STEROWANIE PROCESAMI CIĄGŁYMI
Algorytm Carliera

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

01.05.2021r.

1 Opis problemu

Ponownie będziemy się zajmować problemem RPQ. Dla przypomnienia - jest to problem jednomaszynowy, gdzie na wejściu podane są trzy parametry dla każdego zadania:

- r_i – termin dostępności zadania i,
- p_i – czas wykonania zadania i,
- q_i – czas dostarczenia zadania i.

Wyjściem działania algorytmu obsługującego problem RPQ jest permutacja wykonania zadań π oraz moment maksymalny z możliwych terminów dostarczenia zadań C_{max}. Zaimplementowany został algorytm Carliera do rozwiązania tego problemu.

2 Algorytm

2.1 Opis teoretyczny i implementacja

Algorytm Carliera jest algorytmem dokładnym wykorzystujący paradygmat Branch-and-Bound (metoda podziału i ograniczeń). Określony jest podział na dolne i górne ograniczenia, zasadę podziału oraz warunek eliminacji. Algorytm Schrage (z podziałem i bez podziału) jest wykorzystywany przy implementacji algorytmu Carliera. Dla każdego węzła w drzewie rozwiązań generowane jest kompletne rozwiązanie. Charakterystycznym elementem algorytmu jest także jego rekurencyjne wywoływanie, jeśli wyznaczone dolne ograniczenie będzie mniejsze niż górne oszacowanie. Złożoność obliczeniowa dla rozwiązania optymalnego dla problemu RPQ wynosi O(n logn).

Lista kroków algorytmu Carliera *CARLIER(n, R, P, Q)* prezentuje się następująco:

1. U $\longleftarrow$ SCHRAGE(n, R, P, Q)
2. if U $<$ UB then:
   (a) UB $\longleftarrow$ U, $\pi^* \longleftarrow \pi$
3. end if
4. b $\longleftarrow$ max $\{ j \in N$: $1 \leqslant j \leqslant n \wedge C_{max}(\pi) = C_{\pi(j)} + q_{\pi(j)} \}$
5. a $\longleftarrow$ min $\{ j \in N$: $1 \leqslant j \leqslant n \wedge C_{max}(\pi) = r_{\pi(j)} + \sum_{s=j}^{b} p_{\pi(s)} + q_{\pi(b)} \}$
6. c $\longleftarrow$ max $\{ j \in N$: $a \leqslant j \leqslant b \wedge q_{\pi(j)} < q_{\pi(b)}\}$
7. if c $= \emptyset$ then:
   (a) return π^*
8. end if
9. $\kappa \longleftarrow \{c+1, c+2, ..., b\}$
10. $r(\kappa) \longleftarrow \min\limits_{j \in \kappa} r_{\pi(j)}$, $q(\kappa) \longleftarrow \min\limits_{j \in \kappa} q_{\pi(j)}$, $p(\kappa) \longleftarrow \sum_{j \in \kappa} p_{\pi(j)}$
11. $r_{\pi(c)} \longleftarrow$ max $\{ r_{\pi(c), r(\kappa)+p(\kappa)}\}$
12. LB $\longleftarrow$ SCHRAGEPMTN(n, R, P, Q)
13. LB $\longleftarrow$ max $\{ h(\kappa), h(\kappa \cup \{c\})$, LB$\}$
14. if LB $<$ UB then:
   (a) CARLIER(n, R, P, Q)
15. end if
16. odtwórz $r_{\pi(c)}$
17. $q_{\pi(c)} \longleftarrow$ max $\{ q_{\pi(c), q(\kappa)+p(\kappa)}\}$
18. LB $\longleftarrow$ SCHRAGEPMTN(n, R, P, Q)
19. LB $\longleftarrow$ max $\{ h(\kappa), h(\kappa \cup \{c\})$, LB$\}$
20. if LB $<$ UB then:
   (a) CARLIER(n, R, P, Q)
21. end if
22. odtwórz $q_{\pi(c)}$

Implementacja algorytmu znajduje się poniżej.

