Projektowanie Efektywnych Algorytmów

Kierunek *Informatyka*	**Termin** *Piątek 15:15*
Temat *Algorytmy lokalnego przeszukiwania*	**Problem** *TSP*
Skład grupy *Dawid Zawadzki 241116*	**Nr grupy** -
Prowadzący *Mgr inż. Radosław Idzikowski*	**data** *18 grudnia 2019*

1 Opis problemu

Problem komiwojażera (TSP), jest to zagadnienie optymalizacyjne, polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Problem definiowany jest przez zbiór wierzchołków V i krawędzi E, gdzie wierzchołki są przedstawiane jako miasta, a krawędzie jako łączące je drogi. Zadaniem jest znalezienie najkrótszej scieżki, która przechodzi przez każde miasto ze zbioru V tylko raz i kończy trase w mieście z którego zaczeliśmy podróz. TSP, należy do klasy problemów NP trudnych, oznacza to, że nie istnieje algorytm rozwiązujący problem w czasie wielomianowym oraz nie można zweryfikować w czasie wielomianowym, czy uzyskana odpowiedź jest poprawna. Algorytmy przeszukiwania lokalnego są jednym z typów algorytmów, które można wykorzystać do rozwiązywania TSP, charakteryzują się tym, że zwracany wynik nie zawsze jest optymalny, a czasem nawet nieprawidłowy. Algorytmy przeszukiwania lokalnego są wykorzystywane gdy algorytmy dokładne są zbyt kosztowne w użyciu.

2 Metoda rozwiązania

2.1 Symulowane wyżarzanie

Symulowane wyżarzanie jest rodzajem algorytmu heurystycznego, który jest inspirowany procesem wyżarzania w metalurgii. Wyżarzanie wiąże się z podgrzewaniem i chłodzeniem materiału, aby modyfikować jego fizyczne właściwości dzięki zmianom w jego wewnętrznej strukturze. Podczas chłodzenia materiał staje się sztywny co pozwala mu na zachowanie nowo otrzymanych właściwości. W symulowanym wyżarzaniu tworzymy zmienną, która symuluje proces chłodzenia. Na początek nadajemy jej wysoką wartość i zmniejszamy z biegem algorytmu. Podczas, gdy temperatura jest wysoka algorytm będzie z dużym prawdopodobieństwem akceptował wartości, które są gorsze od naszego aktualnego minimium. Dzięki akceptacji słabszych wyników algorytm jest w stanie wyjść z minimum lokalnego w poszukiwaniu wyniku bliskiego optymalnemu w innym obszarze. Gdy temperatura maleje, maleje też prawdopodobieństwo akcpetacji gorszego wyniku, a algorytm skupia się na jednym obszarze z nadzieją, że znajdzie w nim wynik wystarczający. Efektywność algorytmów heurystycznych zależy od dobranych przez projektanta parametrów algorytmu oraz zaimplementowanych mechanizmów, które mogą skutecznie zwiększyć szybkość oraz jakość otrzymywanych rozwiązań. Kody zaprojektowanych przeze mnie metod algorytmu są przedstawione w **Listing 1**, **Listing 2**, **Listing 3**, **Listing 4**.

Listing 1: Algorytm symulowanego wyżarzania dla zbiorów symetrycznych

```
1  void Annealing::annealingAcc(vector<int> path,double cooling, double endTemp
2  , double temp, int step, int reduce){
3      srand(time(NULL));
4
5      vector<int> pathC(path);
6
7      int min = matrix.pathValue(path);
8
9      int minC = min;
10
11     int start;
12
13     int end;
14
15     int noChangeCounter;
16     int changeCounter;
17
18     while(temp > endTemp) {
```

```
19          noChangeCounter = 0;
20          changeCounter = 0;
21          for (int i = 0; i < step; i++) {
22              start = rand() % (matrix.getMatrixSize() - 2) + 1;
23
24              end = rand() % (matrix.getMatrixSize() - start - 1) + start + 1;
25
26              if(end < (matrix.getMatrixSize()-1))
27                  minC -= (matrix[pathC[start - 1]][pathC[start]] +
28                  matrix[pathC[end]][pathC[end + 1]]);
29              else
30                  minC -= (matrix[pathC[start - 1]][pathC[start]] +
31                  matrix[pathC[end]][0]);
32
33              invert(pathC, start, end);
34
35              if(end < (matrix.getMatrixSize()-1))
36                  minC += (matrix[pathC[start - 1]][pathC[start]] +
37                  matrix[pathC[end]][pathC[end + 1]]);
38              else
39                  minC += (matrix[pathC[start - 1]][pathC[start]] +
40                  matrix[pathC[end]][0]);
41
42              if (accept(min, minC, temp)) {
43                  min = minC;
44
45                  path = pathC;
46                  noChangeCounter = 0;
47                  changeCounter++;
48              }else{
49                  pathC = path;
50
51                  minC = min;
52
53                  noChangeCounter++;
54                  changeCounter = 0;
55              }
56              if(changeCounter == reduce)
57                  break;
58          }
59          if(noChangeCounter == step)
60              break;
61          temp *= cooling;
62
63      }
64  }
```

