

Projektowanie efektywnych algorytmów

Autor:

Tymon Tobolski (181037)

Jacek Wieczorek (181043)

Prowadzący:

Prof. dr hab. inż Adam Janiak

Wydział Elektroniki

III rok

Cz TN 13.15 - 15.00

20 listopada 2011

1 Cel projektu

Celem projektu jest zaimplementowanie i przetestowanie metaheurystycznego algorytmu symulowanego wyżarzania dla problemu szeregowania zadań na jednym procesorze przy kryterium minimalizacji ważonej sumy opóźnień zadań.

2 Opis problemu

Jednoprocesorowy problem szeregowania zadań przy kryterium minimalizacji ważonej sumy opóźnień zadań.

Danych jest n zadań (o numerach od 1 do n), które mają być wykonane bez przerwań przez pojedynczy procesor, mogący wykonywać co najwyżej jedno zadanie jednocześnie. Każde zadanie j jest dostępne do wykonania w chwili zero, do wykonania wymaga $p_j > 0$ jednostek czasu oraz ma określoną wagę (priorytet) $w_j > 0$ i oczekiwany termin zakończenia wykonywania $d_j > 0$. Zadanie j jest spóźnione, jeżeli zakończy się wykonywać po swoim terminie d_j , a miarą tego opóźnienia jest wielkość $T_j = \max(0, C_j - d_j)$, gdzie C_j jest terminem zakończenia wykonywania zadania j . Problem polega na znalezieniu takiej kolejności wykonywania zadań (permutacji) aby zminimalizować kryterium $TWT = \sum_{j=1}^n w_j T_j$.

3 Opis algorytmu

Symulowane wyżarzanie to algorytm heurystyczny przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych. Sposób działania algorytmu jest analogią do zjawiska wyżarzania w metalurgii.

Przebieg algorytmu :

```
1  t = Tmax
   old = S_0
   best = old

   while t < Tmin
       new = S(old)
       if F(new) < F(old)
           old = new
           if F(new) < F(best)
               best = new
       end
11  else if rand() < P(old, new, t)
```

```

        old = new
    end
    t = T(t)
end

```

4 Implementacja

Jezykiem implementacji algorytmu jest *Scala* w wersji 2.9.1 działająca na *JVM*.

Algorytm oparty został na funkcji rekurencyjnej (rekurencja ogonowa) implementującej symulowane wyżarzanie. W celu bezpiecznego zrównoleglenia uruchamiania programy na wiele wątków i tym samym przyspieszyć wyliczanie skorzystaliśmy z programowania funkcyjnego.

Funkcja :

```

def apply(s0: T) = {
    3      def inner(bestState: T, oldState: T, t: Double): T = {
        if(t < Tmin) oldState
        else {
            val newState = S(oldState)
            val (a,b) = if(F(newState) < F(oldState)){
                if(F(newState) < F(bestState)) (newState, newState)
                else (bestState, newState)
            } else if (math.random < P(oldState, newState, t)){
                (bestState, newState)
            } else {
                (bestState, oldState)
            }
            13      inner(a, b, T(t))
        }
    }
    inner(s0, s0, Tmax)
}

```

gdzie :

- S - funkcja generująca nowy losowy stan na podstawie podanego
- F - funkcja celu/kosztu

5 Testy

Testy algorytmu symulowanego wyżarzania przeprowadzone zostały dla trzech zestawów testów o różnym rozmiarze problemu n , każdy składający się ze 125 instancji. Parametry podstawowe jak T_{min} i T_{max} w przypadku każdego testu były takie same. Zmieniany natomiast był parametr T_d .

