Projektowanie efektywnych algorytmów

Autor:
Tymon Tobolski (181037)

Prowadzący:
Mgr inż. Karolina Mokrzysz

Wydział Elektroniki
III rok
Pn TP 11.15 - 1.00

13 stycznia 2012

1 Cel projektu

Celem projektu jest implementacja algorytmu rozwiązującego NP-trudny problem szeregowania n zadań na dwóch równoległych maszynach ($P_2||C_{max}$) przy pomocy wielomianowego schematu aproksymacyjnego PTAS.

2 Wielomianowy schemat aproksymacyjny

Wielomianowy schemat aproksymacyjny (ang. PTAS - Polynomial-time approximation scheme) jest algorytmem aproksymacyjnym, pozwalającym na uzyskanie dowolnie dobrego rozwiązania przybliżonego danego problemu w czasie wielomianowym.

3 Algorytm

Zaimplementowany algorytm składa się z kilku kroków:

- Posortowanie zadań malejąco po czasie wykonania
- Obliczenie wartości optymalnej
- Obliczenie górnego ograniczenia na podstawie wartości optymalnej i zadanego parametru ε
- Podzielenie zadań na dwie grupy względem wartości górnego ograniczenia
- Przyporządkowanie zadania z wartościami większymi za pomocą algorytmu dokładnego (przegląd zupełny)
- Przyporządkowanie pozostałych zadań za pomocą algorytmu przybliżonego (LPT)

4 Implementacja algorytmu

```
import Data.List
import Data.Function
import qualified Data.Text as T
import System (getArgs)
import Debug.Trace (trace)
import Control.Arrow

data Task = Task {index::Int, p::Int}

instance Show Task where
    show t = (show $ index t) ++ "(" ++ (show $ p t) ++ ")"

showTask :: Task -> String
showTask t = show [index t, p t]

readTask :: String -> Int -> Task
readTask s i = Task i (read s)


-- LPT
lpt :: Int -> Int -> [Task] -> (Int, [(Task, Int)])
lpt p0 p1 tasks = case (foldl f (p0, p1, []) tasks) of (p0, p1, sch)
    -> (max p0 p1, reverse sch)
  where f (p0, p1, sch) t | p0 < p1 = (p0 + (p t), p1, (t, 0) : sch)
        f (p0, p1, sch) t           = (p0, p1 + (p t), (t, 1) : sch)


-- Bruteforce
bruteforce :: [Task] -> (Int, [(Task, Int)])
bruteforce tasks = minimumBy (compare `on` fst) $ map (time >>= (,)) $
     map (zip tasks) $ comb (length tasks)

comb :: Int -> [[Int]]
comb n = inner 0 n [[]]

inner :: Int -> Int -> [[Int]] -> [[Int]]
inner k n xss | k == n = xss
inner k n xss          = inner (k+1) n $ xss >>= \xs -> [0:xs, 1:xs]

time :: [(Task, Int)] -> Int
time xs = (uncurry max) $ foldl f (0,0) xs
  where f (p0, p1) (t, 0) = (p0 + (p t), p1)
        f (p0, p1) (t, 1) = (p0, p1 + (p t))

-- Utils
machines :: [(Task, Int)] -> ([Task], [Task])
machines tasks =  (map fst *** map fst) $ partition ((0==) . snd)
    tasks

sump :: [Task] -> Int
sump tasks = foldl (\a t -> a + (p t)) 0 tasks

lb tasks = fromIntegral $ div (sum $ map p tasks) 2

run tasks e = let
  b =  (e * (lb tasks)) :: Float
  (ax, bx) = partition (\t -> (fromIntegral $ p t) > b) tasks
  (p0, p1) = machines $ snd $ bruteforce ax
  (m0, m1) = machines $ snd $ lpt (sump p0) (sump p1) bx
```

```
      in (b, e, p0, m0, p1, m1)

    disp (k, e, p0, m0, p1,m1) = (show k) ++ " " ++ (show e) ++ "\n" ++ "
        m0: " ++ (show $ sump (p0++m0)) ++ " => " ++ (show p0) ++ " --- "
        ++ (show m0) ++ "\n" ++ "m1: " ++ (show $ sump (p1++m1)) ++ " => "
         ++ (show p1) ++ "---" ++ (show m1) ++ "\n"

61  main = do
        args <- System.getArgs
        x <- readFile "data.txt"
        let tasks = reverse $ sortBy (compare `on` p) $ map (uncurry
            readTask) (zip (take (read (args !! 0) :: Int) (lines x))
            [1..] )
        putStrLn $ show $ map (\t -> (run tasks $ (read (args !! 1) ::
            Float))) [1..100000]
```

5 Środowisko testowe

Program testowy został napisany w języku Haskell przy pomocy kompilatora GHC w wersji 7.0.3.

