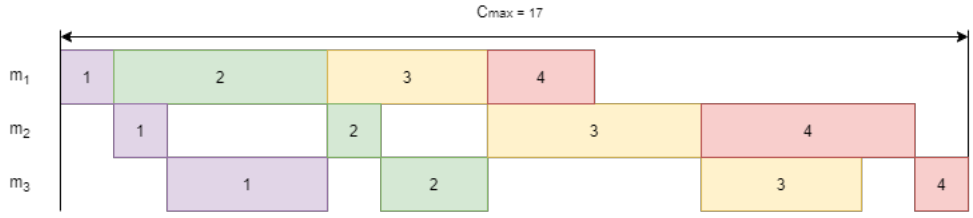
STEROWANIE PROCESAMI CIĄGŁYMI
Problem przepływowy - symulowane wyżarzanie

Prowadzący: dr inż. Paweł Drąg
Autorzy:
Emilia Szymańska 248975
Igor Zieliński 248944

10.06.2021r.

1 Opis problemu przepływowego

Problem przepływowy jest problemem wielomaszynowym. Do wykonania dostajemy zbiór zadań $J = (1, ..., n)$ na zbiorze maszyn $M = (1, ..., m)$. Każde z zadań musi po kolei przejść przez wszystkie maszyny od 1 do m tak jak jest to pokazane na rys. 1.



Rysunek 1: Przykładowe ułożenie zadań J=(1, 2, 3, 4) na maszynach M = (1, 2, 3).

Na wejściu podane są trzy parametry:

- $J = (1, ..., n)$ - zbiór zadań,
- $M = (1, ..., m)$ - zbiór maszyn
- p - macierz czasów p_{ij} wykonywania zadania J_i na maszynie M_j.

Wyjścia działania algorytmu obsługującego ten problem mogą być różnie zdefiniowane - może to być np. jest permutacja wykonania zadań π oraz moment maksymalny z możliwych terminów dostarczenia zadań C_{max}. Zaimplementowany został algorytm symulowanego wyżarzania do rozwiązania tego problemu.

2 Symulowane wyżarzanie

2.1 Opis teoretyczny, implementacja i przykład działania

Algorytm wymaga zdefiniowania kilku elementów: otoczenia (sąsiedztwa), funkcji akceptacji i schematu chłodzenia.

Sąsiedztwo danej permutacji zdefiniowaliśmy jako takie permutacje, w których zamienione są kolejności sąsiadujących z sobą zadań, to znaczy dla przykładu [1 2 3 4 5] do sąsiedztwa należą ciągi [2 1 3 4 5], [1 3 2 4 5], [1 2 4 3 5] oraz [1 2 3 5 4].

Funkcja akceptacji $\psi(\pi, \beta, t)$ zdefiniowana jest jako:

$$\psi(\pi, \beta, t) = exp\left(-\frac{C_{max}(\beta) - C_{max}(\pi)}{t}\right). \tag{1}$$

Schemat chłodzenia ma charakter geometryczny:

$$t_{k+1} = \alpha t_k, \tag{2}$$

gdzie $\alpha = 0.6$.

Dla zadanych wartości początkowych i warunku zatrzymania w pętli while powtarzaj następujące kroki spisane w pseudokodzie:

1. while i $\leqslant$ R do:
   (a) i $\longleftarrow$ i+1
   (b) randomly choose permutation π from the neighbourhood
   (c) if($C_{max}(\beta) < C_{max}(\pi*)$) do $\pi* \longleftarrow \beta$
   (d) if($C_{max}(\beta) < C_{max}(\pi)$) do $\pi \longleftarrow \beta$
   (e) else if ($\psi(\pi, \beta, t) > random[0, 1)$) do $\pi \longleftarrow \beta$
2. i $\longleftarrow$ 0
3. t $\longleftarrow \alpha$t

Parametr R jest najczęściej równy mocy zbioru będącego sąsiedztwem.

Implementacja algorytmu znajduje się poniżej.

