Tomasz Jankowiak 249006 3 V 2021

Sterowanie procesami dyskretnymi

Laboratorium 3

Algorytm *NEH* dla $F*||C_{max}$

Prowadzący:	Dr inż. Mariusz Makuchowski
Termin zajęć:	Piątek, 11.15
Numer grupy laboratoryjnej:	E12-30a

1. Opis problemu przepływowego

Jednym z najbardziej klasycznych problemów szeregowania operacji produkcyjnych jest problem przepływowy. Jest on powszechnie uznawany za jeden z sztandarowych problemów teorii szeregowania, ponieważ modeluje wiele rzeczywistych procesów produkcyjnych, stanowiąc bazę do modelowania innych, bardziej skomplikowanych procesów.

Mamy do wykonania *n* zadań na *m* maszynach. Każde zadanie przechodzi kolejno i nieprzerwanie przez wszystkie maszyny od 1 do *m*. Dane są $p_{i,k} \geq 0$ czasy trwania zadania *i* na maszynie *k*. Analizujemy wersję permutacyjną, kolejność wykonywania zadań na każdej z maszyn jest taka sama. Szukamy uszeregowania minimalizującego czas wykonania wszystkich zadań, który jest równy czasowi zakończenia ostatniego zadania. W notacji trójpolowej problem można zapisać jako: $F^*||C_{max}$. Dla *m>2* jest to problem silnie NP-trudny[1], co praktycznie wyklucza istnienie dokładnego algorytmu wielomianowego rozwiązującego ten problem. Wymagania praktyczne (akceptowalny czas działania algorytmu) powodują, że zależy nam przede wszystkim na konstrukcji efektywnych algorytmów przybliżonych zamiast algorytmów dokładnych, które znajdują uszeregowanie optymalne. Jakość metody przybliżonej ocenia się na podstawie złożoności obliczeniowej algorytmu oraz dokładności przybliżenia. Aktualnie najlepszym znanym w literaturze algorytmem konstrukcyjnym dla problemu przepływowego jest algorytm *NEH*, zbudowany na bazie metody wstawień [2].

2. Opis algorytmu *NEH*

Algorytm *NEH*, oparty na technice wcięć, pozostaje do chwili obecnej bezspornym championem wśród konstrukcyjnych algorytmów przybliżonych dla problemu $F^*||C_{max}$. W praktyce dostarcza rozwiązań z błędem względnym przeciętnie 5-6% w odniesieniu do rozwiązania optymalnego. Zachowuje się znacznie lepiej dla problemów z $m>n$ niż w sytuacji $m \ll n$ [3].

Nazwa algorytmu jest akronimem pochodzącym od pierwszych liter nazwisk twórców – Muhammad **N**awaz, E Emory **E**nscore Jr, Inyong **H**am, którzy opracowali go w 1982 roku i zawarli w pracy [4].

Algorytm buduje rozwiązanie poprzez dokładanie kolejnych zadań do bieżącego częściowego uszeregowania. W pierwszej fazie dla każdego zadania liczymy jego priorytet – sumaryczny czas wykonania na wszystkich maszynach. Następnie należy kolejno brać zadania o największym priorytecie i wstawiać je w optymalne miejsce bieżącej kolejności. W tym celu trzeba sprawdzić i porównać czasy dla każdego miejsca, na którym może być dane zadanie. Ustalenie takiego priorytetu jest intuicyjne – zaczynamy szeregowanie od największego zadania, a następnie coraz mniejsze kolejne zadania staramy się upychać w dziury powstałe w uszeregowaniu, analogicznie jak w przypadku pakowania torby turystycznej elementami o różnych rozmiarach.

Zadanie *i* położyliśmy w najlepszym miejscu, ale to nie oznacza, że nie można zmniejszyć wartości funkcji celu *Cmax* poprzez zmianę położenia innych zadań w otrzymanej permutacji. Niemniej jednak jest to aktualnie najlepszy algorytm konstrukcyjny dla badanego problemu. Jednakże po odpowiednich jego modyfikacjach, wykorzystujących podaną właściwość, zawartych w pracy [2] jesteśmy w stanie znaleźć lepsze rozwiązanie przy takiej samej złożoności obliczeniowej [2].