Jako wyniki testów przedstawiamy średni czas liczenia wszystkich instancji dla danego rozmiaru problemu - \bar{t} , a także średni błąd względny rozwiązań dla każdej instancji - \bar{x} . Według wzoru :

$$\bar{t} = \frac{\sum_{j=0}^n \frac{\sum_{i=1}^k t_i}{k}}{n} \quad (1)$$

$$\bar{x} = \frac{\sum_{j=0}^n \frac{\sum_{i=1}^k x_i}{k}}{n} \quad (2)$$

gdzie :

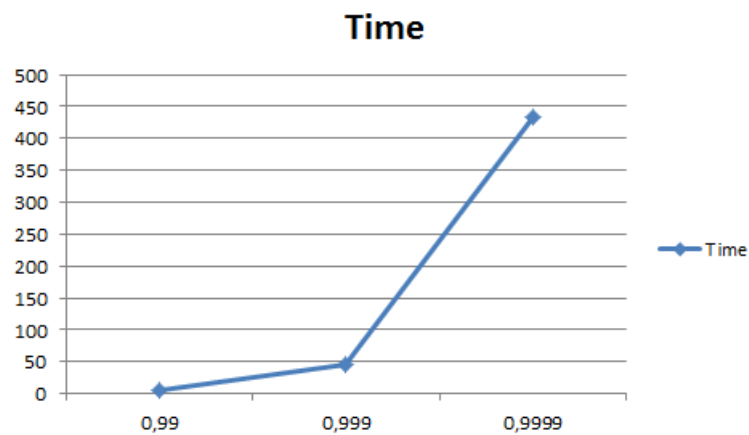
- k - ilość rozwiązań w instancji
- n - ilość instancji danego problemu

Parametry niezmiennie :

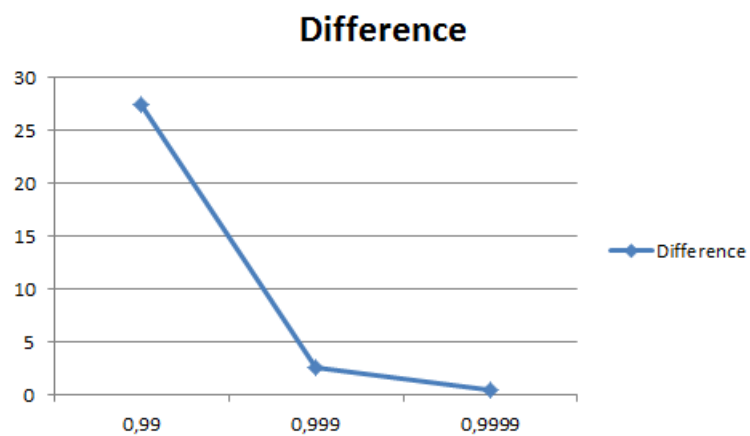
- $T_{min} = 0.01$
- $T_{max} = 100$

5.1 $n = 40$

T_d	Time	Difference
0,99	5,78	27,41
0,999	46,26	2,52
0,9999	434,52	0,43



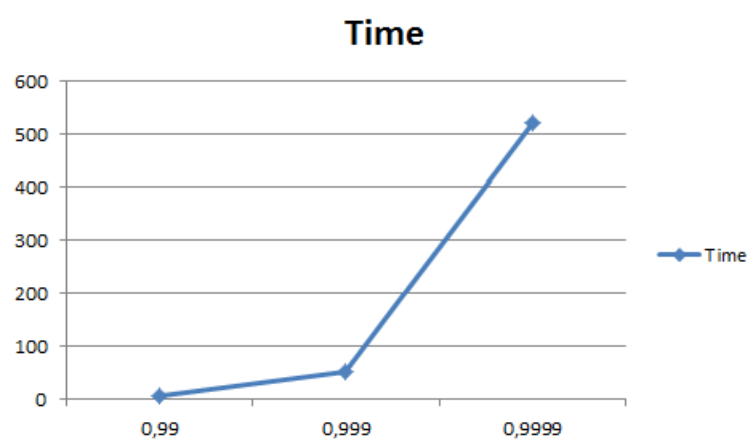
Rysunek 1: Czas rozwiązywania w zależności od parametru T_d



