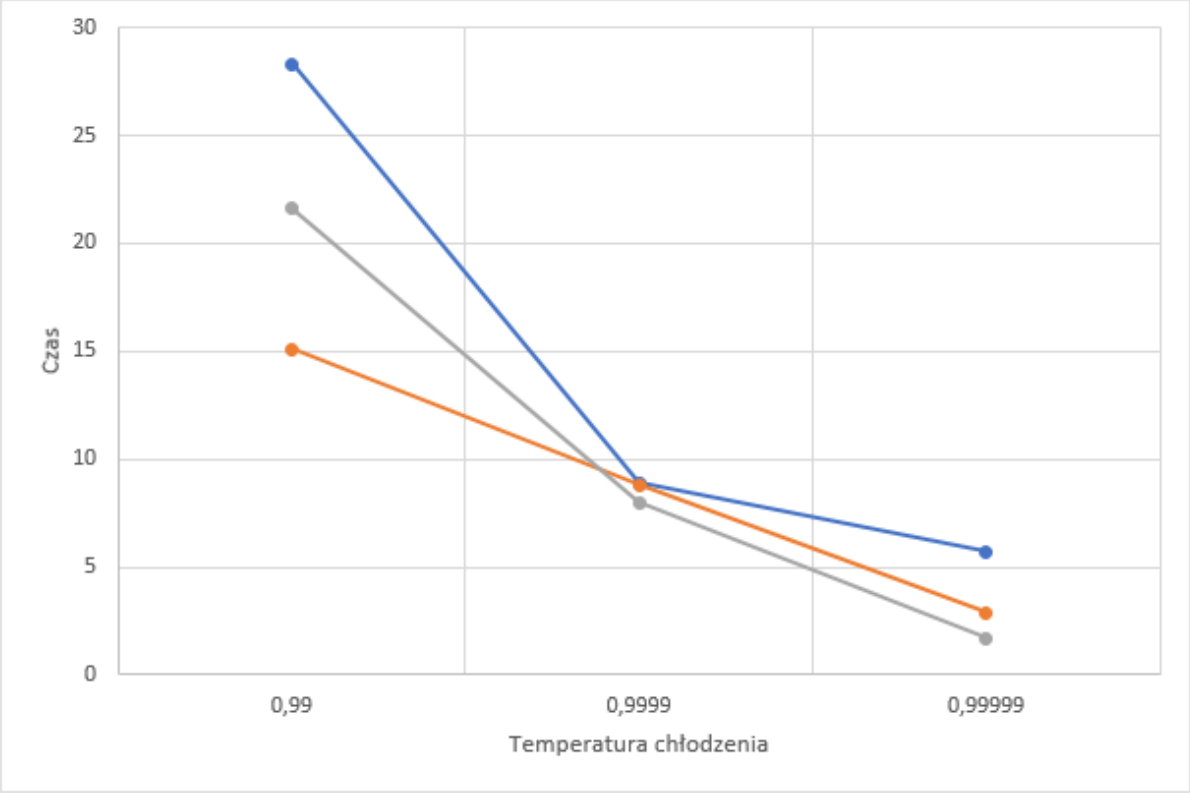
Wykres zależności błędu od temperatury chłodzenia

2) Plik bays29, 29 miast, najkrótsza ścieżka 2020

Temp \ Cooling	F_1	F_2	F_3
CZASY[s]			
$10 * 10^1$	0.002	0.123	1.242
$10 * 10^5$	0.003	0.175	1.723
$10 * 10^{10}$	0.004	0.258	2.497
Średnia długość ścieżki			
$10 * 10^1$	2597	2201	2137
$10 * 10^5$	2327	2198	2079
$10 * 10^{10}$	2448	2183	2056
Mediana ze wszystkich ścieżek			
$10 * 10^1$	2634	2211	2126
$10 * 10^5$	2282	2251	2060
$10 * 10^{10}$	2495	2192	2083
Błąd %			
$10 * 10^1$	28.3	8.9	5.7
$10 * 10^5$	15.1	8.8	2.9
$10 * 10^{10}$	21.6	8.0	1.7



