Projektowanie Efektywnych Algorytmów

Kierunek *Informatyka*	**Termin** *Czwartek 17:25*
Temat *Algorytmy lokalnego przeszukiwania - Symulawane wyżarzanie*	**Problem** *TSP*
Skład grupy *241406 Krzysztof Jopek*	**Nr grupy** -
Prowadzący *Mgr inż. Radosław Idzikowski*	**data** *19 grudnia 2019*

1 Opis problemu

Problem komiwojażera jest problemem optymalizacyjnym, polegającym na odnalezieniu minimalnego cyklu Hamiltona w grafie pełnym ważonym. Nazwa pochodzi od typowej ilustracji problemu: dane jest n miast, które muszą zostać odwiedzone przez komiwojażera oraz odległości między nimi. Zadaniem jest znalezienie najkrótszej drogi przechodzącej przez wszystkie miasta oraz wracającej do miasta początkowego. Problem komiwojażera jest klasy NP-trudny. W realizacji drugiego etapu projektu została wykorzystana metoda symulowanego wyżarzania.

2 Metoda rozwiązania

2.1 Symulowane wyżarzanie

Algorytm Simulated Anealing (Symulowane wyżarzanie) to metoda przeszukiwania lokalnego, która bazuje na dynamicznej zmianie sąsiedztwa danego rozwiązania,jednakże nie przeszukuje całego sąsiedztwa i zmiana zachodzi pod konkretnym prawdopodobieństwem opisanym równaniem. Algorytm polega na losowym przetasowaniu permutacji ścieżki i przyjęciu nowego rozwiązania jeżeli jest lepsze, natomiast jeżeli rozwiązanie jest gorsze to przyjmuje się nowe z pewnym prawdopodobieństwem. Dzięki temu algorytm może wyjść poza obszar minimum lokalnego, co znacznie ułatwia odnalezienie najlepszego rozwiązania . Konieczne jest jednak odpowiednie „dostrojenie" algorytmu, czyli ustawienie specyficznych dla niego parametrów (temperatura początkowa, ilość iteracji, temperatura minimalna, współczynnik schładzania). Jednakże dobre dobranie parametrów nie gwarantuje znalezienia optymalnego wyniku - minimum globalnego, ponieważ zawsze występuje pewien czynnik losowy w generowaniu nowego sąsiada i w funkcji prawdopodobieństwa.

2.1.1 Opis działania oraz kod głównej funkcji algorytmu

Algorytm rozpoczyna się od wygenerowania losowej permutacji początkowej scieżki i wyliczenie jej kosztu . Następnie dokonywana jest losowa zamiana dwóch wierzchołków w permutacji typem sąsiedztwa swap. Nastepnie zostają porównane koszty obydwu ścieżek. Jeśli funkcja celu ścieżki z nowej permutacji jest mniejsza to ta permutacja staje się globalnie najlepszą scieżką oraz jego funkcja celu staje sie globalnym optimum.Jeżeli nie jest równa lub większa to zgodnie z pewnym prawdodobieństwem, wyliczonym przez funkcję prawdopodobienstwa

$$p = \frac{1}{1 + e^{\frac{g(x)-g(y)}{T}}}$$

algorytm może przyjąć gorszą ścieżkę za chwilowo globalnie najlepszą ścieżkę. Na prawdopodobieństwo ma bezpośredni wpływ temperatura, koszt starej ścieżki oraz koszt nowej ścieżki wygenerowanej z sąsiedztwa .Im wyższa jest temperatura, tym większe prawodpodobieństwo na zmianę 'potencjalnie lepszą w przyszłości' do znalezienia ścieżki dla optimum globalnego. Jeżeli funkcja nie pozwoliła na zmianę, wtedy następuje powrót do poprzedniej scieżki. Następnie zostaje wygenerowana nowa temperatura poprzez 'schładzanie' obecnej (polega to na przemnożeniu temperatury przez współczynnik z tablicy z zakresu < 0.75,0.999 >. Sprawdzane jest czy nowa temperatura nie przekroczyła minimalnej temperatury - jeżeli nie to następuje kolejne „przetasowanie" obecnie najlepszej ścieżki i dalsze szukanie minimum, a jeżeli tak, to algorytm rozpoczyna kolejną iterację, która będzie używać większego współczynnika schładzania. Algorytm kończy swoją pracę po wykonaniu wszystkich iteracji.