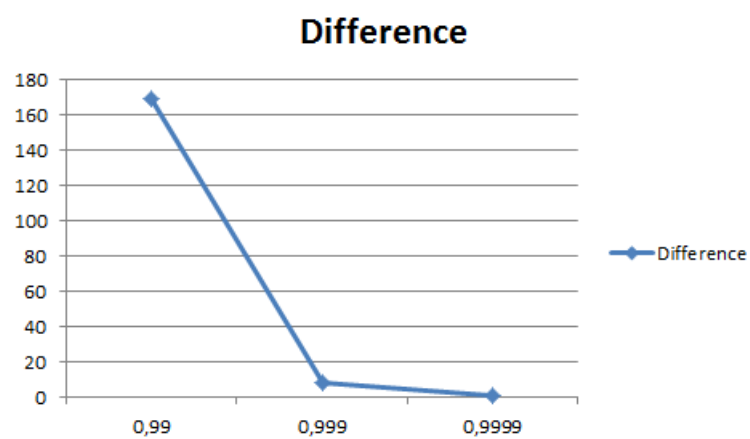
Rysunek 2: Błąd względny rozwiązywania w zależności od parametru T_d

5.2 $n = 50$

T_d	Time	Difference
0,99	6,63	169,02
0,999	53,09	8,47
0,9999	519,04	0,95



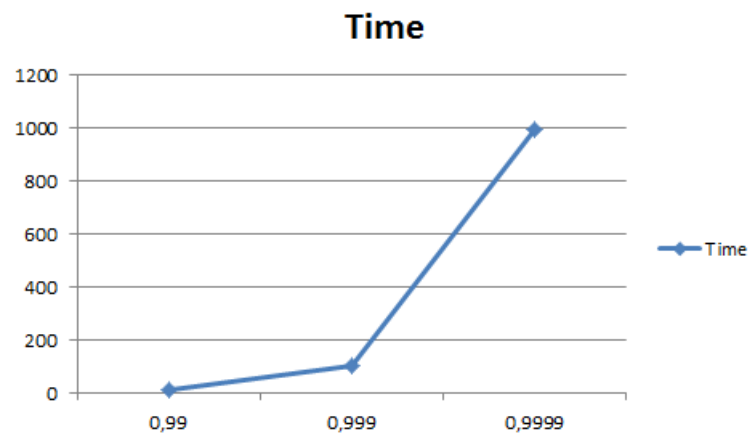
Rysunek 3: Czas rozwiązywania w zależności od parametru T_d



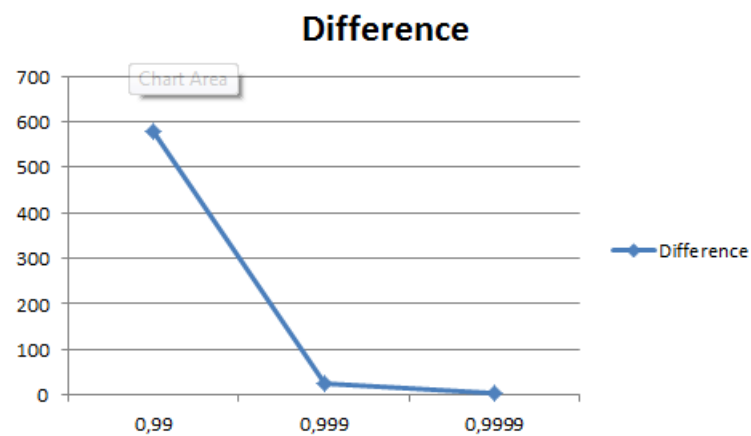
Rysunek 4: Błąd względny rozwiązywania w zależności od parametru T_d