Listing 1: Implementacja symulowanego wyżarzania.

```
double acceptance(const Matrix<int>& M, vector<int> pi, vector<int> beta, double T)
{
        return exp( -(Cmax(M, beta) - Cmax(M, pi)) / T);
}

double cooling( double T, int i)
{
        return T * 0.6;
}
```

```
pair<int, vector<int> > annealing(const Matrix<int>& M)
{
        srand(time(nullptr));
        double T = 1.0;

        int Cmax_star;
        vector<int> pi_star, pi;

        int tasks = M.size().first;

        for(int i = 0; i < tasks; i++) pi_star.push_back(i);

        random_shuffle(pi_star.begin(), pi_star.end());
        Cmax_star = Cmax(M, pi_star);
        pi = pi_star;
        for(int i = 0; i < 200* tasks; i++){
                int Cmax_pi = Cmax(M, pi);
                for(int j = 0; j < tasks; j++){
                        int swap_index = rand()%(tasks - 1);
                        vector<int> beta = pi;
                        swap(beta[swap_index], beta[swap_index + 1]);
                        int Cmax_tmp = Cmax(M, beta);
                        if(Cmax_tmp < Cmax_star){
                                Cmax_star = Cmax_tmp;
                                Cmax_pi = Cmax_tmp;
                                pi_star = beta;
                                pi = beta;
                        }
                        else if(Cmax_tmp < Cmax_pi){
                                Cmax_pi = Cmax_tmp;
                                pi = beta;
                        }
                        else{
                                if( acceptance(M, pi, beta, T)
                                    >= ((double) rand() / (RAND_MAX))){
                                        Cmax_pi = Cmax_tmp;
                                        pi = beta;
                                }
                        }
                }
                T = cooling(T, i);
        }
        return {Cmax_star, pi_star};
}
```

Poniżej przedstawione zostało przykładowe działanie symulowanego wyżarzania dla przypadku czterech zadań i trzech maszyn z rys. 1.

1. Rozpocznij działanie algorytmu: π_{start}: 2 1 4 3
   $C_{max,start}$: 18
   Initial T = 1
2. Wylosuj β z sąsiedztwa:
   β: 2 1 3 4
   $C_{max}(\beta)$: 17
3. Poszeregowanie lepsze niż $\pi*$: podmień $\pi* \longleftarrow \beta$.
4. Wylosuj β z sąsiedztwa:
   β: 2 1 4 3
   $C_{max}(\beta)$: 18
5. Poszeregowanie gorsze niż π, prawdopodobieństwo podmiany: 0.37.
6. Podmień $\pi \longleftarrow \beta$.
7. Wylosuj β z sąsiedztwa:
   β: 1 2 4 3
   $C_{max}(\beta)$: 18
8. Poszeregowanie gorsze niż π, prawdopodobieństwo podmiany: 1.
9. Podmień $\pi \longleftarrow \beta$.
10. Wylosuj β z sąsiedztwa:
    β: 1 4 2 3
    $C_{max}(\beta)$: 17
11. Poszeregowanie lepsze niż $\pi*$: podmień $\pi* \longleftarrow \beta$.
12. Nowe T = 0.6.
13. Wylosuj β z sąsiedztwa:
    β: 1 4 3 2
    $C_{max}(\beta)$: 16
14. Poszeregowanie lepsze niż $\pi*$: podmień $\pi* \longleftarrow \beta$.
15. Wylosuj β z sąsiedztwa:
    β: 1 4 2 3
    $C_{max}(\beta)$: 17
16. Poszeregowanie gorsze niż π, prawdopodobieństwo podmiany: 0.19.
17. Podmień $\pi \longleftarrow \beta$.
18. Nowe T = 0.36.
19. Wylosuj β z sąsiedztwa:
    β: 1 4 3 2
    $C_{max}(\beta)$: 16