Listing 1: Algorytm symulowanego wyżarzania - wyliczanie prawdopodobieństwa

```
1 bool SimulatedAnnealing::probability(double a, double b)
2 {
3     double q = pow( M_E , ((-1 * (b - a)) / T));
4     //wzor na wyliczenie prawdopodobienstwa
5     double r = (double)rand() / RAND_MAX;
6     if (r < q)
```

```
7       {
8           // jesli uznano, ze to gorsza sciezka
9           return true;
10      }
11      else
12          return false;
13  }
```

Listing 2: Algorytm symulowanego wyżarzania

```
1   void SimulatedAnnealing::simulatedAnnealing()
2   {
3
4       // iteracja po zmiennych tyle razy ile poda  uzytkownik
5
6           for (int i = 0; i < iterationCounter; i++)
7           {
8               int cost = INT_MAX;
9               int* firstPerm = new int[size]; // permutacje miast
10              int* secondPerm = new int[size];
11              int firstVertex, secondVertex, a, b;
12
13              T = TMax; //przypisanie poczatkowej maksymalnej temperatury
14
15              generatePerm(firstPerm); // generowanie loswej permutacji
16              a = road(firstPerm);        //  obliczenie kosztu
17              for (int j = 0; j < size; j++)
18              {
19                  secondPerm[j] = firstPerm[j];
20              }
21
22              while (T > 0.01 ) // wykonuje sie, dopoki
23              //temperatura nie spadnie do wartosci minimalnej - (0.01)
24              {
25                  do
26                  {
27                      firstVertex = rand() % size;
28                      secondVertex = rand() % size;
29                  } while (firstVertex == secondVertex);
30
31                  swap(secondPerm[firstVertex],secondPerm[secondVertex]);
32                  // zamiana miast sasiednich -  metoda swap
33
34                  b = road(secondPerm); // obliczenie kosztu drugiej sciezki
35
36                  if (b <= a || probability(a, b) == true)  // jesli b <= a lub
37                  //prawdopodobienstwo jest true
38                  {
39                      a = b;
40                      if (a <= cost) // sprawdzenie, czy sciezka ma nizszy koszt
41                      //ni  obecne najlepsze rozwiazanie
```

```
42                          {
43                              cost = a;
44                          }
45                          firstPerm[firstVertex] = secondPerm[firstVertex];
46                          firstPerm[secondVertex] = secondPerm[secondVertex];
47                          // przyjmujemy druga permutacje jako kolejna wyjsciowa
48                      }
49                      else                    //powrot do poczatku, jesli nie ma szans
50                      //na wyznaczenie lepszego kosztu
51                      {
52                          swap(secondPerm[firstVertex],secondPerm[secondVertex]);
53                      }
54                      T *= constants[i]; // mnozenie temperatury przez wyznaczona
55                      //stala z tablicy
56                  }
57
58                  delete[] firstPerm;
59                  delete[] secondPerm;      //usuwanie permutacji, na ktorych
60                  //wykonala sie iteracja
61                  if (cost < bestCostOfRoad)
62                  {
63                      bestCostOfRoad = cost;        //wyznaczenie najlepszej sciezki
64                  }
65              }
66  }
```

3 Eksperymenty obliczeniowe

Obliczenia zastały wykonane na laptopie z procesorem i7-6700HQ, kartą graficzną NVIDIA GeForce GTX 960M, 8GB RAM i DYSK SSD. Jako miarę jakości algorytmu przyjęto średnie procentowe odchylenie (Percentage Relative Deviation, PRD) najlepszego otrzymanego rozwiązania π względem rozwiązania referencyjnego π^{ref}:

$$PRD(\pi) = 100\%(C_{max}(\pi) - C_{max}(\pi^{ref}))/C_{max}(\pi^{ref}) \tag{1}$$

