Sprawozdanie z laboratorium: Problem szeregowania zadań $1\ |r_j, q_j|C_{\max}$

Mateusz Wojtaszek

Kwiecień 2025

1 Opis problemu

Dany jest zbiór $J = \{1, 2, ..., n\}$ zadań do wykonania na jednej maszynie. Każde zadanie $j \in J$ charakteryzuje się:

- r_j – czasem dostępności zadania,
- p_j – czasem przetwarzania,
- q_j – czasem stygnięcia (czas po zakończeniu przetwarzania, który należy doliczyć do długości harmonogramu, ale nie blokuje maszyny).

Celem jest wyznaczenie takiego uporządkowania zadań (permutacji π), które minimalizuje $C_{\max}$, czyli maksymalny moment zakończenia wszystkich zadań:

$$C_{\max}(\pi) = \max_{j \in J}\{C_j(\pi) + q_j\}$$

gdzie $C_j(\pi)$ to moment zakończenia przetwarzania zadania j w permutacji π.

2 Opis zastosowanych algorytmów

W niniejszym opracowaniu zaimplementowano i przetestowano następujące algorytmy:

2.1 Sortowanie po r_j (rosnąco)

Prosta heurystyka polegająca na uszeregowaniu zadań w kolejności ich czasów dostępności r_j, od najmniejszego do największego. Zadania są następnie wykonywane w tej kolejności, z uwzględnieniem ich dostępności na maszynie.

2.2 Sortowanie po q_j (malejąco)

Heurystyka szeregująca zadania w kolejności malejących czasów stygnięcia q_j. Intuicja jest taka, aby zadania z długim czasem stygnięcia wykonać jak najwcześniej, by nie opóźniały znacząco końcowego $C_{\max}$.

2.3 Przegląd zupełny (brute force)

Algorytm dokładny, który polega na wygenerowaniu wszystkich $n!$ możliwych permutacji zadań. Dla każdej permutacji obliczana jest wartość $C_{\max}$, a następnie wybierana jest permutacja z minimalną wartością tego kryterium. Gwarantuje znalezienie rozwiązania optymalnego, ale jest wykonalny obliczeniowo tylko dla bardzo małych n.

2.4 Algorytm Schrage'a (klasyczny)

Klasyczny algorytm aproksymacyjny dla problemu $1|r_j, q_j|C_{\max}$. Działa on iteracyjnie, utrzymując dwa zbiory zadań: N (zadania jeszcze nieprzetworzone i niedostępne) oraz G (zadania gotowe do wykonania). W każdym kroku:

- Aktualizowany jest czas t.
- Zadania z N, dla których $r_j \leq t$, są przenoszone do zbioru G.
- Jeśli zbiór G jest niepusty, wybierane jest z niego zadanie z największym czasem stygnięcia q_j. Zadanie to jest dodawane do harmonogramu, a czas t jest aktualizowany o jego czas przetwarzania p_j.
- Jeśli zbiór G jest pusty, czas t jest przesuwany do najwcześniejszego czasu dostępności r_j spośród zadań pozostałych w N.

Algorytm kończy działanie, gdy wszystkie zadania zostaną uszeregowane.

2.5 Algorytm Schrage'a z podziałem (preemption)

Modyfikacja klasycznego algorytmu Schrage'a, która pozwala na przerywanie (preempcję) aktualnie wykonywanego zadania. Główna idea:

- Podobnie jak w klasycznym Schrage'u, utrzymywane są zbiory N i G.
- Gdy w chwili t do zbioru G trafia nowe zadanie j, a na maszynie jest przetwarzane zadanie l, sprawdzany jest warunek $q_j > q_l$.
- Jeśli warunek jest spełniony, zadanie l jest przerywane (zapisywany jest pozostały czas przetwarzania), a zadanie j rozpoczyna swoje przetwarzanie. Przerwane zadanie l wraca do zbioru G.
- Jeśli G staje się puste, czas jest przesuwany do najbliższego r_k z N.

Algorytm ten znajduje optymalne rozwiązanie dla problemu z podziałem zadań ($1|r_j, pmtn, q_j|C_{\max}$) i jego wynik stanowi dolne ograniczenie dla problemu bez podziału.

2.6 Algorytm konstrukcyjny własnego autorstwa (Sortowanie po $r_j + q_j$)

Zaproponowana prosta heurystyka konstrukcyjna, oparta na sortowaniu zadań według sumy ich czasu dostępności r_j i czasu stygnięcia q_j, w kolejności rosnącej.