20. Poszeregowanie lepsze niż $\pi*$: podmień $\pi* \longleftarrow \beta$.

21. Wylosuj β z sąsiedztwa:
    β: 1 3 4 2
    $C_{max}(\beta)$: 15

22. Poszeregowanie lepsze niż $\pi*$: podmień $\pi* \longleftarrow \beta$.

3 Wyniki i ich porównanie

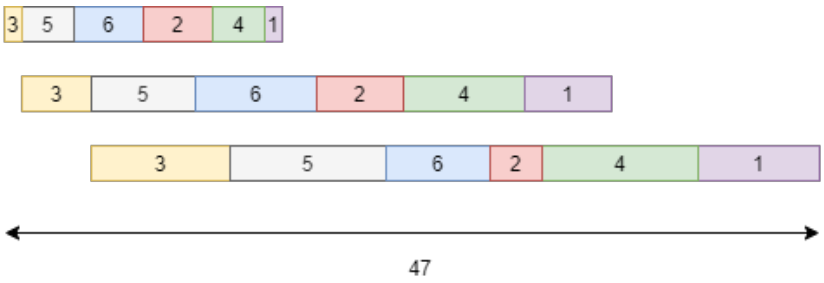
Dane testowe pobrane były ze strony `http://new.zsd.iiar.pwr.edu.pl/educ.php?lid=132&zid=NEH`.

3.1 Przykładowy wynik algorytmu

Na podstawie pierwszego zestawu testowego zostanie przedstawione działanie algorytmu opisanego w sekcji 2. Dane tekstowe przedstawiają się wówczas następująco:
6 3
1 5 7
4 5 3
1 4 8
7 3 9
3 6 9
4 7 6

Pierwsza linia zawiera dwie liczby całkowite: n oznaczającą liczbę zadań oraz m oznaczającą liczbę maszyn. Każda z następnych n linii zawiera m liczb, gdzie i-ta liczba w wierszu k oznacza czas wykonania zadania k na maszynie i.

Po przepuszczeniu przez algorytm NEH otrzymujemy poszeregowanie jak na rys. 2 odpowiadające kolejności zadań: 3 5 6 2 4 1 oraz wartości $C_{max} = 47$.



Rysunek 2: Uszeregowane zadania z pierwszego zbioru danych przez algorytm symulowanego wyżarzania.

W przypadku drugiego zestawu danych (widocznego niżej) symulowane wyżarzanie zwraca następującą kolejność wykonywania zadań: 5 6 3 9 10 4 2 1 7 8. W takim wypadku otrzymujemy $C_{max} = 619$.
10 5 84 53 77 29 39
95 63 57 68 69
26 85 15 73 74
50 8 7 5 52
4 11 92 94 77
9 69 51 37 4
57 71 33 49 12
35 55 27 1 17
79 10 84 36 1
17 91 64 62 21

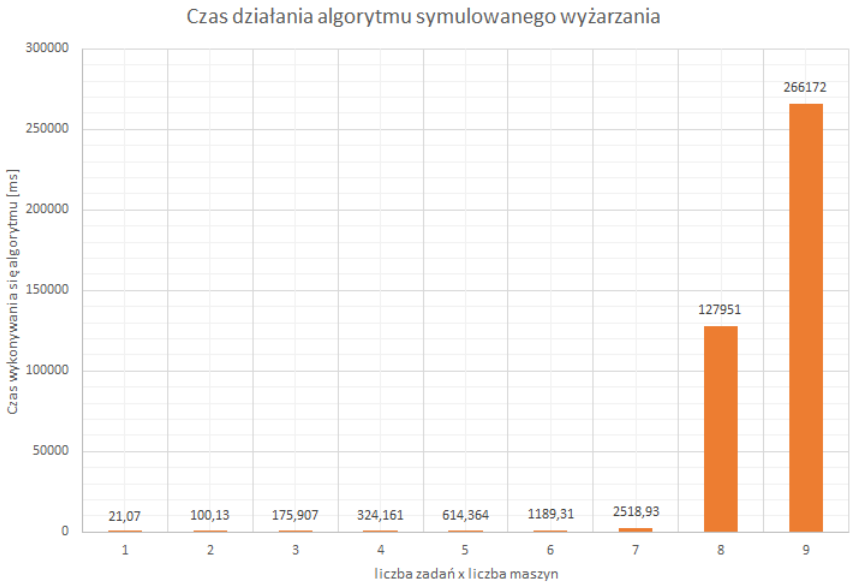
3.2 Wyniki na danych testowych

W tabeli 1 przedstawione zostały wartości otrzymanych C_{max} oraz czas wywoływania algorytmu dla poszczególnych zestawów danych, a na wykresie 3 porównano czasy wykonywania się algorytmów (z dokładnością do mikrosekund).

zestaw testowy	ilość zadań w zestawie (n x m)	C_{max}	czas działania programu $[ms]$
1	6x3	47	21.07
2	10x5	619	100.13
3	10x10	1287	175.907
4	10x20	1861	324.161
5	20x5	1138	614.364
6	20x10	1616	1189.31
7	20x20	2577	2518.93
8	100x10	5981	127951
9	100x20	7079	266172

Tabela 1: Wynik działania symulowanego wyżarzania.

