Schemat Aproksymacyjny dla $P_2||C_{max}$

Autorzy:
Jacek Wieczorek (181043)

Prowadzący : mgr inż. Karolina Mokrzysz
Poniedziałek TP 11.00 - 13.00

Wydział Elektroniki
III rok

11 stycznia 2012

1 Opis problemu

Celem laboratorium jest zaimplementowanie algorytmu szeregowania n zadań na 2 równolegle działających procesorach - $P_2||C_{max}$ za pomocą wielomianowego schematu aproksymacyjnego *PTAS*.

2 PTAS

Wielomianowy schemat aproksymacji (ang. Polynomial-time approximation scheme, w skrócie PTAS) to algorytm aproksymacyjny, który pozwala na uzyskanie dowolnie dobrego rozwiązania przybliżonego danego problemu optymalizacyjnego, i którego złożoność czasowa jest wielomianowa dla każdej żądanej dokładności.

Algorytm A jest wielomianowym schematem aproksymacji dla problemu Π jeśli spełnione są następujące warunki:

- dla każdego odpowiedniego ε A jest algorytmem ε-aproksymacyjnym dla Π,
- dla każdego odpowiedniego ε złożoność czasowa A jest wielomianowa ze względu na rozmiar instancji problemu podanej na wejściu A.

3 Opis algorytmu

- Krok 1 : Posortuj zadania malejąco wg czasu ich wykonania
- Krok 2 : Przyporządkuj k-pierwszych zadań w sposób optymalny
- Krok 3 : Pozostałe $n - k$ zadań przyporządkuj za pomocą algorytmu *LPT*

4 Dowód wielomianowości

- Sortowanie: $O(nlogn)$
- Przyporządkowanie 1 : przegląd zupełny przy pomocą wektora binarnego - 2^k

- Przyporządkowanie $LPT : O(n-k)$

Złożóność obliczeniowa : $O(nlogn + 2^k)$

Dokładność : $\varepsilon = \frac{1}{1+\lfloor k/2 \rfloor}$

Wynika z tego, iż k = $O(\frac{1}{\varepsilon})$

Co daje nam złożoność obliczeniową algorytmu : $O(nlogn + 2^{\frac{1}{\varepsilon}})$

5 Implementacja

```
1
    public void ptasFunction()
    {
        bool [] tmp = new bool[tasksCount];

        Array.Sort(tasks);
        Array.Reverse(tasks);

        best = getTime(tmp, k);
        while (getPermutation(tmp))
11      {
            int tmpTime = getTime(tmp, k);
            if (best > tmpTime)
            {
                best = tmpTime;
                Array.Copy(tmp, optimal, k);
            }
        }


21      for (int i = k; i < tasksCount; i++)
        {
            if (!freeCpu(i + 1))
            {
                optimal[i] = true;
            }
        }
    }


31  private bool getPermutation(bool[] tab)
    {
        for(int i=0; i < k; i++)
        {
            if (tab[i])
            {
                tab[i] = false;
            }
            else
            {
41              tab[i] = true;
```

```
                return true;
            }
        }
        return false;
}
```

