Projektowanie Efektywnych Algorytmów

Kierunek *Informatyka*	**Termin** *Piątek 13:15*
Temat *Symulowane Wyżarzanie*	**Problem** *WST*
Skład grupy *230504 Jeremiasz Romejko*	**Nr grupy**
Prowadzący *Mgr inż. Radosław Idzikowski*	**data** *14 grudnia 2018*

1 Opis problemu

Kryterium Ważonej Sumy Opóźnień dla jednomaszynowego problemu szeregowania zadań*(Weighted Sum Tardiness for task scheduling)*. W problemie mamy zbiór n zadań wykonywanych na jednej maszynie.

$$J = J_1, J_2, ...J_i, ..., J_n, \tag{1}$$

każde i-te zadanie składa się z trzech parametrów:

- p_i - czas wykonania,
- w_i - współczynnik kary,
- d_i - żądany termin zakończenia.

Każde zadanie musi być wykonywane nieprzerwanie przez p_i oraz powinno być ukończone przed terminem d_i , w przeciwnym wypadku zostanie naliczona kara. Naraz może być wykonywane tylko jedynie jedno zadanie. Przez π będziemy oznaczać kolejność wykonywania zadań. Dla każdego zadania należy obliczyć jego spóźnienie T_i:

$$T_i = max(C_i - d_i, 0) \tag{2}$$

gdzie C_i jest to moment zakończenia i-tego zadania, a wcześniejsze zakończenie nie jest dodatkowo premiowane. Kryterium optymalizacyjnym jest suma $w_i T_i$:

$$\sum_{i=1}^{n} w_i T_i \tag{3}$$

2 Metoda rozwiązania

2.1 Algorytm 1

Algorytm symulowanego wyżarzania jest rozwiązaniem aproksymacyjnym, ponieważ z danym prawdopodobieństwem jesteśmy w stanie stwierdzić czy dany wynik π jest rozwiązaniem optymalnym. Losujemy rozwiązanie naszego problemu, jeżeli jest lepsze od poprzedniego przyjmujemy je a licznik prób zerujemy jeżeli jest nie poprawne to przyjmujemy je z danym prawdopodobieństwem P.

$$P = exp(\pi_o - \pi/temp) \tag{4}$$

oraz dodajemy do licznika 1. Jeśli po 3 próbach rozwiązanie nie będzie lepsze obniżamy temperature.

Listing 1: Algorytm symulowanego wyżarzania

```
1  {
2      //zerowanie
3      int helper;
4      int counter = 0;
5      do
6      {
7          //losowa permutacja
8          std::random_shuffle(optTaskList.begin(),optTaskList.end());
9          //liczenie funkcji celu
10         helper = countResultVector(optTaskList);
11         //jezeli rozwiazanie jest lepsze
12         if (result >= helper)
13         {
```

```
14              counter = 0;
15              result = helper;
16          }
17          //jezeli prawdopodobienstwo jest wystarczajaco duze,
18          //aby przyjac gorsze rozwiazanie
19          else if (probability(result, helper, temperature))
20          {
21              result = helper;
22              counter++;
23          }
24          else
25          {
26              //incrematacja prob
27              counter++;
28          }
29          //jesli to 3 proba
30          if (counter >= 3.0)
31          {
32              //obnizamy temperature i zerujemy licznik
33              temperature = temperature - 0.0001;
34              counter = 0;
35          }
36          //powtarzamy az temperatura <= 0
37      }while (temperature > 0.0);
38      showResultVector();
39  }
40  bool Machine::probability(int optResult, int result, double temperature)
41  {
42      double P;
43      //roznica rozwiazan
44      deltaL = optResult - result;
45      //liczmy exp
46      P = exp(deltaL / temperature);
47      //i przy pomocy dystrybuanty bernuliego decydujemy czy zwracamy 0 czy jeden
48      std::bernoulli_distribution random_bool_generator(P);
49      return random_bool_generator(rand_engine);
50  }
51  //funkcja celu
52  int Machine::countResultVector(std::vector<Task> taskList)
53  {
54      int result = 0;
55      int time = 0;
56      for (int i = 0; i < this->size; i++)
57      {
58          if (taskList[i].deadLine - taskList[i].executionTime - time < 0)
59          {
60              result += (taskList[i].deadLine - taskList[i].executionTime - time)*
61              -1 * taskList[i].retioPunishment;
62          }
63
```

```
64              time += taskList[i].executionTime;
65          }
66          return result;
67      }
```