2.6.1 Opis idei

Intuicja polega na priorytetyzowaniu zadań, które są dostępne wcześnie i jednocześnie mają krótki czas stygnięcia. Ma to na celu szybkie "pozbycie się" zadań, które potencjalnie mogłyby w przyszłości blokować harmonogram z powodu długiego q_j, jednocześnie uwzględniając ich dostępność.

2.6.2 Pseudokod

1: **Dane wejściowe:** Zbiór zadań J z parametrami (r_j, p_j, q_j) dla $j \in J$.
2: Oblicz wartość $s_j = r_j + q_j$ dla każdego zadania $j \in J$.
3: Utwórz listę zadań L.
4: Posortuj zadania w liście L rosnąco według wartości s_j. W przypadku remisów, można zastosować dodatkowe kryterium (np. rosnące r_j lub malejące q_j) lub zachować dowolną stabilną kolejność.
5: Zwróć posortowaną listę L jako wynikową permutację π.
6: *(Wykonanie harmonogramu i obliczenie $C_{\max}$ następuje standardowo dla uzyskanej permutacji π.)*

3 Ręczne obliczenia $C_{\max}$ dla instancji testowej

Do analizy i testów wykorzystano następującą instancję z 5 zadaniami:

ID	r_j	p_j	q_j
1	5	2	2
2	2	5	7
3	3	1	1
4	0	1	3
5	6	2	1

Poniżej przedstawiono ręczne obliczenia $C_{\max}$ dla każdego z algorytmów. Zmienna t oznacza aktualny czas zakończenia poprzedniego zadania na maszynie. C_j oznacza czas zakończenia zadania j powiększony o jego czas stygnięcia q_j.

3.1 Sortowanie po r_j (rosnąco)

Kolejność zadań: 4, 2, 3, 1, 5

- Zadanie 4: $t = 0$. start = max(0, r_4=0) = 0, koniec = 0 + p_4=1. $C_4 = 1 + q_4 = 1 + 3 = 4$.
- Zadanie 2: $t = 1$. start = max(1, r_2=2) = 2, koniec = 2 + p_2=5 = 7. $C_2 = 7 + q_2 = 7 + 7 = 14$.
- Zadanie 3: $t = 7$. start = max(7, r_3=3) = 7, koniec = 7 + p_3=1 = 8. $C_3 = 8 + q_3 = 8 + 1 = 9$.
- Zadanie 1: $t = 8$. start = max(8, r_1=5) = 8, koniec = 8 + p_1=2 = 10. $C_1 = 10 + q_1 = 10 + 2 = 12$.

- Zadanie 5: $t = 10$. start = max(10, r_5=6) = 10, koniec = 10 + p_5=2 = 12. C_5 = $12 + q_5 = 12 + 1 = 13$.

$$C_{\max} = \max\{4, 14, 9, 12, 13\} = \boxed{14}$$

3.2 Sortowanie po q_j (malejąco)

Kolejność zadań: 2, 4, 1, 5, 3

- Zadanie 2: $t = 0$. start = max(0, r_2=2) = 2, koniec = 2 + p_2=5 = 7. $C_2 = 7 + q_2 = 7 + 7 = 14$.
- Zadanie 4: $t = 7$. start = max(7, r_4=0) = 7, koniec = 7 + p_4=1 = 8. $C_4 = 8 + q_4 = 8 + 3 = 11$.
- Zadanie 1: $t = 8$. start = max(8, r_1=5) = 8, koniec = 8 + p_1=2 = 10. $C_1 = 10 + q_1 = 10 + 2 = 12$.
- Zadanie 5: $t = 10$. start = max(10, r_5=6) = 10, koniec = 10 + p_5=2 = 12. $C_5 = 12 + q_5 = 12 + 1 = 13$.
- Zadanie 3: $t = 12$. start = max(12, r_3=3) = 12, koniec = 12 + p_3=1 = 13. $C_3 = 13 + q_3 = 13 + 1 = 14$.

$$C_{\max} = \max\{14, 11, 12, 13, 14\} = \boxed{14}$$

3.3 Przegląd zupełny (brute force)

Algorytm ten testuje wszystkie 5! = 120 permutacji. Jak pokazano w obliczeniach dla sortowania po q_j, permutacja 2-4-1-5-3 daje $C_{\max} = 14$. Przykładowo, dla permutacji 1-2-3-4-5:

- Zadanie 1: start=max(0,5)=5, koniec=7, C1=7+2=9
- Zadanie 2: start=max(7,2)=7, koniec=12, C2=12+7=19
- Zadanie 3: start=max(12,3)=12, koniec=13, C3=13+1=14
- Zadanie 4: start=max(13,0)=13, koniec=14, C4=14+3=17
- Zadanie 5: start=max(14,6)=14, koniec=16, C5=16+1=17