6 Testy

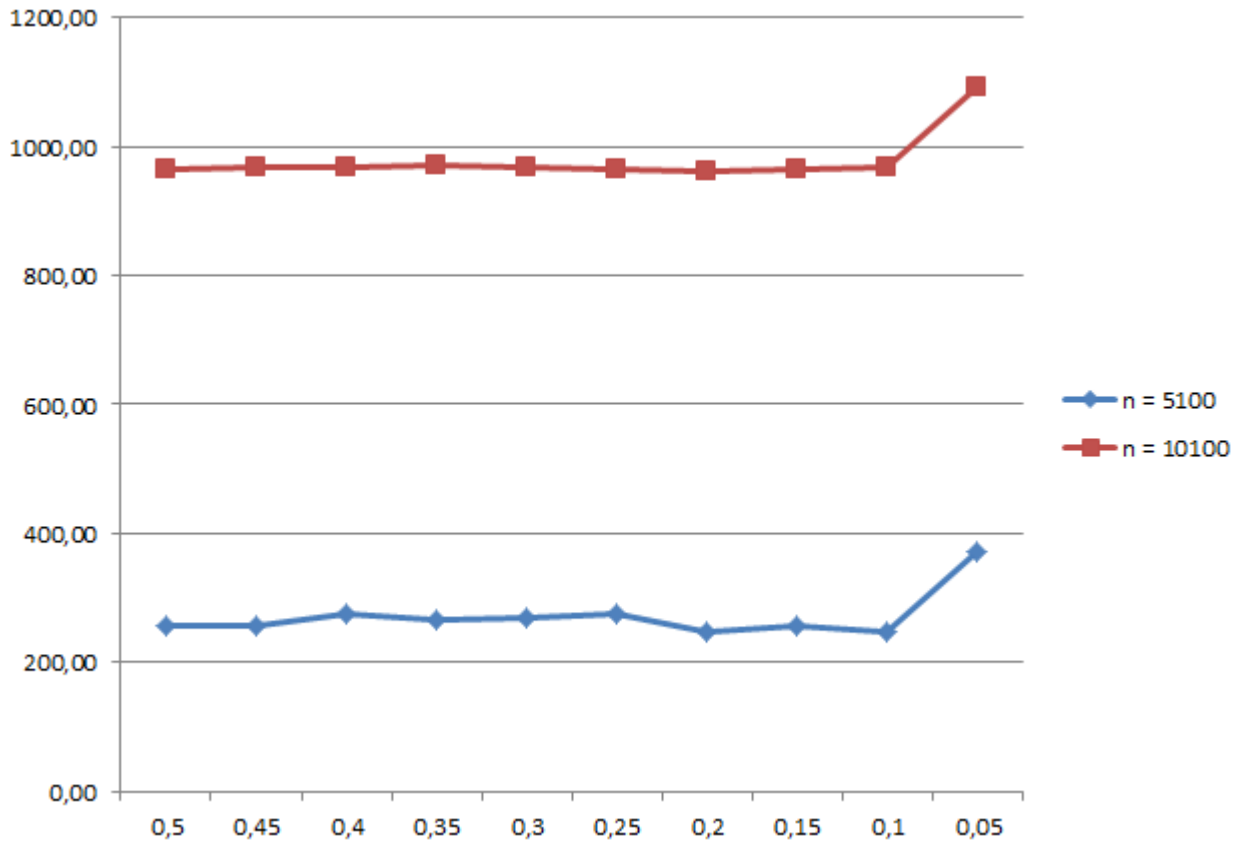
W celu przeprowadzenia dokłądnej analizy wykonanych zostało szereg testów badających czas wykonania algorytmu w zależności od parametru n i ε. W przypadku badania czasu wykonania kazdej instancji wykonano po 3 testy, a prezentowany wynik jest ich średnią arytmetyczną. Mimo iż program posiada graficzny interfejs użytkownika, moduł testowy wykonany został jako aplikacja konsolowa.

n	ε	time[ms]
100	0,5	0
100	0,45	0
100	0,4	0
100	0,35	0
100	0,3	0
100	0,25	0
100	0,2	0
100	0,15	0
100	0,1	0
100	0,05	126
5100	0,5	243,333
5100	0,45	243
5100	0,4	243
5100	0,35	242,333
5100	0,3	242
5100	0,25	242,333
5100	0,2	243
5100	0,15	242
5100	0,1	243
5100	0,05	366