Listing 1: Implementacja algorytmu Carliera.

```
void Calier_algorithm(std::vector<task>& tasks)
{
    std::pair<std::vector<int>, int> pre = schrage_algorithm(tasks);
    std::vector<int> pi = pre.first;
    int U = pre.second;
    if(U < UB)
    {
            UB = U;
            Pi_star = pi;
    }
    int a = tasks.size() + 1, b = -1, c = -1;
    std::vector<int> C;
    compute_complete_times(tasks, pi, C);
    for(unsigned int i = 0; i < tasks.size(); i++)
    {
            if(U == C[pi[i] - 1] + tasks[pi[i] - 1].q)
                    b = i;
    }
    int sum = 0;

    for(int i = tasks.size() - 1; i >= 0; i--)
    {
            if(i <= b)
                    sum += tasks[pi[i] - 1].p;
            if(U == tasks[pi[i] - 1].r + sum + tasks[pi[b] - 1].q)
                    a = i;
    }
    for(unsigned int i = a; i <= b; i++)
    {
            if(tasks[pi[i] - 1].q < tasks[pi[b] - 1].q)
                    c = i;
    }
    if(c == -1)
            return;
    int r_prim = INF, q_prim = INF, p_prim = 0;
    for(int i = c + 1; i <=b; i++)
    {
            r_prim = std::min(r_prim, tasks[pi[i] - 1].r);
            q_prim = std::min(q_prim, tasks[pi[i] - 1].q);
            p_prim += tasks[pi[i] -1].p;
    }
    int LB, hk, hkc;
```

```
        int r_c_prev = tasks[pi[c] - 1].r;

        tasks[pi[c] - 1].r = std::max( tasks[pi[c] - 1].r, r_prim + p_prim);
        LB = schrage_algorithm_pmtn(tasks);
        hk = r_prim + p_prim + q_prim;
        hkc = std::min(r_prim, tasks[pi[c] - 1].r) + std::min(q_prim, tasks[pi[c] - 1].q)
        + p_prim + tasks[pi[c] - 1].p;
        LB = std::max( std::max(LB, hk), hkc);

        if(LB < UB)
                Calier_algorithm(tasks);
        tasks[pi[c] - 1].r = r_c_prev;

        int q_c_prev = tasks[pi[c] - 1].q;

        tasks[pi[c] - 1].q = std::max( tasks[pi[c] - 1].q, q_prim + p_prim);
        LB = schrage_algorithm_pmtn(tasks);
        hk = r_prim + p_prim + q_prim;
        hkc = std::min(r_prim, tasks[pi[c] - 1].r) + std::min(q_prim, tasks[pi[c] - 1].q)
        + p_prim + tasks[pi[c] - 1].p;
        LB = std::max( std::max(LB, hk), hkc);

        if(LB < UB)
                Calier_algorithm(tasks);
        tasks[pi[c] - 1].q = q_c_prev;
}
```

3 Wyniki i ich porównanie

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

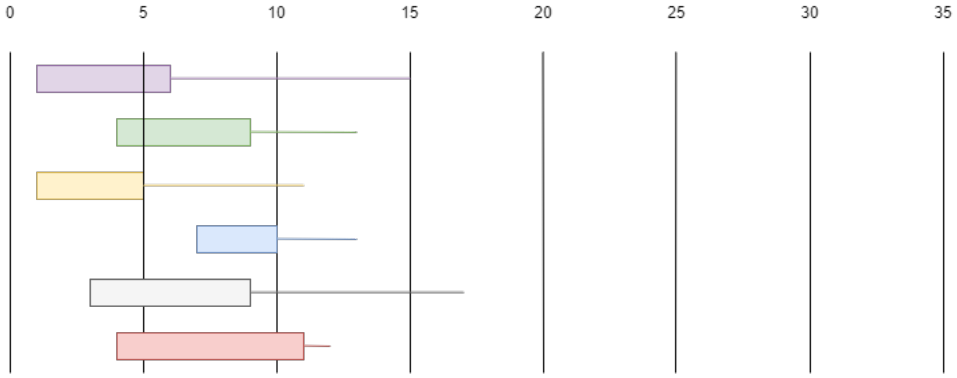
3.1 Przykładowe działanie algorytmu

Na podstawie pierwszego zestawu testowego zostanie przedstawione działanie algorytmu opisanych w sekcji 2. Dane tekstowe przedstawiają się wówczas następująco:

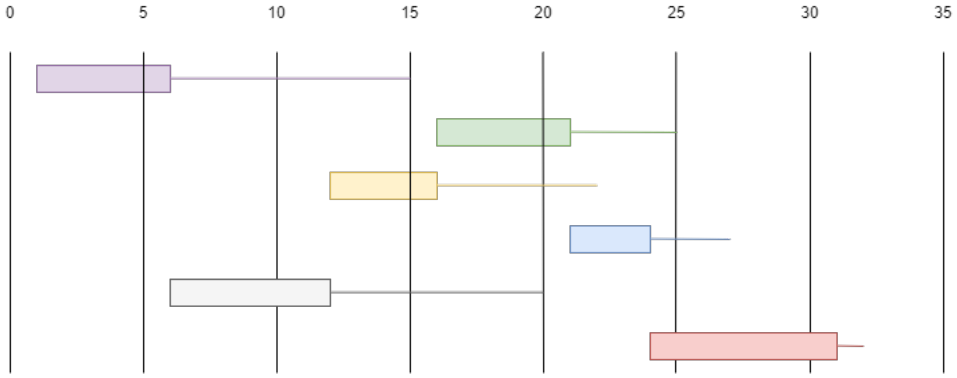
```
6
1 5 9
4 5 4
1 4 6
7 3 3
3 6 8
4 7 1
```

Pierwsza liczba oznacza liczbę zadań, które mamy uszeregować. Kolejne parametry dla każdego zadania odpowiadają wartościom r_i, p_i, q_i objaśnione w opisie problemu na początku sprawozdania. Dane obrazują sytuację zaprezentowane na

rysunku 1. Po przepuszczeniu przez algorytm Carliera otrzymujemy poszeregowanie jak na rys. 2 odpowiadające kolejności zadań: 1, 5, 3, 2, 4, 6 oraz wartości $C_{max} = 32$.



Rysunek 1: Nieposzeregowane zadania z pierwszego zbioru danych.



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

Jak widać, wynik ten jest identyczny z tym otrzymanym wcześniej przez algorytm Schrage. Inny wynik otrzymujemy w przypadku drugiego zestawu danych:
10
219 5 276
84 13 103
336 35 146
271 62 264
120 33 303

299 14 328
106 46 91
181 93 97
263 13 168
79 60 235

Algorytm Carliera zwraca następującą kolejność wykonywania zadań: 10 5 2 8 1 9 6 4 3 7. W takim wypadku otrzymujemy $C_{max} = 641$.

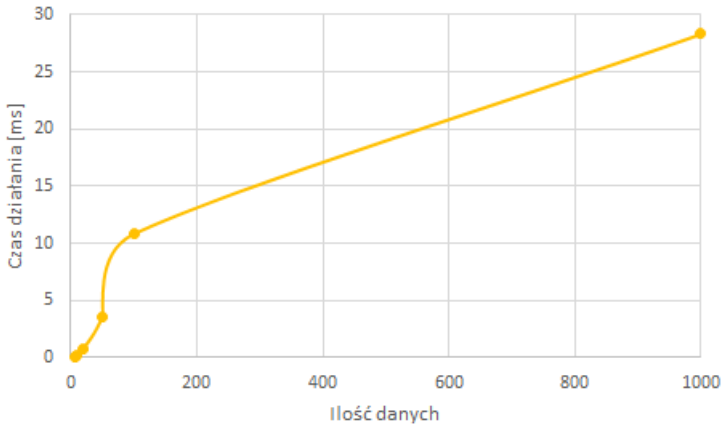
3.2 Wyniki na danych testowych

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	C_{max}	czas działania programu [μs]
1	6	32	38
2	10	641	268
3	20	1267	806
4	20	1386	791
5	50	3472	3527
6	50	3617	6793
7	100	6885	10859
8	100	6904	13735
9	1000	72852	28341

Tabela 1: Wynik działania algorytmu Carliera.