Wszystkie wyniki oraz wykresy zebrano i przedstawiono na rysunkach od nr 1 do nr 7 gdzie:

- $PRD_{SA}(\%)$ - średnie procentowe odchylenie dla algorytmu Simulated Annealing,
- π - najlepsze możliwe rozwiązanie (optimum) dla algotyrmu Simulated Annealing,
- π^{ref} - rozwiązanie referencyjne dla algotyrmu Simulated Annealing,
- Tp - temperatura początkowa,
- i - liczba iteracji,
- r - rozmiar instancji,
- Tm - temperatura minimalna,
- t - czas wykonywania algorytmu,
- f - współczynnik chłodzenia,

	ATSP	Tp = r^2 * 700, i = 10, Tm = 0.01, f = <0.75, 0.999>			
Alias	Instancja	t [s]	π	π^{ref}	PRD_{SA} [%]
data17	br17	2,59	39	39	0,00%
data34	ftv33	3,19	1286	1286	0,00%
data36	ftv35	3,18	1473	1473	0,00%
data39	ftv38	3,36	1530	1551	1,35%
data43	p43	3,27	5620	5622	0,04%
data45	ftv44	3,36	1613	1625	0,74%
data48	ry48p	4,00	14422	14511	0,61%
data53	ftv53	3,60	6905	7157	3,52%
data56	ftv55	3,59	1608	1681	4,34%
data65	ftv64	3,69	1839	1978	7,03%
data70	ft70	3,99	38673	40362	4,18%
data71	ftv70	4,15	1950	2151	9,34%
data100	kro124	4,85	36230	40243	9,97%
data171	ftv170	6,01	2755	3887	29,12%
data323	rbg323	11,20	1326	1433	7,47%
data358	rbg358	13,21	1163	1281	9,21%
data403	rbg403	14,63	2465	2517	2,07%
data443	rbg443	16,69	2720	2803	2,96%

Rysunek 1: Pomiary dla parametrów Tp = $r^2 * 700, i = 10, Tm = 0.01, f =< 0.75, 0.999 >$

	ATSP	Tp = r^2 * 700, i = 10, Tm = 0.001, f = <0.75, 0.999>			
Alias	Instancja	t [s]	π	π^{ref}	PRD_{SA} [%]
data17	br17	3,13	39	39	0,00%
data34	ftv33	3,54	1286	1286	0,00%
data36	ftv35	3,59	1473	1487	0,94%
data39	ftv38	3,91	1530	1597	4,20%
data43	p43	3,76	5620	5622	0,04%
data45	ftv44	3,52	1613	1676	3,76%
data48	ry48p	3,69	14422	14641	1,50%
data53	ftv53	3,70	6905	7339	5,91%
data56	ftv55	3,69	1608	1721	6,57%
data65	ftv64	3,85	1839	1998	7,96%

Rysunek 2: Pomiary dla parametrów Tp = $r^2 * 700, i = 10, Tm = 0.001, f =< 0.75, 0.999 >$

	ATSP	Tp = r^2 * 700 , i = 10, Tm = 0.01, f = 0.99			
Alias	Instancja	t [s]	π	π^{ref}	PRD_{SA} [%]
data17	br17	0,012	39	39	0%
data34	ftv33	0,023	1286	1566	18%
data36	ftv35	0,026	1473	1781	17%
data39	ftv38	0,033	1530	1906	20%
data43	p43	0,025	5620	6175	9%
data45	ftv44	0,026	1613	2210	27%
data48	ry48p	0,027	14422	17942	20%
data53	ftv53	0,027	6905	10049	31%
data56	ftv55	0,031	1608	2594	38%
data65	ftv64	0,028	1839	3478	47%