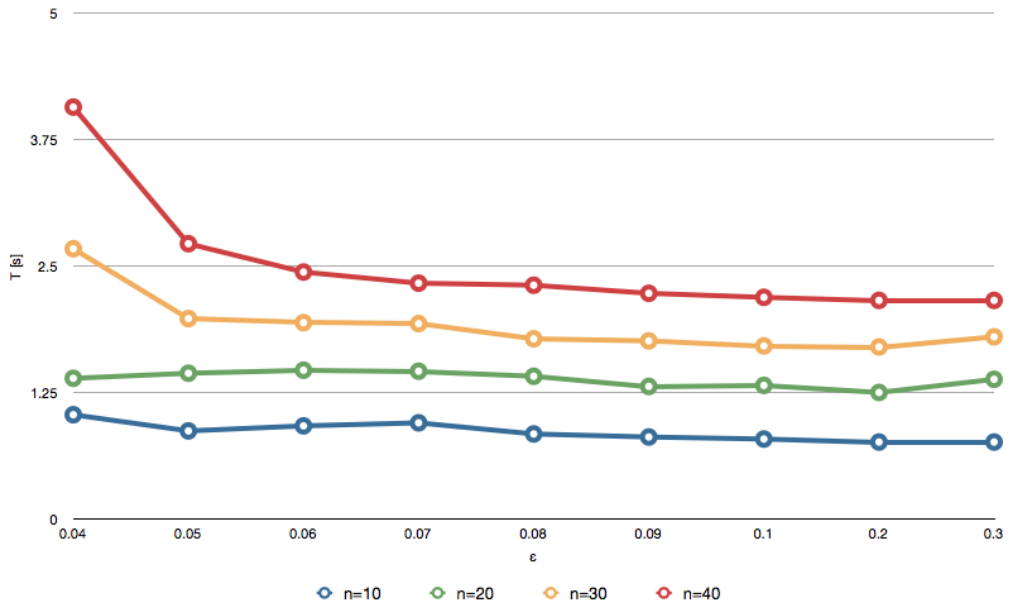
Testy zostały przeprowadzone na komputerze o poniższych parametrach:

- System operacyjny: Max OS X 10.7.1
- Procesor: 2.4 GHz Intel Core 2 Duo
- Pamięc RAM: 8 GB 1067 MHz

6 Wyniki

Uśredniony czas (w sekundach) wykonania 100.000 szeregowań w zależności od ilości zadań n oraz parametru ε:

ε \ n	10	20	30	40
0.04	1.03	1.39	2.67	4.07
0.05	0.87	1.44	1.98	2.72
0.06	0.92	1.47	1.94	2.44
0.07	0.95	1.46	1.93	2.33
0.08	0.84	1.41	1.78	2.31
0.09	0.81	1.31	1.76	2.23
0.10	0.79	1.32	1.71	2.19
0.20	0.76	1.25	1.70	2.16
0.30	0.76	1.38	1.80	2.16



Rysunek 1: Czas potrzebny do wyznaczenia przybliżonego uszeregowania w zależności od n i ε

Na Rysunku 1 przedstawiono zależność między parametrem ε, a czasem wykonania szeregowania dla czterech różnych długości listy zadań. Analizując wyniki testów można wnioskować, iż największy wpływ na czas szeregowania ma ilosc zadań. Na wykresie widać także, że czas szeregowania dla ε należącego do przedziału $(0.06 - 0.3)$ nie różni się znacząco. Dopiero dla wartości $\varepsilon < 0.05$ widać znaczne spowolnienie algorytmu.

7 Wnioski

W opracowanych przypadku PTAS jest swego rodzaju kompromisem pomiędzy przeglądem zupełnym, a algorytmem przybliżonym (LPT). Dzięki zastosowaniu schematu aproksymacyjnego uzyskane wyniki są lepsze niż przy zastosowaniu tylko algorytmu przybliżonego LPT lecz nadal gorsze niż w przypadku pełnego algorytmu dokładnego.

Wielomianowy schemat aproksymacyjny jest stosunkowo łatwą do zaimplementowania metodą uzyskiwania przybliżonych wyników skomplikowanych problemów optymalizacyjnych. Jak w wiekszości algorytmów przybliżonych

rezultaty algorytmu zależą w dużej części od zadanych parametrów. W celu uzyskania dobrych wyników należy zadać małą wartość parametru ε, co oznacza dłuższy czas wykonania algorytmu i zbliża całość (w opracowanych przypadku) do przeglądu zupełnego.