Ponieważ wyliczenie czasu danego uszeregowania $C_{max}(\pi)$ jest złożoności $O(nm)$, złożoność obliczeniowa trywialnej implementacji algorytmu *NEH* jest rzędu $\boldsymbol{O(n^3m)}$. Możliwa jest akceleracja algorytmu *NEH*, która wykorzystuje podobieństwo kolejności, dla których sprawdzamy czasy średnio *n/2* razy. Dzięki operacjom na grafach algorytm ***quick NEH*** redukuje czas wyznaczenia $C_{max}(\pi)$ do $O(m)$ i osiąga złożoność obliczeniową $\boldsymbol{O(n^2m)}$.

---

[1] A. Wielgus, „*Rozwiązanie problemu przepływowego przy pomocy algorytmu uczenia ze wzmocnieniem*", 2007.

[2] E. Nowicki, M. Makuchowski, „*Metoda wstawień w klasycznych problemach szeregowania. Cz. I. Problem Przepływowy*", Komputerowo Zintegrowane Zarządzanie, tom II, 2001r., str. 113-122.

[3] C. Smutnicki – „*Algorytmy szeregowania zadań*", Oficyna Wydawnicza PWr, Wrocław 2012, str. 236-239.

[4] M. Nawaz, E.E. Enscore Jr., I. Ham, „*A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem*", OMEGA International Journal of Management Science, 11, 1983, str. 91-95.