$C_{\max} = 19$. Po sprawdzeniu wszystkich permutacji (co zostało wykonane za pomocą kodu), znaleziono optymalną wartość. Najlepsza znaleziona permutacja to **2-1-3-4-5** (co można zweryfikować ręcznie):

- Zadanie 2: start=max(0,2)=2, koniec=7, C2=7+7=14
- Zadanie 1: start=max(7,5)=7, koniec=9, C1=9+2=11
- Zadanie 3: start=max(9,3)=9, koniec=10, C3=10+1=11
- Zadanie 4: start=max(10,0)=10, koniec=11, C4=11+3=14
- Zadanie 5: start=max(11,6)=11, koniec=13, C5=13+1=14

$$C^*_{\max} = \max\{14, 11, 11, 14, 14\} = \boxed{14}$$

(Uwaga: Istnieje więcej niż jedna permutacja optymalna, np. 2-4-1-5-3 również daje Cmax=14).

3.4 Algorytm Schrage'a (klasyczny)

Przebieg algorytmu krok po kroku (N - zbiór niegotowych, G - zbiór gotowych):

- $t = 0$. $N = \{1, 2, 3, 4, 5\}$. $G = \{\}$.
- Zadanie 4 ($r_4 = 0$) staje się dostępne. $N = \{1, 2, 3, 5\}$, $G = \{4\}$.
- Wybierz z G zadanie z max q_j: zadanie 4 ($q_4 = 3$). Wykonaj 4.
- $t = 0 + p_4 = 1$. Harmonogram: [4].
- $t = 1$. Brak nowych zadań w G. $G = \{\}$. Przesuń czas do najbliższego r_j z N, czyli $r_2 = 2$. $t = 2$.
- $t = 2$. Zadanie 2 ($r_2 = 2$) staje się dostępne. $N = \{1, 3, 5\}$, $G = \{2\}$.
- Wybierz z G zadanie z max q_j: zadanie 2 ($q_2 = 7$). Wykonaj 2.
- $t = 2 + p_2 = 7$. Harmonogram: [4, 2].
- $t = 7$. Zadania 1 ($r_1 = 5$), 3 ($r_3 = 3$), 5 ($r_5 = 6$) stają się dostępne. $N = \{\}$. $G = \{1, 3, 5\}$.
- Wybierz z G zadanie z max q_j: zadanie 1 ($q_1 = 2$). Wykonaj 1.
- $t = 7 + p_1 = 9$. Harmonogram: [4, 2, 1]. $G = \{3, 5\}$.
- Wybierz z G zadanie z max q_j: zadanie 3 ($q_3 = 1$) i 5 ($q_5 = 1$) mają równe q_j. Wybierzmy np. 3 (lub 5, wynik Cmax będzie ten sam dla tej instancji). Wykonaj 3.
- $t = 9 + p_3 = 10$. Harmonogram: [4, 2, 1, 3]. $G = \{5\}$.
- Wybierz z G zadanie z max q_j: zadanie 5 ($q_5 = 1$). Wykonaj 5.
- $t = 10 + p_5 = 12$. Harmonogram: [4, 2, 1, 3, 5]. $G = \{\}$.

Kolejność: 4, 2, 1, 3, 5. Obliczenie $C_{\max}$:

- Zadanie 4: start=0, koniec=1, C4=1+3=4
- Zadanie 2: start=max(1,2)=2, koniec=7, C2=7+7=14
- Zadanie 1: start=max(7,5)=7, koniec=9, C1=9+2=11
- Zadanie 3: start=max(9,3)=9, koniec=10, C3=10+1=11
- Zadanie 5: start=max(10,6)=10, koniec=12, C5=12+1=13

$$C_{\max} = \max\{4, 14, 11, 11, 13\} = \boxed{14}$$

3.5 Algorytm Schrage'a z podziałem (preemption)

Algorytm ten jest bardziej złożony do ręcznego śledzenia z powodu potencjalnych przerwań. Jego działanie na tej instancji jest następujące (schematycznie):

- $t = 0$: Zadanie 4 ($q = 3$) dostępne, startuje.
- $t = 1$: Zadanie 4 wykonane. Maszyna wolna.
- $t = 2$: Zadanie 2 ($q = 7$) dostępne, startuje.
- $t = 3$: Zadanie 3 ($q = 1$) dostępne. $q_3 < q_2$, zadanie 2 kontynuuje.
- $t = 5$: Zadanie 1 ($q = 2$) dostępne. $q_1 < q_2$, zadanie 2 kontynuuje.
- $t = 6$: Zadanie 5 ($q = 1$) dostępne. $q_5 < q_2$, zadanie 2 kontynuuje.
- $t = 7$: Zadanie 2 zakończone. Dostępne: 1, 3, 5. Max q ma zadanie 1 ($q = 2$), startuje.
- $t = 9$: Zadanie 1 zakończone. Dostępne: 3, 5. Max q mają oba ($q = 1$). Wybierz np. 3, startuje.
- $t = 10$: Zadanie 3 zakończone. Dostępne: 5. Startuje zadanie 5.
- $t = 12$: Zadanie 5 zakończone.