Listing 2: Algorytm symulowanego wyżarzania dla zbiorów asymetrycznych

```
1  void Annealing::annealingAccAsym(vector<int> path, double cooling,
2  double endTemp,double temp, int step, int reduce){
```

```
3       srand(time(NULL));
4       vector<int> pathC(path);
5       int min = matrix.pathValue(path);
6       int minC = min;
7       int start;
8       int end;
9       int noChangeCounter;
10      int changeCounter;
11      while(temp > endTemp) {
12          noChangeCounter = 0;
13          changeCounter = 0;
14          for (int i = 0; i < step; i++) {
15              start = rand() % (matrix.getMatrixSize() - 2) + 1;
16              end = rand() % (matrix.getMatrixSize() - start - 1) + start + 1;
17              minC -= countSetCost(pathC, start, end);
18              invert(pathC, start, end);
19              minC += countSetCost(pathC, start, end);
20              if (accept(min, minC, temp)) {
21                  min = minC;
22                  path = pathC;
23                  noChangeCounter = 0;
24                  changeCounter++;
25              }else{
26                  pathC = path;
27                  minC = min;
28                    noChangeCounter++;
29                  changeCounter = 0;
30              }
31              if(changeCounter == reduce)
32                  break;
33          }
34          if(noChangeCounter == step)
35            break;
36          temp *= cooling;
37
38      }
39  }
```

Listing 3: Funkcja obliczająca koszt odwróconego podzbioru

```
1   int Annealing::countSetCost(vector<int> & path, int start, int end) const {
2       int sum = 0;
3       for(int i=start-1; i<end; i++)
4           sum+=matrix[path[i]][path[i+1]];
5       if(end == matrix.getMatrixSize() - 1)
6           sum+=matrix[path[end]][0];
7       else
8           sum+=matrix[path[end]][path[end+1]];
9       return sum;
10  }
```

Listing 4: Funkcja odwracająca dany podzbiór

```
1 void Annealing::invert(vector<int> & path, int start, int end) {
2     int level = 0;
3     for (int i = start; i <= end - level; i++) {
4         swap(path[i], path[end - level]);
5         level++;
6     }
7 }
```

3 Eksperymenty obliczeniowe

3.1 Sprzęt

Obliczenia zastały wykonane na komputerze klasy PC z procesorem Intel Core i5-2540M CPU 2.60GHz, kartą graficzną Intel HD 4000, 4GB RAM i DYSK SSD.

3.2 Mechanizmy

Zaimplementowane przeze mnie algorytmy posiadały niżej wymienione mechanizmy:

- Akceleracja obliczania wartości ścieżki,
- Mechanizm wcześniejszego zmniejszenia temperatury,
- Mechanizm wcześniejszego skończenia algorytmu,

Bardziej szczegółowe omówienie wymienionych funkcji zostanie zrealizowane w dalszej części sprawozdania.

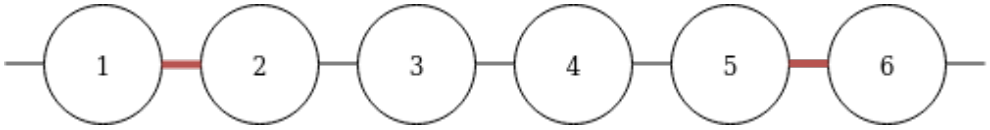
3.3 Parametry

Podczas wykonywanych badań korzystałem z parametrów posiadających dane wartości:

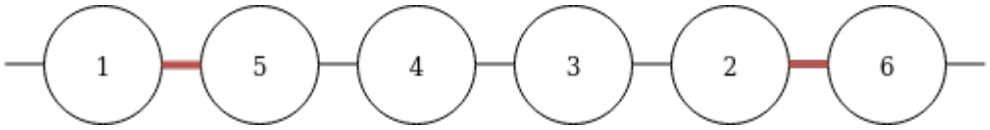
- temp = 10000000000.0, określa temperaturę początkową algorytmu,
- endTemp = 0.0001, określa temperaturę końcową algorytmu,
- cooling = 0.9, określa współczynnik chłodzenia po każdej iteracji,
- step = (ilość wierzchołków*ilość wierzchołków), określa liczbę iteracji, dla każdej temperatury,
- reduce = (10*ilość wierzchołków), określa warunek wcześniejszego zmniejszenia temperatury.