3. Kod programu

```
#include <iostream>
#include <fstream>
using namespace std;

int Cmax(int N, int M, int*P, int*X)
{
    int T[100]; for(int i=0;i<=M;i++)T[i]=0;
    for(int n=0;n<N;n++)
        for(int m=1;m<=M;m++)
            T[m]=max(T[m-1],T[m])+P[(m-1)+X[n]*M];
     return T[M];
 }

 int Neh(int N, int M, int*P, int*X)
 {
     //etap 1: w X bedzie kolejnosc ukladania (wg wag rosnaco)
     int *w = new int[N];
     for(int n=0;n<N;n++){
         X[n]=n; //tymczasowa kolejnosc 123...
         w[n]=Cmax(1,M,P,&n); //policzenie wag kazdego zadania
     }
     for(int b=0;b<N-1;b++) for(int a=0;a<N-1;a++) // sortowanie babelkowe
         if(w[a]<w[a+1]){swap(w[a],w[a+1]);swap(X[a],X[a+1]);}
     delete[]w;
     //etap 2
     for(int n=0; n<N; n++){
         int best=-1, cmax=999999999;
         for(int p=n;p>=0;p--){
             int tmp = Cmax(n+1,M,P,X);
             if(tmp<=cmax){best=p;cmax=tmp;}
             if(p)swap(X[p],X[p-1]);
         }
         for(int i=0;i<best;i++)swap(X[i],X[i+1]);
     }
     return Cmax(N,M,P,X);
 }


 int main()
 {
     int N,M,P[10000],X[1000];
     string data4_3[] = {"data.w:", "data.000:"};
     string data20_5[] = {"data.001:", "data.002:", "data.003:", "data.004:",
 "data.005:", "data.006:", "data.007:", "data.008:", "data.009:", "data.010:"};
     string data20_10[] = {"data.011:", "data.012:", "data.013:", "data.014:",
 "data.015:", "data.016:", "data.017:", "data.018:", "data.019:", "data.020:"};
     string data20_20[] = {"data.021:", "data.022:", "data.023:", "data.024:",
 "data.025:", "data.026:", "data.027:", "data.028:", "data.029:", "data.030:"};
     string data50_5[] = {"data.031:", "data.032:", "data.033:", "data.034:",
 "data.035:", "data.036:", "data.037:", "data.038:", "data.039:", "data.040:"};
     string data50_10[] = {"data.041:", "data.042:", "data.043:", "data.044:",
 "data.045:", "data.046:", "data.047:", "data.048:", "data.049:", "data.050:"};
     string data50_20[] = {"data.051:", "data.052:", "data.053:", "data.054:",
 "data.055:", "data.056:", "data.057:", "data.058:", "data.059:", "data.060:"};
     string data100_5[] = {"data.061:", "data.062:", "data.063:", "data.064:",
 "data.065:", "data.066:", "data.067:", "data.068:", "data.069:", "data.070:"};
     string data100_10[] = {"data.071:", "data.072:", "data.073:", "data.074:",
 "data.075:", "data.076:", "data.077:", "data.078:", "data.079:", "data.080:"};
     string data100_20[] = {"data.081:", "data.082:", "data.083:", "data.084:",
 "data.085:", "data.086:", "data.087:", "data.088:", "data.089:", "data.090:"};
     string data200_10[] = {"data.091:", "data.092:", "data.093:", "data.094:",
 "data.095:", "data.096:", "data.097:", "data.098:", "data.099:", "data.100:"};
     string data200_20[] = {"data.101:", "data.102:", "data.103:", "data.104:",
 "data.105:", "data.106:", "data.107:", "data.108:", "data.109:", "data.110:"};
     string data500_20[] = {"data.111:", "data.112:", "data.113:", "data.114:",
 "data.115:", "data.116:", "data.117:", "data.118:", "data.119:", "data.120:"};
```

```
55.
56.      string data_all[] = {"data.000:", "data.001:", "data.002:", "data.003:",
   "data.004:", "data.005:", "data.006:", "data.007:", "data.008:", "data.009:",
   "data.010:",
57.                           "data.011:", "data.012:", "data.013:", "data.014:",
   "data.015:", "data.016:", "data.017:", "data.018:", "data.019:", "data.020:",
58.                           "data.021:", "data.022:", "data.023:", "data.024:",
   "data.025:", "data.026:", "data.027:", "data.028:", "data.029:", "data.030:",
59.                           "data.031:", "data.032:", "data.033:", "data.034:",
   "data.035:", "data.036:", "data.037:", "data.038:", "data.039:", "data.040:",
60.                           "data.041:", "data.042:", "data.043:", "data.044:",
   "data.045:", "data.046:", "data.047:", "data.048:", "data.049:", "data.050:",
61.                           "data.051:", "data.052:", "data.053:", "data.054:",
   "data.055:", "data.056:", "data.057:", "data.058:", "data.059:", "data.060:",
62.                           "data.061:", "data.062:", "data.063:", "data.064:",
   "data.065:", "data.066:", "data.067:", "data.068:", "data.069:", "data.070:",
63.                           "data.071:", "data.072:", "data.073:", "data.074:",
   "data.075:", "data.076:", "data.077:", "data.078:", "data.079:", "data.080:",
64.                           "data.081:", "data.082:", "data.083:", "data.084:",
   "data.085:", "data.086:", "data.087:", "data.088:", "data.089:", "data.090:",
65.                           "data.091:", "data.092:", "data.093:", "data.094:",
   "data.095:", "data.096:", "data.097:", "data.098:", "data.099:", "data.100:",
66.                           "data.101:", "data.102:", "data.103:", "data.104:",
   "data.105:", "data.106:", "data.107:", "data.108:", "data.109:", "data.110:",
67.                           "data.111:", "data.112:", "data.113:", "data.114:",
   "data.115:", "data.116:", "data.117:", "data.118:", "data.119:", "data.120:"};
68.
69.      for(int d=0;d<10;d++)
70.      {
71.          string data = data20_10[d];
72.          ifstream f("data.txt");
73.          string s; while(s!=data)f>>s;
74.          f>>N>>M; for(int i=0;i<N*M;i++) f>>P[i]; f.close();
75.
76.          cout<<endl<<data<<endl;
77.          cout<<N<<" "<<M<<endl; // wypisanie wczytanej liczby zadan i maszyn
78.        /*  for(int n=0;n<N;n++){ // wypisanie wczytanych danych
79.              for(int m=0;m<M;m++)
80.                  cout<<P[m+n*M]<<" "; //P[n][m]
81.              cout<<endl;
82.          }*/
83.
84.          cout<< Neh(N,M,P,X)<<endl; // wypisanie wartosci cmax
85.          for(int i=0;i<N;i++){ // wypisanie permutacji
86.              cout<<X[i]+1<<" ";
87.          }cout<<endl;
88.      }
89.
90.      //cin.get();
91.      return 0;
92. }
```

4. Działanie programu

Zaimplementowano trywialną wersję algorytmu *NEH*, bez akceleracji. Program wczytuje z pliku liczbę maszyn, zadań oraz czasy wykonania każdego zadania na poszczególnych maszynach. Następnie wywoływany jest algorytm *NEH*, który tworzy dobrą permutację oraz zwraca czas C_{max}.