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



Rysunek 3: Przedstawienie czasu działania algorytmu.

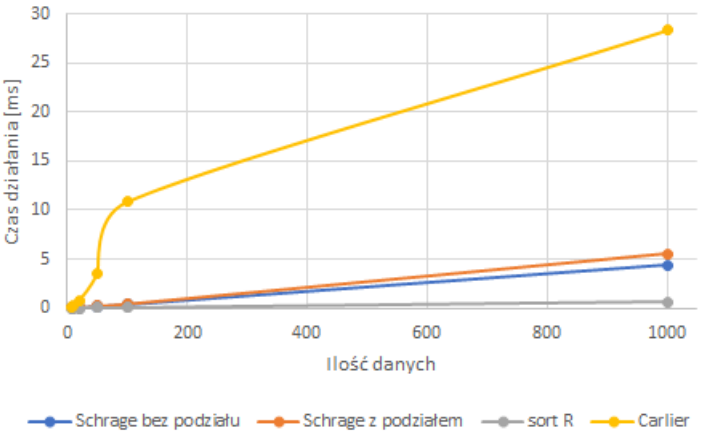
3.3 Porównanie algorytmu Carliera z poprzednimi algorytmami

Algorytm Carliera jest lepszym rozwiązaniem w porównaniu do algorytmu Jacksona ze względu na złożoność $O(n \cdot logn)$ lepszą niż $O(n^2)$ Jacksona. Ciężko jednak byłoby porównać oba algorytmy w czystych formach, gdyż w przypadku alg. Jacksona nie był podawany parametr q - należałoby je odpowiednio zmodyfikować, aby wykonywać je z tymi samymi danymi testowymi. Algorytmy można jednak porównać z poprzednimi rozwiązaniami problemu RPQ - sortowaniu po r, algorytmem Schrage z podziałem i bez podziału. Dla przypomnienia zamieszona poniżej tabela 2 przedstawia porównanie wyników C_{max} dla tych algorytmów.

zestaw testowy	alg. Schrage	alg. Schrage (podział)	sort. r	Carlier
1	32	32	34	32
2	687	641	764	641
3	1299	1257	1594	1267
4	1399	1386	1576	1386
5	3487	3472	4013	3472
6	3659	3617	4296	3617
7	6918	6885	7831	6885
8	6936	6904	7973	6904
9	72853	72852	86648	72852

Tabela 2: Porównanie wyników C_{max}.

Na rys. 4 można zauważyć różnicę w czasowym wykonywaniu się programów.



Rysunek 4: Porównanie czasu działania algorytmów.

4 Wnioski

- Czasowa złożoność algorytmu prezentuje się dużo gorzej niż w przypadku sortowania po r czy obu wariantów algorytmu Schrage, jednak otrzymywane permutacje są zdecydowanie lepsze. Za wyjątkiem Schrage z podziałem, C_{max} mają mniejsze wartości.
- Algorytm Carliera zwraca taką samą wartość C_{max} co algorytm Schrage z podziałem. Nie powinno się jednak ich porównywać bez wzięcia pod uwagę faktu, że oba algorytmy mają odmienną naturę - algorytm Carliera nie dzieli zadań. W zależności od potrzeb i możliwości systemu, można wybrać bardziej złożony czasowo alg. Carliera nie dzielący zadań lub szybszy, ale dzielący zadania algorytm Schrage.
- Całkiem długo czas wykonywania się algorytmu wynika z rekurencji i tworzącemu się w związku z tym drzewem wywołań rekurencyjnych.