Projektowanie Efektywnych Algorytmów

Kierunek *Informatyka*	**Termin** *Czwartek 17:05*
Temat *Symulowane wyżarzanie*	**Problem** *TSP*
Autor *229218 Michał Honc*	**Nr grupy** -
Prowadzący *Mgr inż. Radosław Idzikowski*	**data** *11 grudnia 2018*

1 Opis problemu

Problem Komiwojażera (Travelling Salesman Problem), W problemie mamy n miast oraz macierz nxn przejść pomiędzy miastami. Za zadanie mamy znaleźć jak najkrótszą ścieżkę przechodzącą przez wszystkie miasta wliczając w to powrót do miasta początkowego.

2 Metoda rozwiązania

2.1 Symulowane wyżarzanie

Algorytm symulowanego wyżarzania jest algorytmem wyszukującym przybliżone rozwiązanie konkretnej instancji jakiegoś problemu NP-trudnego. Wykorzystując algorytmy przybliżone jesteśmy w stanie znaleźć rozwiązanie, które może i nie będzie najlepszym (optymalnym), jednak będzie „wystarczająco dobrym", a do tego zostanie ono znalezione w czasie w miarę rozsądnym. Algorytmy metaheurystyczne zawsze działają inaczej, ponieważ w swoim działaniu opierają się na losowości. Zdefiniujemy dla tego algorytmu otoczenie rozwiązania jako rozwiązanie mu sąsiadujące na przykład przez zamieniony jeden element. Czyli sąsiedztwem permutacji $3, 7, 1, 2, 4, 5, 6$ może być permutacja $3, 7, 1, 4, 2, 5, 6$. Pierwszym etapem powinno być, by algorytm dostawał „jakieś" rozwiązanie początkowe (chociażby wylosowane) i działał tak długo, aż w sąsiedztwie nie znajdują się żadne lepsze rozwiązania. Następnie zezwalamy na zmianę rozwiązania z pewnym prawdopodobieństwem, nawet wtedy, gdy jest ono gorsze – bo możliwe, że w sąsiedztwie gorszego będą lepsze niż te, które mamy w sąsiedztwie rozwiązania aktualnie rozpatrywanego. Dzięki temu symulowane wyżarzanie potrafi „unikać" problemów minimum lokalnego. Prezentowany algorytm cechuje się następującymi przyjętymi parametrami: początkową temperaturą T(10000), końcową temperaturą Tmin(0,0001), funkcją prawdopodobieństwa P oraz temperaturą chłodzenia(0,9999) lub (0,99999).

Listing 1: Symulowane wyżarzanie

```
1  void Simulated_annealing::znajdz_rozwiazanie()
2  {
3      while (obecna_temperatura > temperatura_minimalna)
4      {
5          for (int j = 0; j < liczba_iteracji; j++)
6          {
7              losowa_zamiana(); //zamiana losowa dwoch miast
8              dlugosc_obecnej_trasy = dlugosc_trasy(obecna_trasa);
9
10             if (dlugosc_obecnej_trasy < dlugosc_najlepszej_trasy
11                 // pozwalamy na zmiane na gorsze z pewnym prawdopodobienstwem w
12                 // w nadzieii na uzyskanie lepszego rozwiazania
13                 // w nowym sasiedztwie, ma to na celu unikniecie minimum lokalnego,
14                 // wraz z kolejnymi iteracjami
15                 //ponizsze kryterium zostaje spelnione coraz rzadziej
16                 || ((double)rand() / (double)RAND_MAX) < prawdopodobienstwo())
17             {
18                 najlepsza_trasa = obecna_trasa;
19                 dlugosc_najlepszej_trasy = dlugosc_obecnej_trasy;
20             }
21         }
22         obecna_temperatura *= temperatura_chlodzenia;
23     }
24 }
```

Funkcja prawdopodobieństwa określa, z jakim prawdopodobieństwem będziemy zmieniać rozwiązanie na gorsze. Funkcja ta jest zależna od rozwiązania nowego, starego i aktualnej temperatury. Z czasem prawdopodobieństwo zmiany będzie coraz mniejsze, ponieważ w miarę postępowania algorytmu i obniżania temperatury dojdziemy do rozwiązania „w miarę dobrego", tak więc „ryzykowne" zmiany na rozwiązania początkowo gorsze nie będą już potrzebne.

Listing 2: Funkcja prawdopodobieństwa

```
1 double Simulated_annealing::prawdopodobienstwo()
2 {
3     double potega = -((dlugosc_obecnej_trasy - dlugosc_najlepszej_trasy)
4     / obecna_temperatura);
5     return pow(M_E, potega);
6 }
```

3 Eksperymenty obliczeniowe

Obliczenia zastały wykonane na komputerze klasy PC z procesorem Intel Pentium, kartą graficzną zintegrowaną, 8GB RAM i DYSK SSD. Algorytmy operują na zmiennych integer 32bit. Pomiar czasu wykonany jest w mikrosekundach.