3 Eksperymenty obliczeniowe

Obliczenia zastały wykonane na komputerze klasy PC z procesorem AMD Athlon II X4 620, kartą graficzną NVIDIA GeForce GTS 250, 8GB RAM i DYSK SSD. Dla danej instancji został zmierzony czas znalezienia optymalnego rozwiązania t.Dla każdego n eksperyment powtórzono 10 razy i wyliczono z sumy t średnią arytmetyczną:

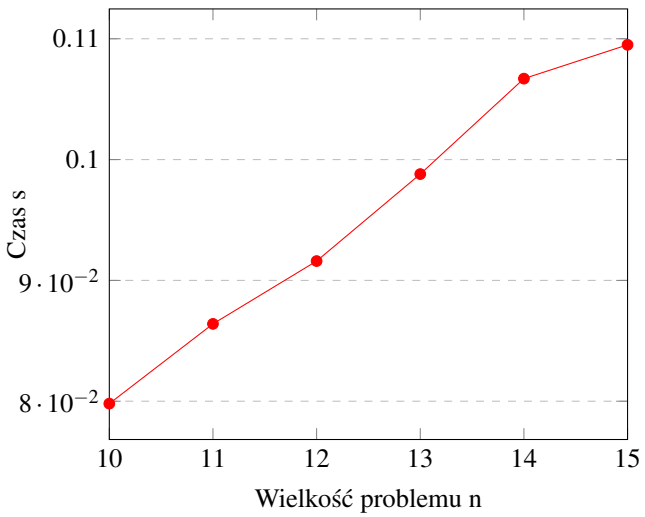
$$T = \frac{\sum_{i=1}^{n} t_i}{10} \tag{5}$$

Wszystkie wyniki zebrano i przedstawiono w tabeli nr 2 gdzie:

- n - liczba zadań,
- T - średni czas znalezienia optymalnego π w s

Tablica 1: Czas obliczeń dla ustalonej liczby instancji.

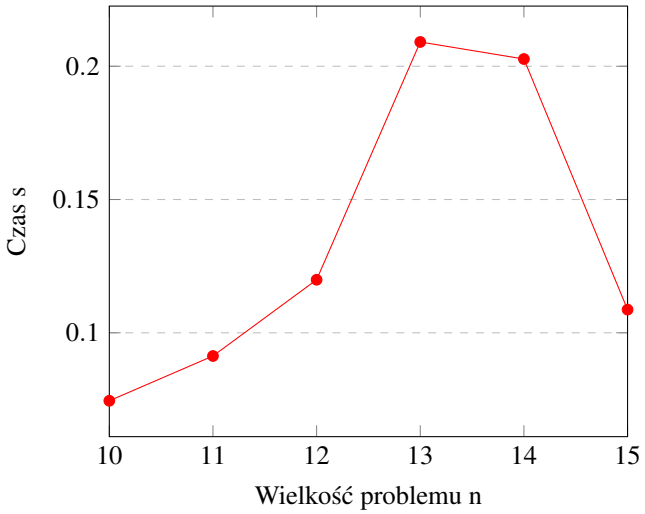
n	T	π	opt
10	0.0798	1011	1004
11	0.0864	962	962
12	0.0916	951	915
13	0.0988	939	681
14	0.1067	995	646
15	0.1095	359	310



Rysunek 1: Czas wykonywania się algorytmów w zależności od wielkości problemu losowe nowe rozwiązanie

Tablica 2: Czas obliczeń dla ustalonej liczby instancji.

n	T	π	opt
10	0.0745	1025	1004
11	0.0913	1119	962
12	0.1199	1384	915
13	0.2091	1436	681
14	0.2027	1650	646
15	0.1087	1794	310



Rysunek 2: Czas wykonywania się algorytmów w zależności od wielkości problemu swap nowe rozwiązanie

4 Wnioski

Jest to algorytm który jest o wiele szybszy od algorytmów przeglądu zupełnego, za to zwykle nie otrzymujemy dzięki niemu rozwiązań optymalnych ,a tylko zbliżone do nich. Jak widać na powyższych tabelach znajdowanie nowych rozwiazań za pomocą losowości daje lepsze rozwiązania (bliższe optymalnym), niż zamienianie kolejnych sąsiadów ze sobą.Symulowane wyrzażanie bardzo uzależnione jest od parametrów początkowych które można dobierać po porzez doświadczenie/heurystyke.