Zestaw nr 1: 3 5 6 2 4 1
Zestaw nr 2: 5 6 3 9 10 4 2 1 7 8
Zestaw nr 3: 2 3 4 8 9 1 6 10 7 5
Zestaw nr 4: 6 4 7 9 10 2 5 3 8 1
Zestaw nr 5: 13 3 20 9 4 1 17 16 8 6 18 19 14 7 2 5 12 10 11 15
Zestaw nr 6: 17 13 5 3 8 15 16 12 1 4 18 14 9 7 10 6 19 2 20 11
Zestaw nr 7: 17 11 19 5 15 2 13 10 8 18 20 9 4 7 14 12 1 16 6 3
Zestaw nr 8: 58 1 30 54 65 21 94 38 37 77 82 41 56 19 62 6 83 40 13 95 100 57 64 23 48 66 44 26 71 63 16 88 28 72 35 3 52 96 84 4 45 61 98 34 49 99 59 86 32 33 46 68 36 75 22 78 87 29 79 89 85 81 91 73 97 25 2 11 47 80 76 17 69 60 8 27 15 42 24 18 74 55 93 9 31 67 20 92 53 51 39 10 7 50 14 70 90 5 12 43
Zestaw nr 9: 77 22 79 10 96 58 47 65 42 8 20 18 48 51 87 31 53 39 17 93 13 35 44 15 75 84 94 89 11 7 2 71 69 97 86 67 26 34 25 52 88 43 62 55 57 76 83 14 1 27 23 63 70 73 80 5 24 49 90 56 74 3 46 64 92 6 37 59 98 4 78 66 32 33 72 50 12 40 85 9 82 29 36 28 91 68 99 61 16 60 54 100 95 45 41 21 38 81 19 30



Rysunek 3: Przedstawienie czasu działania algorytmu.

3.3 Porównanie algorytmu NEH z symulowanym wyżarzaniem

W tabeli 2 zostały porównane wyniki zwracane przez oba algorytmy - NEH i symulowane wyżarzanie.

zestaw testowy	ilość zadań w zestawie (n x m)	C_{max} - NEH	C_{max} - wyżarzanie
1	6x3	47	47
2	10x5	637	619
3	10x10	1163	1287
4	10x20	1805	1861
5	20x5	1137	1138
6	20x10	1471	1616
7	20x20	2397	2577
8	100x10	5948	5981
9	100x20	6404	7079

Tabela 2: Porównanie wyników zwracanych przez algorytmy.

Niestety, nie da się w czytelny sposób porównać czasów działania algorytmów na tym samym wykresie - algorytm NEH dla pierwszych zestawów danych wykonuje się w mniej niż milisekundę, podczas gdy wyżarzanie już wtedy trwa co najmniej 20 ms. Wynika z tego, że mimo podobnego rzędu złożoności obliczeniowej $O(n^3)$ w przypadku wyżarzania mamy do czynienia z tak dużymi stałymi, że wykonuje się on zdecydowanie dłużej.

4 Wnioski

- Obliczeniowa złożoność algorytmu wynosi $O(n^3)$, a stałe występujące przy n^3 na tyle duże, że przy używaniu go dla większej liczby maszyn oraz zadań czas wykonywania się algorytmu może być bardzo długi.
- Algorytm NEH jest zdecydowanie lepszym rozwiązaniem niż symulowane wyżarzanie dla zbadanych przypadków - zwracane C_{max} jest w większości przypadków mniejsze, a czas wykonywania o wiele krótszy.
- Algorytm symulowanego wyżarzania wprowadza element losowości do swojego działania, zatem nie mamy pewności, że otrzymamy te same wyniki za każdym razem.
- Efekt działania algorytmu zależy również od przyjętych parametrów (definicji sąsiedztwa, funkcji akceptacji schematu chłodzenia) - najpewniej inne wyniki otrzymalibyśmy dla różnych kombinacji tych elementów.