5.3 $n = 100$

T_d	Time	Difference
0,99	12,4	579,97
0,999	104,18	23,43
0,9999	990	3,42

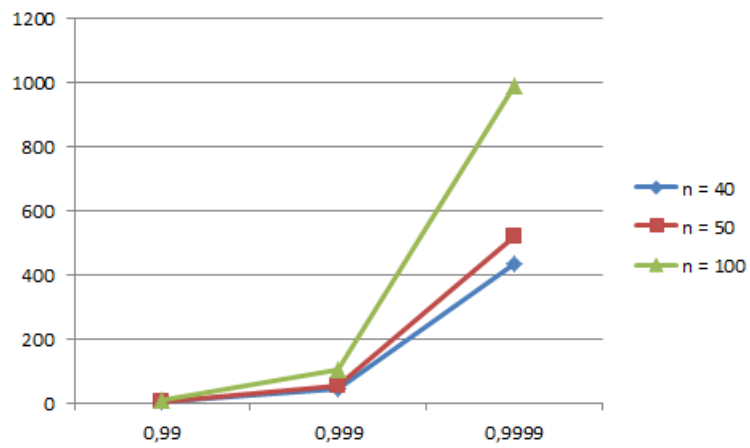


Rysunek 5: Czas rozwiązywania w zależności od parametru T_d

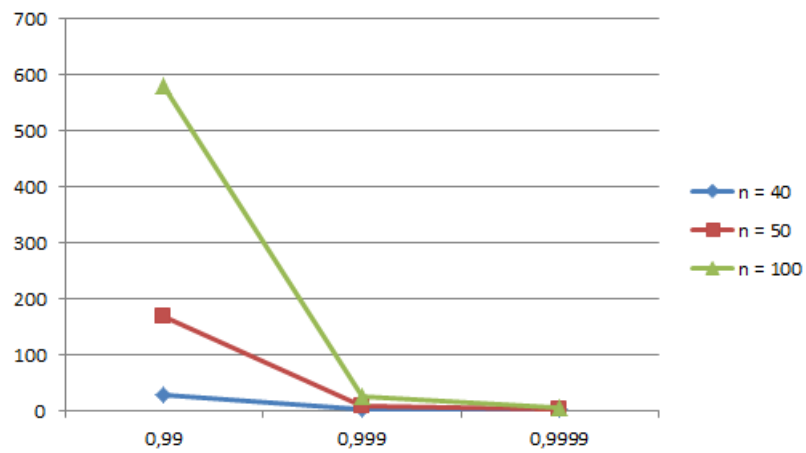


Rysunek 6: Błąd względny rozwiązywania w zależności od parametru T_d

5.4 Porównanie wszystkich zadań



Rysunek 7: Czas rozwiązywania w zależności od parametru T_d



Rysunek 8: Błąd względny rozwiązywania w zależności od parametru T_d

6 Wnioski

Analizując wyniki testów łatwo zauważyć, że rozmiar instancji ma znaczny wpływ na czas działania algorytmu. Pomimo tej samej liczby iteracji (dla ustalonego T_d i zmiennego n) samego algorytmu dużo czasu zajmuje obliczanie funkcji celu, która jest liniowo zależna od rozmiaru instancji. Na czas algorytmu ma również wpływ parametr T_d określający jak szybko zmienia się temperatura. Ilość iteracji algorytmu (zależna od parametru T_d , stałe n) ma również wpływ na jakość wyników. Im parametr T_d jest większy, tym wynik dokładniejszy. Niestety zwiększa to ilość samych iteracji dla danego problemu :

T_d	Ilość iteracji
0,99	917
0,999	9206
0,9999	92099

Jak widać na wykresie dla $T_d = 0.99$ i $n = 100$ algorytm uzyskał średnią względną różnicą od optymalnego wyniku na poziomie 500%. Analizując poszczególne instancje błąd względny wahał się od 10 do nawet 2500%.

Algorytm symulowanego wyżarzania pozwala na znacznie szybsze wyznaczenie dokładnego lub zbliżonego do dokładnego rozwiązania niż przegląd zupełny. Wiąże się to jednak z koniecznością dobrania odpowiednich parametrów, co nie jest zadaniem łatwym. W miarę poprawy wyników poprzez dobierane parametry, wzrasta czas wykonania algorytmu. W celu obliczenia problemu, musimy odpowiedzieć sobie na pytanie, jak dokładne rozwiązanie nas interesuje i ile czasu możemy na nie poświęcić.