3.4 Mechanizm akceleracji

Zaimplementowana przeze mnie funkcja, w każdej iteracji odwraca podzbiór pomiędzy dwoma wylosowanymi wierzchołkami, a następnie oblicza wartość nowo otrzymanego zbioru. Ten sposób losowania zbioru ma pewną zależność, którą warto wykorzystać do przyspieszenia obliczeń. Jak widać na **Rysunek 1** i **Rysunek 2** dla symetrycznego TSP jedyna zmiana zachodzi dla zewnętrznych krawędzi wylosowanych wierzchołków.

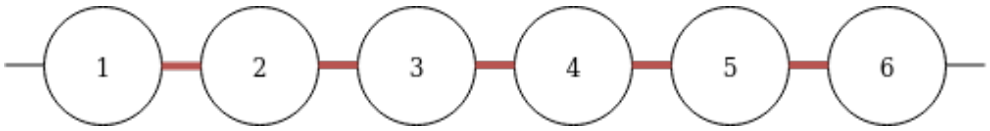


Rysunek 1: Wylosowany zbiór przed odwróceniem.



Rysunek 2: Wylosowany zbiór po odwróceniu.

Tak jak wspomniałem wyżej ta zależność działa tylko dla zbiorów symetrycznych, ponieważ koszt przejścia z np. wierzchołka 2 do wierzchołka 3 jest taki sam jak z 3 do 2. W przypadku zbiorów asymetrycznych sytuacja wygląda inaczej. Koszt przejścia między dwoma wierzchołkami może się różnić w zależności od ich ustawienia, dlatego w tym przypadku należy obliczyć wartość wszystkich krawędzi w zamienionym podzbiorze, tak jak to pokazano na **Rysunek 3**.



Rysunek 3: Zmiana krawędzi w asymetrycznym TSP.

3.5 Mechanizmy wcześniejszego zmniejszenia temperatury

Dla każdej temperatury algorytm wykonuje zadaną w parametrze **step** liczbę obrotów pętli. Podczas, gdy temperatura jest wysoka, algorytm z dużym pradopodobieństwem akceptuje zmianę wyniku na gorszy niż jest dotychczas. Omawiany w tym podrozdziale mechanizm zlicza ile razy pod rząd nastąpiła zmiana aktualnej wartości minimalnej, a kiedy zliczona wartość będzie równa wartości przechowywanej w zmiennej **reduce** następuje wyjście z pętli i zmiana temperatury na mniejszą. W późniejszych rozdziałach sprawdzimy czy użycie owego mechanizmu akceleracyjnego nie wpływa negatywnie na jakość uzyskanych wyników algorytmu.

3.6 Mechanizm wcześniejszego skończenia algorytmu

Dany mechanizm kończy algorytm i zwraca aktualnie najmniejszą wartość, jeśli podczas trwania pętli dla danej temperatury nie nastąpiła żadna zmiana aktualnej wartośći minimalnej. W późniejszych rozdziałach sprawdzimy czy użycie owego mechanizmu akceleracyjnego nie wpływa negatywnie na jakość uzyskanych wyników algorytmu.

3.7 Wyniki

Tablica 1 przedstawia wyniki osiągnięte, dla symetrycznego TSP, gdzie przedstawione wyniki są rezultatem średniej wyliczonej na podstawie 10 wywołań algorytmu. Algorytm posiada wszystkie wymienione w punkcie **3.2** mechanizmy, dzięki czemu czas wykonanania jest stosunkowo krótki, nawet dla dużej instancji problemu.

Rozmiar problemu	**Czas[s]**	**Optymalny wynik**	**Średni wynik**	**Średnia różnica %**	**Największy wynik**	**Największa różnica %**
10	0.00772564	212	212	0	212	0
12	0.00786621	264	264	0	264	0
13	0.00947837	269	269	0	269	0
15	0.0119682	291	291	0	291	0
21	0.0198072	2707	2707	0	2707	0
24	0.0274074	1272	1272	0	1272	0
26	0.0336699	937	937	0	937	0
29	0.0359967	1610	1611	0.075	1622	0.745
42	0.0812363	699	703	0.615	724	3.577
58	0.230392	25395	25403	0.032	25475	0.315
120	1.05469	6942	7066	1.799	7176	3.371

Tablica 1: Wyniki obliczeń dla grafów symetrycznych, dla 10 powtórzeń algorytmu.