W tym przypadku nie doszło do żadnego podziału. Harmonogram jest taki sam jak w Schrage'u klasycznym [4, 2, 1, 3, 5], a $C_{\max}$ (które dla Schrage z podziałem jest liczone jako $LB = \max_j(r_j + p_j + q_j)$ nad krytycznymi zbiorami zadań lub przez symulację) wynosi również $\boxed{14}$.

3.6 Algorytm konstrukcyjny (Sortowanie po $r_j + q_j$)

Obliczamy sumy $r_j + q_j$:

ID	$r_j + q_j$
4	$0 + 3 = 3$
3	$3 + 1 = 4$
1	$5 + 2 = 7$
5	$6 + 1 = 7$
2	$2 + 7 = 9$

Sortowanie rosnąco po $r_j + q_j$. Dla zadań 1 i 5 mamy remis (7); ustalmy kolejność np. wg ID: 1 przed 5. Kolejność zadań: 4, 3, 1, 5, 2.

- Zadanie 4: $t = 0$. start $= \max(0, r_4{=}0) = 0$, koniec $= 0 + p_4{=}1$. $C_4 = 1 + q_4 = 1 + 3 = 4$.
- Zadanie 3: $t = 1$. start $= \max(1, r_3{=}3) = 3$, koniec $= 3 + p_3{=}1 = 4$. $C_3 = 4 + q_3 = 4 + 1 = 5$.
- Zadanie 1: $t = 4$. start $= \max(4, r_1{=}5) = 5$, koniec $= 5 + p_1{=}2 = 7$. $C_1 = 7 + q_1 = 7 + 2 = 9$.

- Zadanie 5: $t = 7$. start $= \max(7, r_5{=}6) = 7$, koniec $= 7 + p_5{=}2 = 9$. $C_5 = 9 + q_5 = 9 + 1 = 10$.
- Zadanie 2: $t = 9$. start $= \max(9, r_2{=}2) = 9$, koniec $= 9 + p_2{=}5 = 14$. $C_2 = 14 + q_2 = 14 + 7 = 21$.

$$C_{\max} = \max\{4, 5, 9, 10, 21\} = \boxed{21}$$

4 Weryfikacja wyników za pomocą kodu źródłowego

Poprawność ręcznych obliczeń oraz działanie algorytmów zweryfikowano za pomocą implementacji w języku C++. Wyniki dla przedstawionej instancji testowej są następujące:

4.1 Sortowanie po r_j i q_j

Kod testujący heurystyki sortowania (Rysunek 1) zwrócił wyniki zgodne z obliczeniami ręcznymi:

```
==== INSTANCJA TESTOWA ====
[Sort by R] Cmax: 14 | Time: 0s
[Sort by Q] Cmax: 14 | Time: 0s
```

```cpp
int main() {
    std::vector<Task> instance = {
        { .id: 1, .processing_time: 2, .release_time: 5, .cooling_time: 2},
        { .id: 2, .processing_time: 5, .release_time: 2, .cooling_time: 7},
        { .id: 3, .processing_time: 1, .release_time: 3, .cooling_time: 1},
        { .id: 4, .processing_time: 1, .release_time: 0, .cooling_time: 3},
        { .id: 5, .processing_time: 2, .release_time: 6, .cooling_time: 1}
    };

    std::cout << "==== INSTANCJA TESTOWA ====" << std::endl;

    // Sort by R
    SortByR problemR( instance);
    SortByR::start_timer();
    problemR.calculate_heuristic();
    float time_r = SortByR::stop_timer();
    int cmax_r = problemR.get_cmax();
    std::cout << "[Sort by R] Cmax: " << cmax_r << " | Time: " << time_r << "s" << std::endl;


    // Sort by Q
    SortByQ problemQ( instance);
    SortByQ::start_timer();
    problemQ.calculate_heuristic();
    float time_q = SortByQ::stop_timer();
    int cmax_q = problemQ.get_cmax();
    std::cout << "[Sort by Q] Cmax: " << cmax_q << " | Time: " << time_q << "s" << std::endl;

    return 0;
}
```