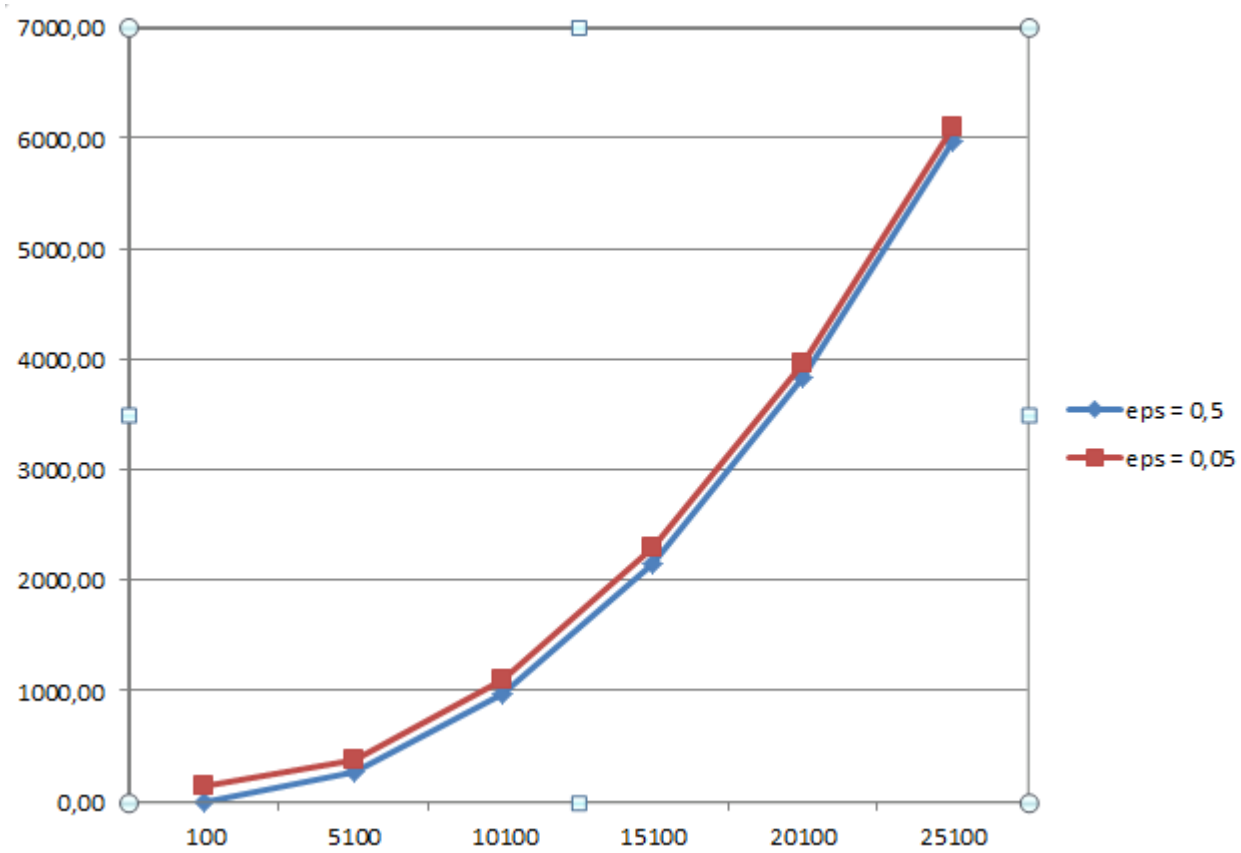
10100	0,5	954,333
10100	0,45	953,333
10100	0,4	952,333
10100	0,35	952
10100	0,3	950,666
10100	0,25	950,666
10100	0,2	957,666
10100	0,15	1038,33
10100	0,1	952
10100	0,05	1080
15100	0,5	2149,66
15100	0,45	2129
15100	0,4	2128
15100	0,35	2127,33
15100	0,3	2125,33
15100	0,25	2128
15100	0,2	2129,33
15100	0,15	2128,66
15100	0,1	2328,33
15100	0,05	2253
20100	0,5	3769,66
20100	0,45	3779,33
20100	0,4	3769,66
20100	0,35	3767,33
20100	0,3	3766,33
20100	0,25	3773
20100	0,2	3774,66
20100	0,15	3780
20100	0,1	3776,33
20100	0,05	3894,33
25100	0,5	6131
25100	0,45	6044,66
25100	0,4	5898
25100	0,35	5883
25100	0,3	5900,66
25100	0,25	5879,66
25100	0,2	5877
25100	0,15	5888

25100	0,1	5886
25100	0,05	6003,66

Tabela 1: Wynik zależny od ε i n



Rysunek 1: Wykres zależności czasu od ε



Rysunek 2: Wykres zależności czasu od n

Analizując powyższe dane można wywnioskować, iż największy wpływ na czas wykonywania algorytmu ma rozmiar instancji problemu n. Na Rysunku 1 przedstwiono zależność pomiędzy czasem wykonania algorytmu, a wartością ε dla instancji o stałym rozmiarze n. Wykres obu funkcji przez dłuższy okres jest linią poziomą, dopiero dla ostaniego parametru nieznacznie wzrasta,lecz wciąż liniowo. Dla wartości parametru $\varepsilon = 0.05$ długość problemu, którą należy uszeregować dokładnie wzrasta znaczniej, niż w poprzednich przykładach. Z powodu zbyt długiego czasu wykonywania, nie przeprowdzone zostały testy dla $\varepsilon = 0.01$.

Analizując wykres przedstawiony na Rysunku 2 można dojść do wniosku, iż bez znaczącej zmiany parametru ε np. 0.01 czas wykonywania algorytmu dla tych samych instancji problemu, ale większych parametrach ε jest bardzo

zbliżony.

7 Wnioski

Algorytm *PTAS* jest jednym z łatwiejszych algorytmów obliczania przybliżonej wartości dla skomplikowanych problemów *NPh i sNPh*. Jednak by uzyskać wyniki z zadowalającą dokładnością, należy użyć małych parametrów ε, co wiąże się z dużym nakładem czasu na obliczanie dokładnych wyników i zbliża algorytm do przeglądu zupełnego (w tym przypadku).

Głównym obszarem zastosowań algorytmu *PTAS* są sytuacje, gdy możemy sobie pozwolić na dość duży błąd rozwiązania.