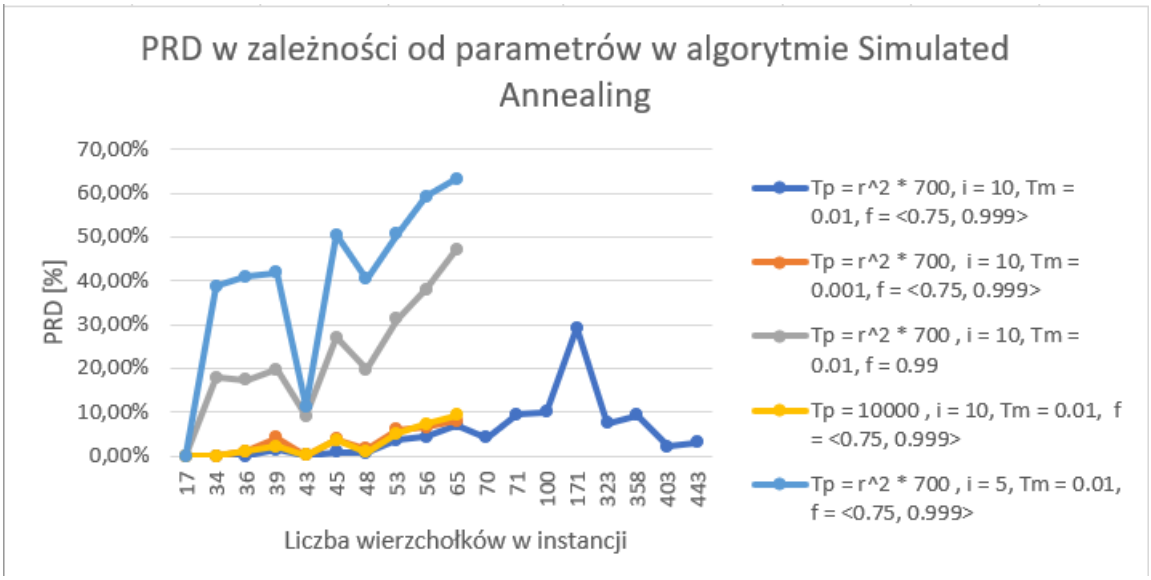
Rysunek 3: Pomiary dla parametrów Tp = $r^2 * 700, i = 10, Tm = 0.01, f = 0.99$

	ATSP	Tp = 10000 , i = 10, Tm = 0.01, f = <0.75, 0.999>			
Alias	Instancja	t [s]	π	π^{ref}	PRD_{SA} [%]
data17	br17	1,98	39	39	0%
data34	ftv33	2,51	1286	1286	0%
data36	ftv35	2,48	1473	1487	1%
data39	ftv38	2,50	1530	1562	2%
data43	p43	2,31	5620	5623	0%
data45	ftv44	2,40	1613	1674	4%
data48	ry48p	2,52	14422	14535	1%
data53	ftv53	2,77	6905	7277	5%
data56	ftv55	2,45	1608	1732	7%
data65	ftv64	2,47	1839	2028	9%

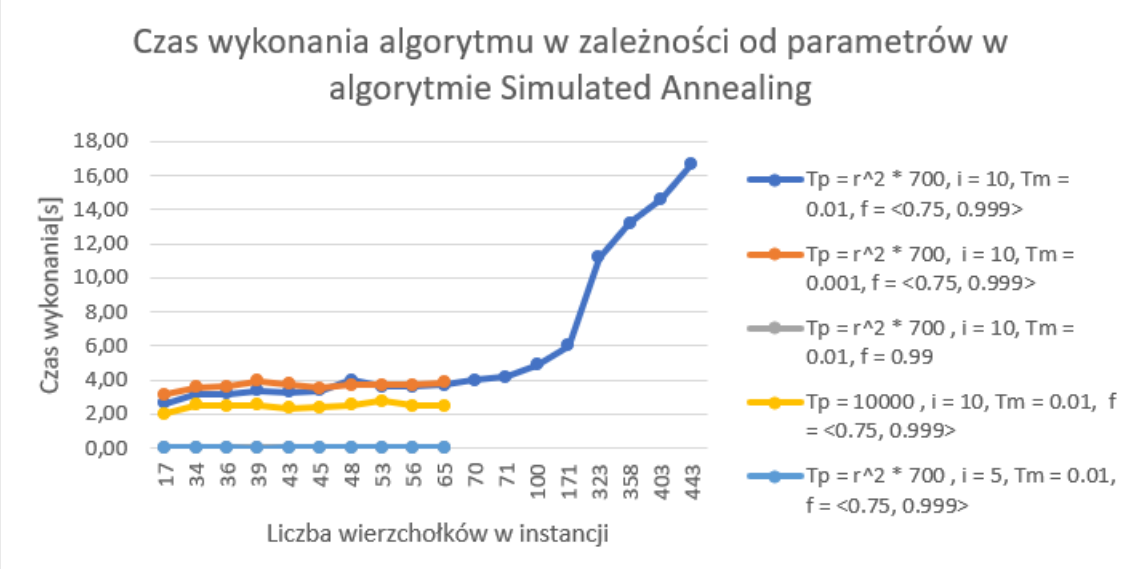
Rysunek 4: Pomiary dla parametrów Tp = 10000 , i = 10, Tm = 0.01, f = <0.75, 0.999>