Tablica 2 przedstawia wyniki osiągnięte, dla symetrycznego TSP, gdzie przedstawione wyniki są rezultatem średniej wyliczonej na podstawie 100 wywołań algorytmu. Użyty algorytm był identyczny jak w przypadku wyników przedstawionych w **Tablica 1**.

Rozmiar problemu	**Czas[s]**	**Optymalny wynik**	**Średni wynik**	**Średnia różnica %**	**Największy wynik**	**Największa różnica %**
10	0.00764321	212	212	0	212	0
12	0.00921266	264	264	0	264	0
13	0.0106423	269	269	0	269	0
15	0.0131738	291	291	0	291	0
21	0.0241173	2707	2707	0.035	2801	3.472
24	0.0369396	1272	1272	0	1272	0
26	0.046056	937	937	0.019	955	1.921
29	0.0420481	1610	1610	0.028	1625	0.932
42	0.289706	699	699	0.029	719	2.861
58	0.289706	25395	25395	0	25395	0
120	1.67968	6942	7023	1.178	7047	1.513

Tablica 2: Wyniki obliczeń dla grafów symetrycznych, dla 100 powtórzeń algorytmu.

Jak widać w **Tablica 1** oraz **Tablica 2** uzyskane wyniki są bardzo zadowalające niezależnie od instancji problemu oraz liczby wykonanych powtórzeń badania. Wyniki w przybliżeniu posiadają masymalnie **3%** błędy.

Tablica 3 przedstawia wyniki osiągnięte, dla asymetrycznego TSP, gdzie przedstawione wyniki są rezultatem średniej wyliczonej na podstawie 10 wywołań algorytmu. Użyty algorytm posiada wszystkie wymienione w punkcie **3.2** mechanizmy.

Rozmiar problemu	**Czas[s]**	**Optymalny wynik**	**Średni wynik**	**Średnia różnica %**	**Największy wynik**	**Największa różnica %**
17	0.0367574	39	39	0	39	0
33	0.119506	1286	1591	23.717	1651	28.383
35	0.108741	1473	1894	28.608	2222	50.849
38	0.149313	1530	1885	23.229	2096	36.994
43	0.332541	5620	5622	0.036	5622	0.036
45	0.235847	1613	2202	36.5654	2393	48.3571
48	0.196876	14422	14898	3.30329	15164	5.14492
53	0.263456	6905	10270	48.7343	10781	56.1332
56	0.541653	1608	2405	49.5771	2774	72.5124
65	0.663473	1839	2941	59.9565	2968	61.3921
70	0.494065	38673	46406	19.9979	48264	24.8002
124	1.433	36230	44714	23.4171	46601	28.6254
170	12.0662	2755	7484	171.673	7724	180.363
323	74.6889	1326	3108	134.404	3200	141.327

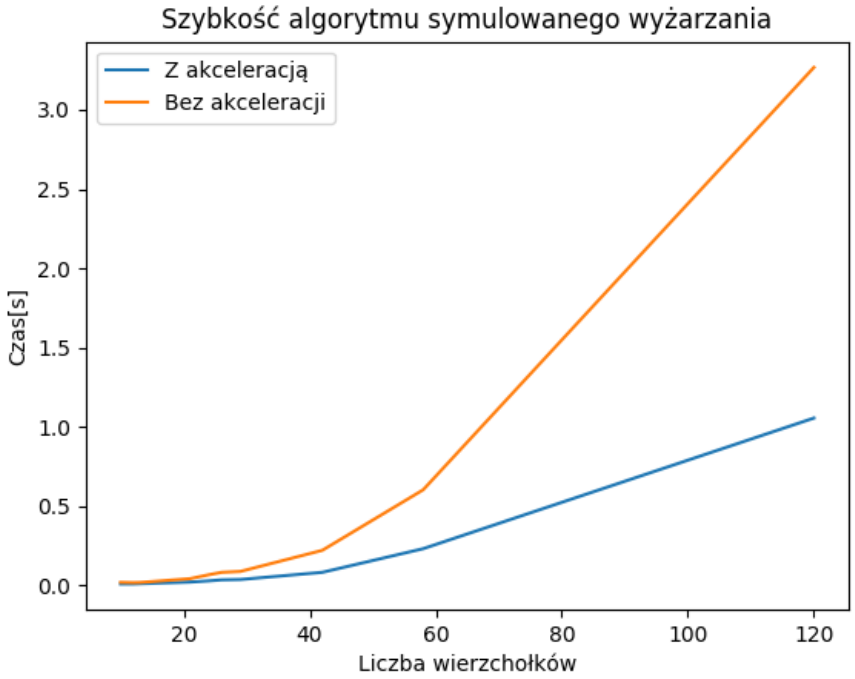
Tablica 3: Wyniki obliczeń dla grafów asymetrycznych, dla 10 powtórzeń algorytmu z mechanizmami wcześniejszego zmniejszenia temperatury oraz wcześniejszego skończenia algorytmu.