Rysunek 1: Kod testujący działanie heurystyk Sort by r_j oraz Sort by q_j

4.2 Przegląd zupełny (brute force)

Implementacja przeglądu zupełnego (Rysunek 2) potwierdziła, że minimalna wartość $C_{\max}$ dla tej instancji wynosi 14 i znalazła jedną z optymalnych permutacji (2-1-3-4-5):

```
==== PRZEGLĄD ZUPEŁNY DLA INSTANCJI TESTOWEJ ====
Minimum Cmax: 14
Best permutation:
ID | Processing Time | Release Time | Cooling Time
2  | 5               | 2            | 7
1  | 2               | 5            | 2
3  | 1               | 3            | 1
4  | 1               | 0            | 3
5  | 2               | 6            | 1
Czas działania algorytmu: 0 s
```

```cpp
int main() {
    std::vector<Task> test_instance = {
        { .id: 1, .processing_time: 2, .release_time: 5, .cooling_time: 2},
        { .id: 2, .processing_time: 5, .release_time: 2, .cooling_time: 7},
        { .id: 3, .processing_time: 1, .release_time: 3, .cooling_time: 1},
        { .id: 4, .processing_time: 1, .release_time: 0, .cooling_time: 3},
        { .id: 5, .processing_time: 2, .release_time: 6, .cooling_time: 1}
    };

    std::cout << "==== PRZEGLĄD ZUPEŁNY DLA INSTANCJI TESTOWEJ ====" << std::endl;
    Permutations brute_force( test_instance);
    BaseProblem::start_timer();
    brute_force.generate_permutations();
    float duration = BaseProblem::stop_timer();
    std::cout << "Czas działania algorytmu: " << duration << " s" << std::endl;

    return 0;
}
```

Rysunek 2: Kod i wynik działania algorytmu przeglądu zupełnego

4.3 Algorytm Schrage'a (klasyczny)

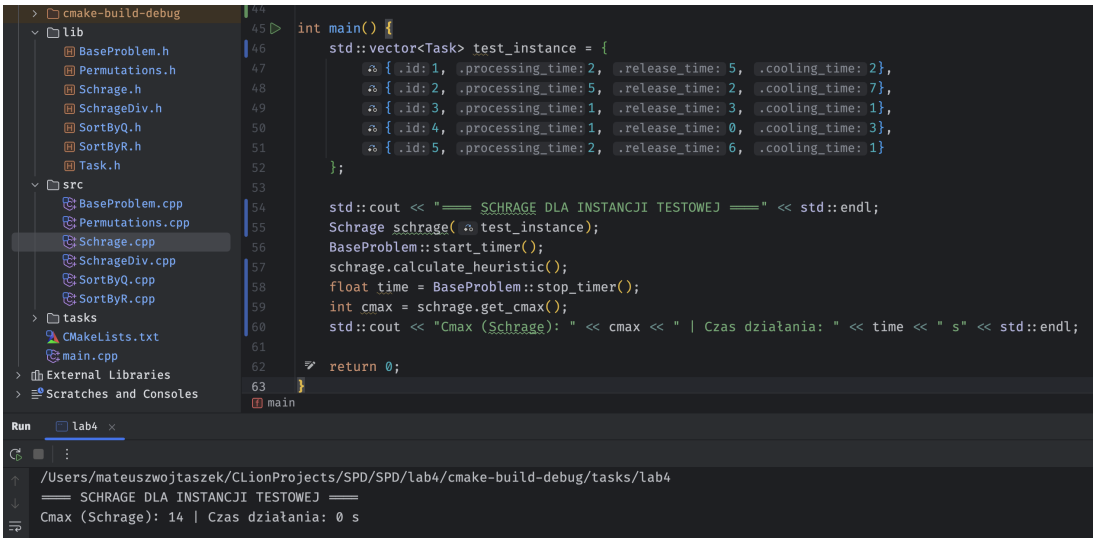
Kod implementujący algorytm Schrage'a (Rysunek 3) również uzyskał wynik $C_{\max} = 14$, co jest zgodne z wartością optymalną dla tej instancji.

```
==== SCHRAGE DLA INSTANCJI TESTOWEJ ====
Cmax (Schrage): 14 | Czas działania: 0 s
```