Dla przykładowego zestawu danych z wykładu oraz 121 zestawów danych z pliku porównawczego (*http://mariusz.makuchowski.staff.iiar.pwr.wroc.pl/download/courses/sterowanie.procesami.dyskretnymi/lab.instrukcje/lab03.neh/neh.data/neh.data.txt*) program poprawnie zwrócił permutację oraz wartość celu C_{max}. Można zatem wyciągnąć wniosek, iż najprawdopodobniej program działa poprawnie.

5. Testy złożoności obliczeniowej

Dla danej liczby zadań i maszyn (z wyjątkiem *4/3*) program wykonywał algorytm na dziesięciu zestawach danych. W tabeli zawarto średni czas działania programu dla jednego zestawu w sekundach. W naiwnej implementacji spodziewamy się, że podwójny wzrost liczby zadań będzie powodować ośmiokrotny wzrost czasu (zależność sześcienna), natomiast dwukrotny wzrost liczby maszyn spowoduje dwukrotny przyrost czasu (zależność liniowa). Program kompilowano z flagą *-O3*, zapewniającą optymalizację i szybsze działanie. Czas mierzono za pomocą komendy *time* pisanej przed wywołaniem programu. Interesuje nas czas pracy procesora (ang. CPU time) zawarty w linii *user*. Warto pamiętać, że czasy dotyczą działania całego programu, łącznie z wczytaniem danych (ponieważ wszystkie zestawy danych są w jednym pliku, późniejsze zestawy powinny wczytywać się odrobinę dłużej niż początkowe, z drugiej strony różnica może być niezauważalna, a początkowe zestawy zawierają mniej danych, co działa na ich niekorzyść).

n \ m	3	5	10	20
4	0.0020	X	X	X
20	X	0.0005	0.0014	0.0018
50	X	0.0028	0.0030	0.0049
100	X	0.0069	0.0104	0.0154
200	X	X	0.0352	0.0602
500	X	X	X	0.7311

Widać, że dla większych danych program działa nawet nieco szybciej niż się spodziewaliśmy – dwukrotny wzrost liczby zadań lub maszyn powoduje mniejszy przyrost czasu działania programu niż oczekiwany. Jednak różnice nie są duże, a tendencja jest zachowana, co widać szczególnie przy większych danych – z 200/20 na 500/20 oraz z 200/10 na 200/20.

6. Podsumowanie i wnioski

Algorytm *QuickNEH* całkiem nieźle sobie radzi z problemem przepływowym, dając w czasie $O(n^2m)$ rozwiązanie różniące się od optymalnego zaledwie o 5-6%. Mowa tutaj o akceleracji algorytmu NEH, którą można uzyskać, działając na grafach. W tej pracy zaimplementowano zwykłą wersję algorytmu, która przez dłuższe liczenie funkcji celu C_{max} osiąga złożoność $O(n^3m)$.

7. Bibliografia

1. Eugeniusz Nowicki, Mariusz Makuchowski, „*Metoda wstawień w klasycznych problemach szeregowania. Cz. I. Problem Przepływowy*", Komputerowo Zintegrowane Zarządzanie, tom II, 2001r., str. 113-122, http://mariusz.makuchowski.staff.iiar.pwr.wroc.pl/download/publications/001.pdf.
2. Nawaz M., Enscore Jr. E.E., Ham I. – „*A Heuristic Algorithm for the m-Machine, n-Job Flow-shop Sequencing Problem*", *OMEGA* The International Journal of Management Science, tom 11, str. 91-95, Wielka Brytania 1983, http://mariusz.makuchowski.staff.iiar.pwr.wroc.pl/download/courses/sterowanie.procesami.dyskretnymi/lab.instrukcje/lab03.neh/neh.literatura/NEH.1982.pdf.
3. Czesław Smutnicki – „*Algorytmy szeregowania zadań*", Oficyna Wydawnicza PWr, Wrocław 2012, str. 236-239.
4. Mariusz Makuchowski – wykład „Akceleracja algorytmu *NEH*", 2019, http://mariusz.makuchowski.staff.iiar.pwr.wroc.pl/download/courses/sterowanie.procesami.dyskretnymi/wyk.slajdy/SPD_w04_QNEH.pdf.
5. Agnieszka Wielgus – „*Rozwiązanie problemu przepływowego przy pomocy algorytmu uczenia ze wzmocnieniem*", 2007, https://kcir.pwr.edu.pl/~witold/aiarr/2007_projekty/flow/#problem.