Politechnika
Wrocławska

Algorytm Neh z akceleracją

Autor: Jan Ploska

Kierunek: Informatyczne Systemy Automatyki

Semestr: VI

Prowadzący: dr. hab. inż. Mariusz Makuchowski

Opis problemu: Algorytm NEH (Nawaz-Enscore-Ham) jest heurystyczną metodą planowania produkcji, używaną do rozwiązywania problemów harmonogramowania przepływu prac w systemach produkcyjnych typu flow shop. W takim systemie zadania muszą przejść przez określoną sekwencję maszyn w tym samym porządku, a celem jest zminimalizowanie całkowitego czasu wykonania wszystkich zadań, nazywanego czasem przepływu.

Cel algorytmu Neh: Problem harmonogramowania przepływu prac (flow shop scheduling problem), polega na znalezieniu optymalnego porządku wykonywania zadań na zestawie maszyn, tak aby zminimalizować całkowity czas produkcji (makespan). Ten problem jest trudny do rozwiązania optymalnie (należy do problemów NP-trudnych), zwłaszcza gdy mamy do czynienia z dużą liczbą maszyn i zadań.

Działanie algorytmu Neh:

1. Sortowanie zadań: Na początku wszystkie zadania są sortowane malejąco według sumarycznego czasu pracy na wszystkich maszynach.

```
def max_sort_zadania():
    z= sorted_zadania = copy.deepcopy(zadania)
    for i in range(len(z)):
        for j in range(len(z) -1):
            if z[j+1].sum > z[j].sum:
                z[j], z[j+1] = z[j+1], z[j]
    return z
```

2. Konstrukcja harmonogramu: Zadania są dodawane jedno po drugim do harmonogramu początkowego, próbując je umieszczać w każdej możliwej pozycji.

```
for j in range(m):
    for i in range(n):
        if j == 0:
            if i != 0:
                cmax_tab[j][i] = cmax_tab[j][i-1] + cmax_tab[j][i]
        else:
            if i == 0:
                cmax_tab[j][i]= cmax_tab[j][i] + cmax_tab[j-1][i]
            else:
                cmax_tab[j][i] = max(cmax_tab[j][i] + cmax_tab[max(j-1, 0)][i],
                                     cmax_tab[j][i] + cmax_tab[j][max(i-1, 0)])
```

3. Wybór pozycji: Dla każdego nowo dodanego zadania wybierana jest taka pozycja w harmonogramie, która minimalizuje aktualny czas przepływu.

```
schedule= []
start= time.time()
for z in sorted_zadania:
    if len(schedule) == 0:
        schedule.append(z.id)
    else:
        best_index= 0
        best_cmax= math.inf
        for j in range(len(schedule) +1):
            schedule.insert(j, z.id)
            current_cmax= compute_cmax(schedule)
            del schedule[j]
            if current_cmax < best_cmax:
                best_cmax= current_cmax
                best_index= j
        schedule.insert(best_index, z.id)
```

4.

Algorytm Neh z akceleracją(QNeh): ta wersja algorytmu NEH, która wprowadza dodatkowe optymalizacje w celu przyspieszenia procesu obliczeniowego. Bazuje na podstawowej zasadzie NEH, czyli sortowaniu zadań według ich sumarycznego czasu pracy i sekwencyjnym dodawaniu ich do harmonogramu w celu minimalizacji maksymalnego czasu zakończenia (makespan). QNEH wprowadza jednak dodatkowe tablice obliczeniowe, które pozwalają na szybsze ocenienie wpływu wstawienia nowego zadania w każdą możliwą pozycję.

Działanie algorytmu QNeh:

1. Sortowanie zadań: Na początku wszystkie zadania są sortowane malejąco według sumarycznego czasu pracy na wszystkich maszynach.
2. Zadania są dodawane jedno po drugim do wstępnego harmonogramu. Przy każdym dodaniu korzystam z dwóch pomocniczych tablic (cmax_tab z przodu i reversed_cmax_tab z tyłu), które są generowane tylko raz dla aktualnego stanu harmonogramu. Te tablice reprezentują akumulacyjne czasy zakończenia pracy na każdej maszynie dla kolejnych zadań, obliczane od początku i od końca harmonogramu. Pozwala to na szybką ocenę wpływu wstawienia nowego zadania na każdą możliwą pozycję w harmonogramie, bez potrzeby wielokrotnego obliczania tych samych wartości. Dla każdego nowo dodanego zadania obliczany jest maksymalny czas zakończenia (cmax) dla każdej potencjalnej pozycji w harmonogramie, korzystając z wcześniej wspomnianych tablic, co znacząco przyspiesza i usprawnia proces wyboru optymalnej pozycji zadania.

```
r_cmax_tab= cmax_tab.copy()
for j in range(m):
    for i in range(n):
        if j == 0:
            if i != 0:
                cmax_tab[j][i] = cmax_tab[j][i-1] + cmax_tab[j][i]
                r_cmax_tab[m-1-j][n-1-i]=r_cmax_tab[m-1-j][n-1-i+1] +r_cmax_tab[m-1-j][n-1-i]
        else:
            if i == 0:
                cmax_tab[j][i]= cmax_tab[j][i] + cmax_tab[j-1][i]
                r_cmax_tab[m-1-j][n-1-i]=r_cmax_tab[m-1-j][n-1-i] +r_cmax_tab[m-1-j+1][n-1-i]
            else:
                cmax_tab[j][i] = max(cmax_tab[j][i] + cmax_tab[max(j-1, 0)][i],
                                     cmax_tab[j][i] + cmax_tab[j][max(i-1, 0)])
                r_cmax_tab[m-1-j][n-1-i] = max(r_cmax_tab[m-1-j][n-1-i] +r_cmax_tab[min(m-1-j+1, m-1)][n-1-i],
                                               r_cmax_tab[m-1-j][n-1-i] +r_cmax_tab[m-1-j][min(n-1-i+1, n-1)])
```

3. Wybór pozycji: Dla każdego nowo dodanego zadania wybierana jest taka pozycja w harmonogramie, która minimalizuje aktualny czas przepływu.

Porównanie czasów działania algorytmu Neh i Qneh: czasy podane są w sekundach.

Porównanie czasu działania Neh i Qneh			
m	**n**	**neh t**	**qneh t**
3	4	0,001	0,000
5	20	0,059	0,005
20	20	0,225	0,019
10	100	13,941	0,248
5	50	0,899	0,032
20	50	3,351	0,121
20	200	-	2,084
20	500	-	13,005

Wnioski:

- Moja implementacja QNeh znacząco przyśpieszyła działanie algorytmu
- Kod napisany jest w pythonie i czas działania dla wszystkich 120 instancji jest powyżej 12 sekund
- Niepotrzebnie za każdym razem buduje harmonogram od samego początku.