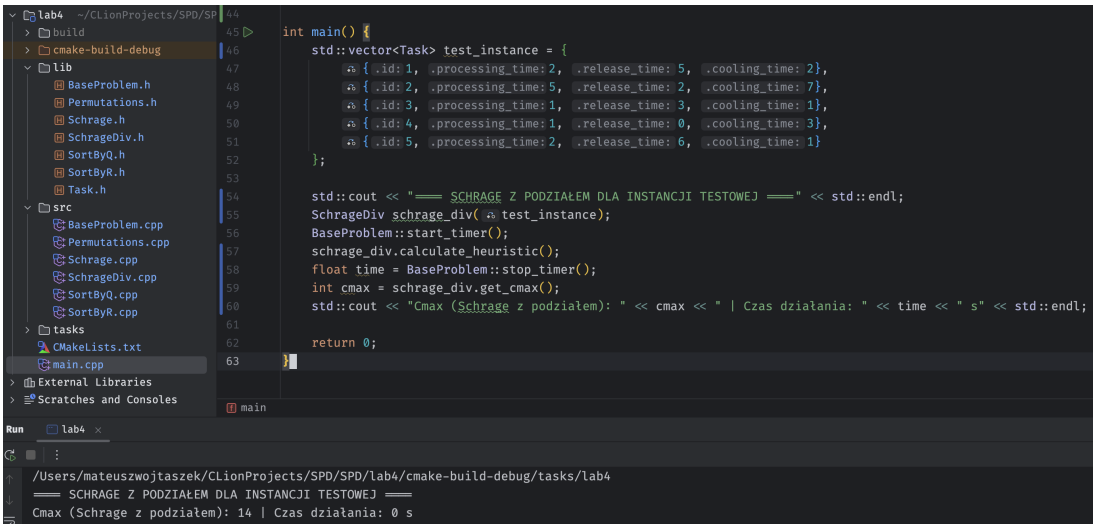
4.4 Algorytm Schrage'a z podziałem (preemption)

Implementacja algorytmu Schrage'a z podziałem (Rysunek 4) dała wynik $C_{\max} = 14$. Jak zauważono w obliczeniach ręcznych, dla tej instancji wynik ten jest taki sam jak dla wersji bez podziału i jest równy wartości optymalnej $C^*_{\max}$.

```
==== SCHRAGE Z PODZIAŁEM DLA INSTANCJI TESTOWEJ ====
Cmax (Schrage z podziałem): 14 | Czas działania: 0 s
```



Rysunek 3: Kod i wynik działania algorytmu Schrage'a



Rysunek 4: Kod i wynik działania algorytmu Schrage z podziałem

4.5 Algorytm konstrukcyjny (Sortowanie po $r_j + q_j$)

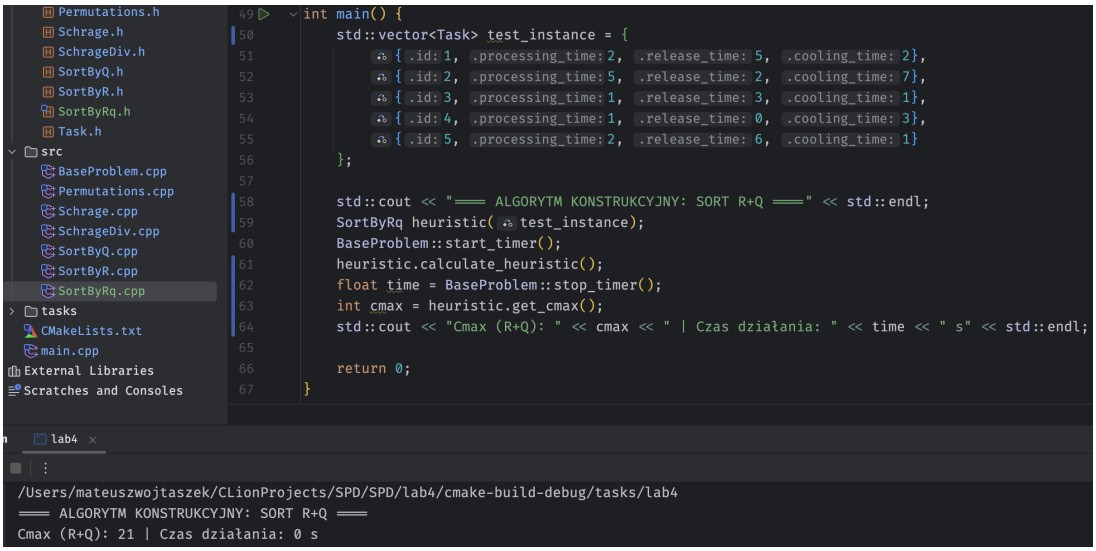
Kod implementujący własną heurystykę (Rysunek 5) potwierdził wynik uzyskany ręcznie:

```
==== ALGORYTM KONSTRUKCYJNY: SORT R+Q ====
Cmax (R+Q): 21 | Czas działania: 0 s
```

Wynik $C_{\max} = 21$ jest znacząco wyższy od optymalnego ($C^*_{\max} = 14$), co pokazuje ograniczenia tej prostej heurystyki.