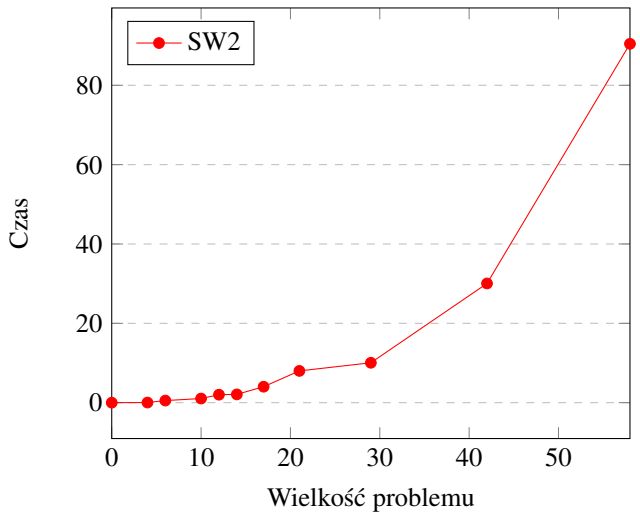
Wszystkie wyniki zebrano i przedstawiono w tabeli nr 1 gdzie:

- n - liczba miast,
- D - koszt najkrótszej ścieżki,
- $D1$ - koszt najkrótszej ścieżki znalezionej przez algorytm SW dla dla tempreratury chłodzenia 0,99999,
- $D2$ - koszt najkrótszej ścieżki znalezionej przez algorytm SW dla dla tempreratury chłodzenia 0,9999,
- $DP[us]$ - czas dla algorytmu programowania dynamicznego
- $SW1[us]$ - czas dla algorytmu symulowanego wyżarzania dla tempreratury chłodzenia 0,99999
- $SW2[us]$ - czas dla algorytmu symulowanego wyżarzania dla tempreratury chłodzenia 0,9999

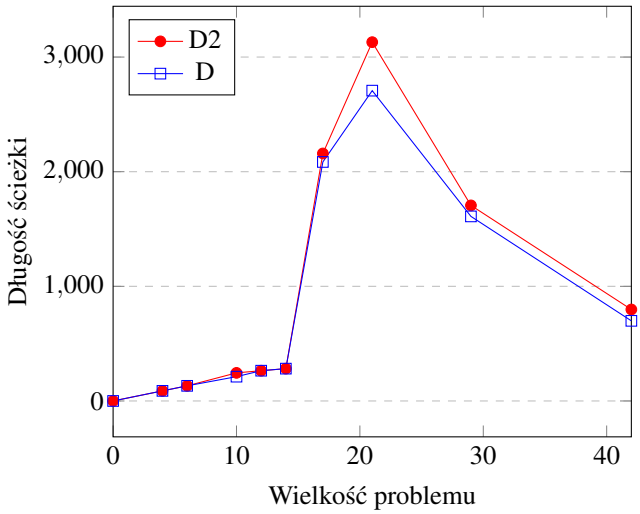
Wnioski - algorytm symulowanego wyżarzania sprawdza się znakomicie dla TSP dla dużych instancji(>20), przy stosunkowo niedużym błędzie, gdzie czas działania algorytmów dokładnych jest zbyt duży. Czas działania algorytmu SW zależy głównie od temperatury początkowej, końcowej, chłodzenia oraz od ilości iteracji dla danej temperatury.

Tablica 1: Czas obliczeń oraz błąd dla ustalonej liczby miast.

n	$DP[us]$	$SW1[us]$	$SW2[us]$	D	$D1$	$D2$
0	0	0	0	0	0	0
4	1.34×10^2	2.31316×10^6	2.90925×10^5	88	88	88
6	2.12×10^2	4.90655×10^6	5.26230×10^5	132	132	132
10	1.802×10^3	1.40921×10^7	1.39959×10^6	212	212	245
12	1.0383×10^4	2.03559×10^7	2.09477×10^6	264	264	264
14	5.2670×10^4	3.22177×10^7	2.84479×10^6	282	282	282
17	5.7411×10^5	4.39725×10^7	4.20395×10^6	2085	2154	2159
21	1.4324×10^7	7.58379×10^7	8.04416×10^6	2707	2760	3130
29		1.58345×10^8	1.53412×10^7	1610	1663	1706
42			3.45926×10^7	699		798
58			9.44869×10^7	25395		28293
120			5.62488×10^8	6942		8005



Rysunek 1: Czas wykonywania się algorytmu w sekundach w zależności od wielkości problemu



Rysunek 2: Najkrótsza ścieżka znaleziona przez algorytm a faktyczna najkrótsza ścieżka w zależności od wielkości problemu