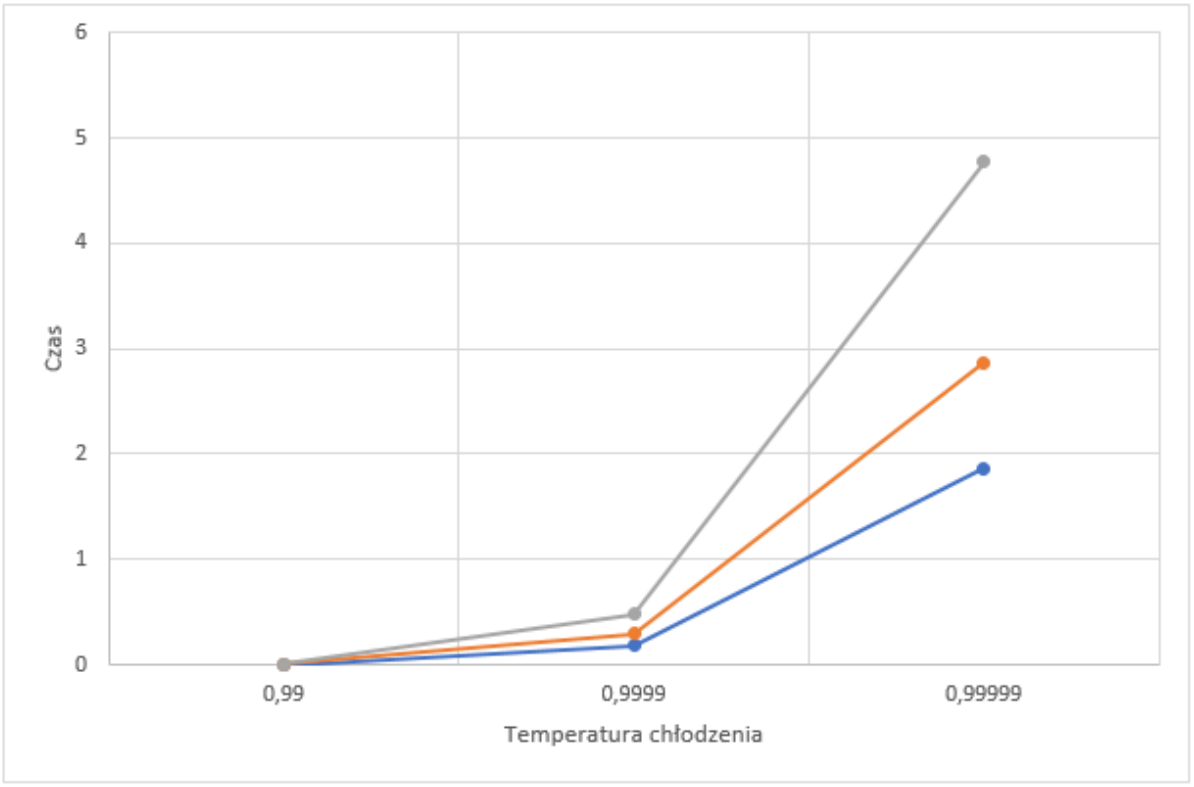
Wykres zależności czasu od temperatury chłodzenia