	ATSP	Tp = r^2 * 700 , i = 5, Tm = 0.01, f = <0.75, 0.999>			
Alias	Instancja	t [s]	π	π^{ref}	PRD_{SA} [%]
data17	br17	0,0011	39	39	0%
data34	ftv33	0,0012	1286	2098	39%
data36	ftv35	0,0012	1473	2496	41%
data39	ftv38	0,0010	1530	2634	42%
data43	p43	0,0014	5620	6335	11%
data45	ftv44	0,0012	1613	3259	51%
data48	ry48p	0,0013	14422	24167	40%
data53	ftv53	0,0012	6905	14001	51%
data56	ftv55	0,0016	1608	3942	59%
data65	ftv64	0,0012	1839	4994	63%

Rysunek 5: Pomiary dla parametrów Tp = $r^2 * 700, i = 5, Tm = 0.01, f =< 0.75, 0.999 >$



Rysunek 6: PRD_{SA} w zależności od parametrów w algorytmie Simulated Annealing



Rysunek 7: Czas wykonania algorytmu w zależności od parametrów w algorytmie Simulated Annealing

4 Wnioski

Z powyższych wykresów i tabel można wywnioskować, iż algorytm symulowanego wyżarzania jest dokładniejszy na mniejszych instancjach problemu. Wraz ze wzrostem liczby miast rośnie czas działania algorytmu. Ponadto na dokładnośc algorytmu mają duży wpływ wprowadzane parametry. Zarówno zmniejszenie ilości iteracji, jak i zwiększenie współczynnika chłodzenia skraca czas wykonywania algorytmu, jednakże wtedy diametralnie zwiększa się PRD (do nawet 63 %). Natomiast zmniejszenie temperatury początkowej lub zmniejszenie temperatury minimalnej nie wprowadziło większych zmian w pomiarach, można zauważyć lekko zwiększone PRD dla wyższych instancji. Ze względu na charakter algorytmu nie da się jednoznacznie stwierdzić jego złożoności obliczeniowej, lecz można założyć, że czas jego wykonania rośnie wraz z liczbą iteracji czy też ilości miast. Kluczowe dla tego problemu staje się odpowiednie dobranie parametrów, aby algorytm wykonał swoją pracę jak najdokładniej. Efekt ten udało się uzyskać z różnym skutkiem – zakres błędu oscylował od bardzo małego (0-4%) do dużego (ponad 50%) dla najgorszego zestawienia parametrów algorytmu.

Literatura

[1] Optymalizacja dyskretna (w tym algorytmy heurystyczne), symulowane wyżarzanie. https://nowosad.github.io/ahod/11-simulated-annealing.html#1. Accessed: 2019-12-19.

[2] Projektowanie efektywnych algorytmów - poszukiwanie lokalne i symulowane wyżarzanie. http://www.zio.iiar.pwr.wroc.pl/pea/w4_ls.pdf. Accessed: 2019-12-19.

[3] Simulated annealing. https://en.wikipedia.org/wiki/Simulated_annealing. Accessed: 2019-12-19.

[4] Symulowane wyżarzanie. https://tdb0.wordpress.com/2010/12/03/symulowane-wyzarzanie/. Accessed: 2019-12-19.