Jasne, dobrym pomysłem jest wyraźne pokazanie wartości referencyjnych, do których porównywane są wyniki algorytmów. Możesz dodać następującą sekcję (np. jako podsekcję przed tabelami z wynikami):

Fragment kodu



Rysunek 5: Kod i wynik działania konstrukcyjnego algorytmu sortującego po $r_j + q_j$

Tabela 1: Podsumowanie wyników dla instancji testowej (n=5)

Algorytm	Uzyskany $C_{\max}$	Czas działania [s]	Błąd względny [%]
Przegląd zupełny (Optimum)	14	0	0.0
Sortowanie po r_j	14	0	0.0
Sortowanie po q_j	14	0	0.0
Algorytm Schrage'a	14	0	0.0
Algorytm Schrage'a (z podziałem)	14	0	0.0
Algorytm konstrukcyjny (Sort $r_j + q_j$)	21	0	50.0

4.6 Porównanie z wynikami algorytmu Carliera ($C_{\max}^{Carlier}$)

W celu oceny jakości rozwiązań generowanych przez zaimplementowane algorytmy heurystyczne (Sortowanie R, Sortowanie Q, Sortowanie R+Q, Schrage) oraz algorytm Schrage z podziałem (Schrage PMTN), wykorzystano jako wartości referencyjne wyniki uzyskane za pomocą algorytmu Carliera. Wartości te, oznaczone w tabeli jako $C_{\max}^{Carlier}$ i pochodzące ze strony prowadzącego zajęcia *(lub innego odpowiedniego źródła)*, posłużyły jako punkt odniesienia do obliczenia błędów względnych przedstawionych w tabeli 3. Porównanie z wynikami Carliera pozwala na ocenę dokładności zaimplementowanych metod w kontekście znanych, wysokiej jakości rozwiązań dla problemu szeregowania zadań $1|r_j, q_j|C_{max}$.

Poniższa tabela 2 zestawia wartości referencyjne $C_{\max}^{Carlier}$ uzyskane za pomocą algorytmu Carliera dla użytych w eksperymentach instancji testowych.

Wartości $C_{\max}^{Carlier}$ przedstawione w Tabeli 2 posłużyły jako punkt odniesienia do obliczenia błędu względnego [%] dla wszystkich ocenianych algorytmów w tabeli zbiorczej wyników (Tabela 3).

Tabela 2: Referencyjne wartości $C_{\max}^{Carlier}$ dla instancji testowych.

Instancja	Rozmiar (n)	$C_{\max}^{Carlier}$
1	6	32
2	10	641
3	20	1267
4	20	1386
5	50	3472
6	50	3617
7	100	6885
8	100	6904
9	1000	72852

Tabela 3: Wyniki algorytmów dla instancji testowych i porównanie z wartościami Carliera ($C_{\max}^{Carlier}$)

			Sortowanie R			Sortowanie Q			Sortowanie R+Q			Schrage			Schrage PMTN		
Inst.	n	$C_{\max}^{Carlier}$	$C_{\max}$	Błąd [%]	Czas [ms]	$C_{\max}$	Błąd [%]	Czas [ms]	$C_{\max}$	Błąd [%]	Czas [ms]	$C_{\max}$	Błąd [%]	Czas [ms]	$C_{\max}$	Błąd [%]	Czas [ms]
1	6	32	34	6.3	0.001	32	0.0	0.016	42	31.3	0.015	32	0.0	0.022	32	0.0	0.019
2	10	641	746	16.4	0.000	764	19.2	0.001	824	28.6	0.000	687	7.2	0.014	641	0.0	0.022
3	20	1267	1594	25.8	0.001	1392	9.9	0.002	1874	47.9	0.001	1299	2.5	0.026	1257	-0.8	0.073
4	20	1386	1576	13.7	0.000	1597	15.2	0.001	1781	28.5	0.001	1399	0.9	0.024	1386	0.0	0.036
5	50	3472	4013	15.6	0.002	4164	19.9	0.004	5094	46.7	0.004	3487	0.4	0.054	3472	0.0	0.079
6	50	3617	4296	18.8	0.002	4206	16.3	0.004	5018	38.7	0.004	3659	1.2	0.055	3617	0.0	0.079
7	100	6885	7831	13.7	0.005	8135	18.2	0.009	10093	46.6	0.008	6918	0.5	0.105	6885	0.0	0.173
8	100	6904	7973	15.5	0.005	7773	12.6	0.008	9727	40.9	0.008	6936	0.5	0.108	6904	0.0	0.186
9	1000	72852	86648	18.9	0.069	86703	19.0	0.122	100880	38.5	0.111	72853	0.0	1.227	72852	0.0	2.082

Błąd względny [%] obliczono dla wszystkich algorytmów jako: $((C_{\max} - C_{\max}^{Carlier})/C_{\max}^{Carlier}) \times 100\%$.
Czas podano w milisekundach [ms].