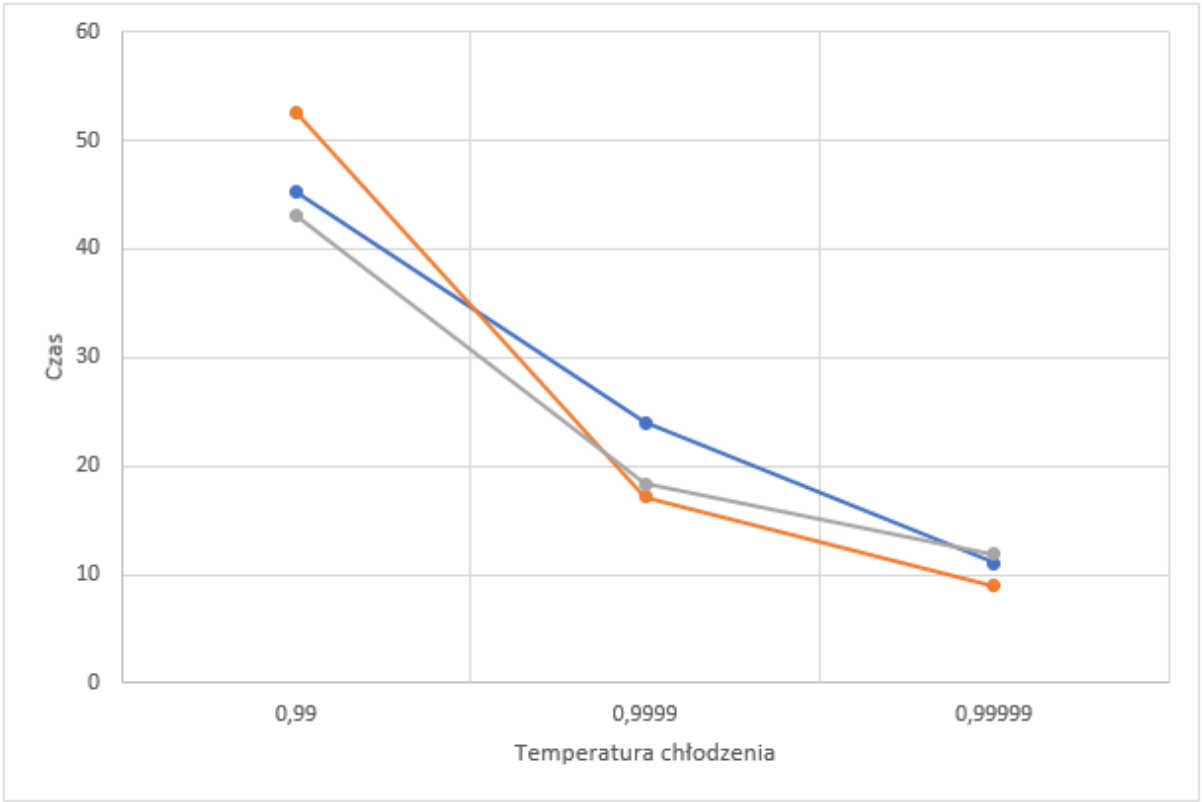
Wykres zależności błędu od temperatury chłodzenia

3) Plik berlin52, 52 miasta, najkrótsza ścieżka 7544

Temp \ Cooling	F_1	F_2	F_3
CZASY[s]			
$10 * 10^1$	0.003	0.180	1.863
$10 * 10^5$	0.004	0.291	2.871
$10 * 10^{10}$	0.006	0.485	4.772
Średnia długość ścieżki			
$10 * 10^1$	11040	9362	8429
$10 * 10^5$	11477	8887	8227
$10 * 10^{10}$	10845	8922	8447
Mediana ze wszystkich ścieżek			
$10 * 10^1$	10917	9652	8425
$10 * 10^5$	11485	9034	8265
$10 * 10^{10}$	11117	8965	8453
Błąd %			
$10 * 10^1$	45.2	24	11.1
$10 * 10^5$	52.5	17.2	9
$10 * 10^{10}$	43	18.4	11.9



Wykres zależności czasu od temperatury chłodzenia



Wykres zależności błędu od temperatury chłodzenia

6. Wnioski

Przygotowanie projektu pozwoliło na dobre zapoznanie się z tematyką algorytmiki metaheurystycznej. Niestety testowanie tego programu było dosyć ciężkie ze względu na zaimplementowany element losowości. Mimo to algorytm radzi sobie dobrze nawet z dużymi instancjami problemów, chociaż jak widać, im instancja większa tym margines błędu staje się większy. Można również zauważyć że największy wpływ na działanie algorytmu ma funkcja obniżająca temperaturę, znacznie większy niż początkowa wartość temperatury. Dlatego aby uzyskać możliwie bliski rozwiązaniu optymalnemu wynik, warto ustawić te wartość na możliwie bliską liczbie 1, tak aby iteracji było jak najwięcej. Ponadto samo zwiększanie temperatury nie daje tak spektakularnych rezultatów ze względu na fakt iż przy wysokiej temperaturze mamy wyższa szansę na wybranie gorszego rozwiązania, którego sąsiedztwo nie koniecznie może być lepsze. Dlatego też optymalnym wyjściem jest temperatura rzędu $10 * 10^5$ oraz funkcja równa +/- 0.999999, co pozwala w rozsądnym czasie na uzyskanie wyników zbliżonych do rozwiązania właściwego.