Tablica 4 przedstawia wyniki osiągnięte, dla asymetrycznego TSP, gdzie przedstawione wyniki są rezultatem średniej wyliczonej na podstawie 100 wywołań algorytmu. Użyty algorytm w przeciwieństwie do wyników z **Tablica 3** posiada zaimplementowany jedynie mechanizm akceleracji obliczania wartości ścieżki.

Rozmiar problemu	**Czas[s]**	**Optymalny wynik**	**Średni wynik**	**Średnia różnica %**	**Największy wynik**	**Największa różnica %**
17	0.0367574	39	39	0	39	0
33	0.214259	1286	1584	23.219	1721	33.826
35	0.254501	1473	1809	22.838	1998	35.642
38	0.287314	1530	1898	24.065	2013	31.569
43	0.375066	5620	5622	0.036	5637	0.302
45	0.408649	1613	2103	30.434	2148	33.168
48	0.497898	14422	15081	4.571	15585	8.064
53	0.641814	6905	10219	47.996	11726	69.819
56	0.78059	1608	2384	48.277	2769	72.2015
65	1.12581	1839	2917	58.668	3850	109.353
70	1.48934	38673	46111	19.234	48019	24.167
124	3.84338	36230	41366	14.176	41366	14.1761
170	14.3266	2755	7306	165.209	7485	171.688
323	122.425	1326	3123	135.535	3168	138.914

Tablica 4: Wyniki obliczeń dla grafów asymetrycznych, dla 10 powtórzeń algorytmu bez mechanizmów wcześniejszego zmniejszenia temperatury oraz wcześniejszego skończenia algorytmu.

Jak widać w **Tablica 3** oraz **Tablica 4** użycie mechanizmu wcześniejszego zmniejszenia temperatury oraz wcześniejszego skończenia algorytmu nie wpływa znacznie na jakość otrzymywanego wyniku, ale daje rezultaty w szybszym czasie.



Rysunek 4: Porównanie czasu wykonania algorytmu dla różnych instancji problemu.

Na **Rysunek 4** widać różnicę pomiędzy algorytmem z zaimpelmentowanym mechanizmem akceleracji obliczania wartości ścieżki oraz bez tego mechanizmu. Jak widać czas wykonania algorytmu z mechanizmem akceleracji jest nawet **3 razy** szybszy niż bez. Użycie algorytmu dla większej liczby wierzchołków lub dla innych parametrów będzie dawać znacznie lepsze rezultaty. Jest to jedyny z zaprezentowanych mechanizmów, który napewno nie będzie negatywnie oddziaływał na jakość rozwiązania.

4 Wnioski

Symulowane wyżarzanie jest dobrym algorytmem do rozwiązywania TSP dla niewielkiej oraz dużej liczby wierzchołków grafu. Jego zaletą jest fakt, że niezaleznie od rozmiaru problemu zawsze zwróci wynik. Dla zadanych przeze mnie parametrów, algorytm zwracał bardzo zadawalające wyniki dla każdej badanej wielkości problemów symetrycznych. Niestety nieporadził sobie z problemami asymetrycznymi gdzie wyniki miały ponad **100%** błędy, moze to wynikać ze źle dobranych parametrów dla tego rodzaju problemu. Niestety w świecie rzeczywistym większość obszarów, w których symulowane wyżarzanie mogłoby znaleźć swoje zastosowanie ma charakter asymetryczny. Zaimpelementowane mechanizmy znacząco poprawiły szybkość algorytmu, praktycznie nie wpływając na jakość otrzymywanych wyników.

5 Literatura

- Lee Jacobson, Simulated Annealing for beginners, http://www.theprojectspot.com/tutorial-post/simulated-annealing-algorithm-for-beginners/6,
- Todd W. Schneider, The Traveling Salesman with Simulated Annealing, R, and Shiny, https://toddwschneider.com/posts/traveling-salesman-with-simulated-annealing-r-and-shiny,
- Dr Maciej Komosiński, Symulowane wyżarzanie: inspiracje, idea działania, schematy chłodzenia, dobór „temperatury", https://www.youtube.com/watch?v=gX-X85dCib0,
- John Walker, Simulated Annealing The Travelling Salesman Problem, https://www.fourmilab.ch/documents/travelling/anneal/