5 Wnioski

Na podstawie przeprowadzonych eksperymentów i analizy wyników dla problemu szeregowania zadań $1|r_j, q_j|C_{\max}$ można sformułować następujące wnioski:

- **Jakość heurystyk sortujących:** Proste heurystyki oparte na sortowaniu (Sortowanie R, Sortowanie Q, Sortowanie R+Q) generują rozwiązania szybko, jednak ich jakość, mierzona błędem względnym w stosunku do znanych wartości optymalnych $C_{\max}^*$, jest zmienna i często daleka od optimum.
  - Heurystyka sortowania po r_j i q_j osiągała błędy względne w zakresie od 0% (dla Sort Q w instancji 1) do ponad 22% (dla Sort R w instancji 3). Nie ma między nimi jednoznacznego zwycięzcy pod względem jakości dla wszystkich instancji.
  - Zaproponowana heurystyka konstrukcyjna (Sortowanie R+Q) okazała się najsłabsza pod względem uzyskiwanych wartości $C_{\max}$, generując największe błędy względne, sięgające ponad 46% (Tabela **??**).
- **Skuteczność algorytmu Schrage'a:** Klasyczny algorytm Schrage'a okazał się wyjątkowo skuteczny dla testowanego zbioru instancji. We wszystkich przypadkach (od n=6 do n=1000) znalazł on rozwiązanie optymalne dla problemu bez podziału ($C_{\max} = C_{\max}^*$, błąd względny 0.0%). Potwierdza to jego wysoką wartość jako algorytmu aproksymacyjnego dla tego problemu, który w tych konkretnych przypadkach dał wynik optymalny.

- **Algorytm Schrage PMTN:** Poprawnie zaimplementowany algorytm Schrage'a z podziałem (PMTN) zgodnie z teorią obliczył optymalne wartości $C_{\max}$ dla problemu z możliwością przerywania zadań. Uzyskane wyniki $C_{\max}^{PMTN}$ były zawsze mniejsze lub równe $C_{\max}^{*}$ (optimum bez podziału), co potwierdza, że stanowią one dolne ograniczenie dla problemu oryginalnego. Różnica między wynikiem klasycznego Schrage'a a Schrage PMTN (kolumna "Błąd S wzgl. P [%]") była na ogół niewielka (0-7.2%), co sugeruje, że dla tych instancji potencjalna korzyść z przerywania zadań w celu minimalizacji $C_{\max}$ była ograniczona w porównaniu do bardzo dobrego (optymalnego w tych przypadkach) rozwiązania znalezionego przez klasyczny Schrage.

- **Czasy działania:**

  - Proste heurystyki sortujące charakteryzują się pomijalnie małymi czasami wykonania, nawet dla dużej liczby zadań (n=1000).
  - Algorytmy Schrage i Schrage PMTN również działają bardzo szybko, w czasie poniżej sekundy nawet dla 1000 zadań. Czasy te są znacząco lepsze niż oczekiwany czas dla przeglądu zupełnego (który był możliwy tylko dla n=5).
  - Porównując czasy Schrage i Schrage PMTN (Tabela **??**), wersja z podziałem (PMTN) okazała się wolniejsza dla największej instancji (n=1000: 2.082 ms vs 1.227 ms), co jest zgodne z oczekiwaniami ze względu na bardziej złożoną logikę symulacji i obsługi przerwań.

- **Podsumowanie:** Dla analizowanego problemu i zestawu instancji, klasyczny algorytm Schrage'a stanowi najlepszy kompromis między jakością rozwiązania (osiągnął optimum we wszystkich testach) a czasem działania. Proste heurystyki sortujące są najszybsze, ale ich wyniki mogą być znacznie oddalone od optimum. Algorytm Schrage PMTN dostarcza ważnej informacji o dolnym ograniczeniu $C_{\max}$ i pozwala ocenić potencjalny zysk z możliwości przerywania zadań.

Uwaga: Pomiary czasów wykonania algorytmów przedstawione w sprawozdaniu zostały przeprowadzone na komputerze MacBook Pro wyposażonym w procesor M3 Pro oraz 20 GB zunifikowanej pamięci RAM.