Wrocław University
of Science and Technology

Sprawozdanie 1 - algorytm do rozwiązywania problemu RPQ

Komputerowo zintegorwane wytwarzanie

Autor
Miłosz Mikulski 254171

Prowadzący
Dr inż. Mariusz Makuchowski

Grupa: Śr. 13:15 - 15:00
Termin oddania: 20.03.2024

March 19, 2024

Spis treści

1 Cel laboratorium

Celem laboratorium było zaproponowanie przez nas algorytmu wyznaczającego optymalne ułożenie zadań dla problemu RPQ.

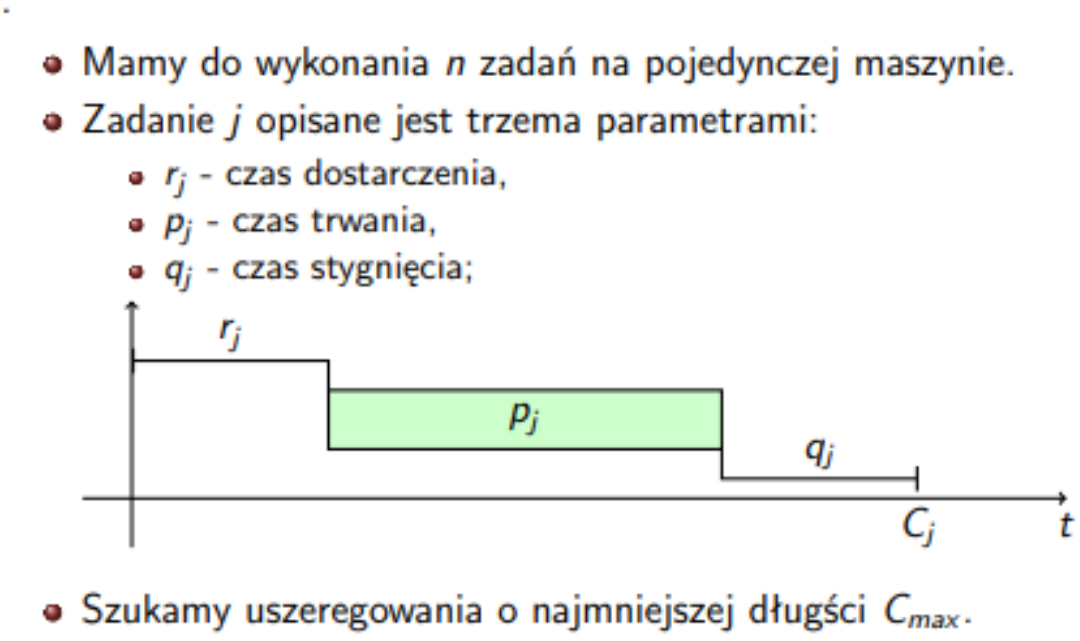

Figure 1: Przedstawienie problemu RPQ (źródło: prezentacja z wykładu)

2 Opis zastosowanego algorytmu

Zastosowany przeze mnie algorytm nosi nazwę *tabusearch*. Jest to modyfikacja algorytmu siłowego (ang. *brute force*) przechodzącego po wszystkich możliwych rozwiązaniach. Na algorytm zostały nałożone ograniczenia w postaci maksymalnej liczby iteracji oraz zastosowaniu tablicy sąsiedztwa. Jest ona kluczowym elementem algorytmu, przechowuje ona poprzednie rozwiązania przez co nie rozważamy ponownie tych samych, nieoptymalnych rozwiązań. Schemat działania algorytmu jest następujący:

1. Utwórz kopię tablicy elementów.
2. Sortuj tę tablicę według wartości R, co ułatwia wstępną selekcję lepszych rozwiązań.
3. Inicjuj najlepsze rozwiązanie jako początkową, posortowaną tablicę i oblicz jego $C_{\max}$.
4. Dla każdej iteracji, sprawdź wszystkie możliwe zamiany par elementów w tablicy:
   - Jeśli nowa sekwencja znajduje się w tablicy podobieństwa, pomiń ją.
   - W przeciwnym razie, oblicz $C_{\max}$ dla nowej sekwencji.
5. Jeśli $C_{\max}$ nowej sekwencji jest lepszy niż aktualnie najlepszy $C_{\max}$:

- Aktualizuj najlepsze rozwiązanie i najlepszy C_{max}

6. Dodaj nową sekwencję do tablicy podobieństwa, kontrolując jej maksymalny rozmiar. Jeśli tablica przekracza maksymalny rozmiar, usuń jej najstarszy element.
7. Powtarzaj kroki 4-6 dla ustalonej liczby iteracji.
8. Zwróć najlepsze znalezione rozwiązanie jako optymalną sekwencję zadań.

```
for(int i=0; i< maxIteracji; i++) {
    vector<Obiekt> najlepszaKolejnosc;
    int najlepszyCmax = INT_MAX;

    for(int k = 0; k < tab.size(); k++){
        for(int j= k + 1; j < tab.size(); j++){
            vector<Obiekt> aktualnaKolejnosc = tab;
            zamien( &: aktualnaKolejnosc[k],  &: aktualnaKolejnosc[j]);

            bool istnieje = false;
            for(auto& tablica :vector<Obiekt> & : tablicaPodobienstwa){
                if(tablica == aktualnaKolejnosc){
                    istnieje = true;
                    break;
                }
            }

            if(!istnieje){
                int aktualnyCmax = obliczCmax( obiekty: aktualnaKolejnosc);
                if(aktualnyCmax < najlepszyCmax){
                    najlepszaKolejnosc = aktualnaKolejnosc;
                    najlepszyCmax = aktualnyCmax;
                }
            }
        }
    }
```

Figure 2: Fragment kodu służący do generowania kolejnych permutacji zadań poprzez zamianę elementów

3 Wyniki

Wyznaczone wartości C_{max} przy pomocy mojej implementacji algorytmu zostały przedstawione w tab.[1]. Wszystkie wyniki zostały otrzymane przy wartości maksymalnej liczby iteracji = 50 oraz wielkości tablicy podobieństwa = 30. W tab.[2] możemy zaobserwować jak prezentuje się mój algorytm na tle dwóch, najbliższych mu wynikami, zaproponowanych algorytmów. Testy działania przeprowadziłem zarówno na systemie operacyjnym windows (rys.[3] jak i linux (rys.[4]. Różnice pomiędzy systemami były widoczne jedynie w czasie kompilacji.

Zbiór danych	Otrzymane wartości Cmax
data.1, n=24	13966
data.2, n=24	20919
data.3, n=48	33156
data.4, n=48	33878
suma	101919

Table 1: Tabela przedstawiająca otrzymane przeze mnie wartości C_{max} dla poszczególnych zbiorów danych

SortRQ	Otrzymane wyniki	Schrage
13862	13966	13981
21176	20919	21529
33424	33156	31683
33878	33878	34444
102340	101919	101637

Table 2: Tabela porównująca wyniki otrzymane przy użyciu mojego algorytmu z zaproponowanymi algorytmami

```
========= Przetwarzanie danych:  data.1 =========
Cmax: 13966
Kolejnosc: 9 18 22 11 5 20 10 21 24 3 13 23 17 14 12 8 4 6 15 1 7 19 2 16

Czas wykonania: 22 milisekund
========= Przetwarzanie danych:  data.2 =========
Cmax: 20919
Kolejnosc: 2 23 22 21 12 19 18 17 16 24 14 13 20 11 10 9 8 7 6 5 4 3 1 15

Czas wykonania: 22 milisekund
========= Przetwarzanie danych:  data.3 =========
Cmax: 33156
Kolejnosc: 9 10 44 30 48 42 29 22 12 47 35 15 33 26 43 37 19 14 5 16 31 2 34 28 41 21 45 23 4 40 24 3 38 11 27 17 18 36
8 46 1 7 39 20 6 25 13 32

Czas wykonania: 156 milisekund
========= Przetwarzanie danych:  data.4 =========
Cmax: 33878
Kolejnosc: 37 19 45 17 20 21 41 25 29 12 8 7 23 24 15 43 6 48 27 10 32 46 16 26 33 31 36 2 4 42 13 47 30 39 1 18 34 35 2
8 14 9 40 5 3 38 11 22 44

Czas wykonania: 168 milisekund
==================================
Laczny czas wykonania: 368 milisekund
Laczne Cmax: 101919
```

Figure 3: Wyniki wraz z czasem wykonywania algorytmu dla maszyny używającej systemu operacyjnego Windows

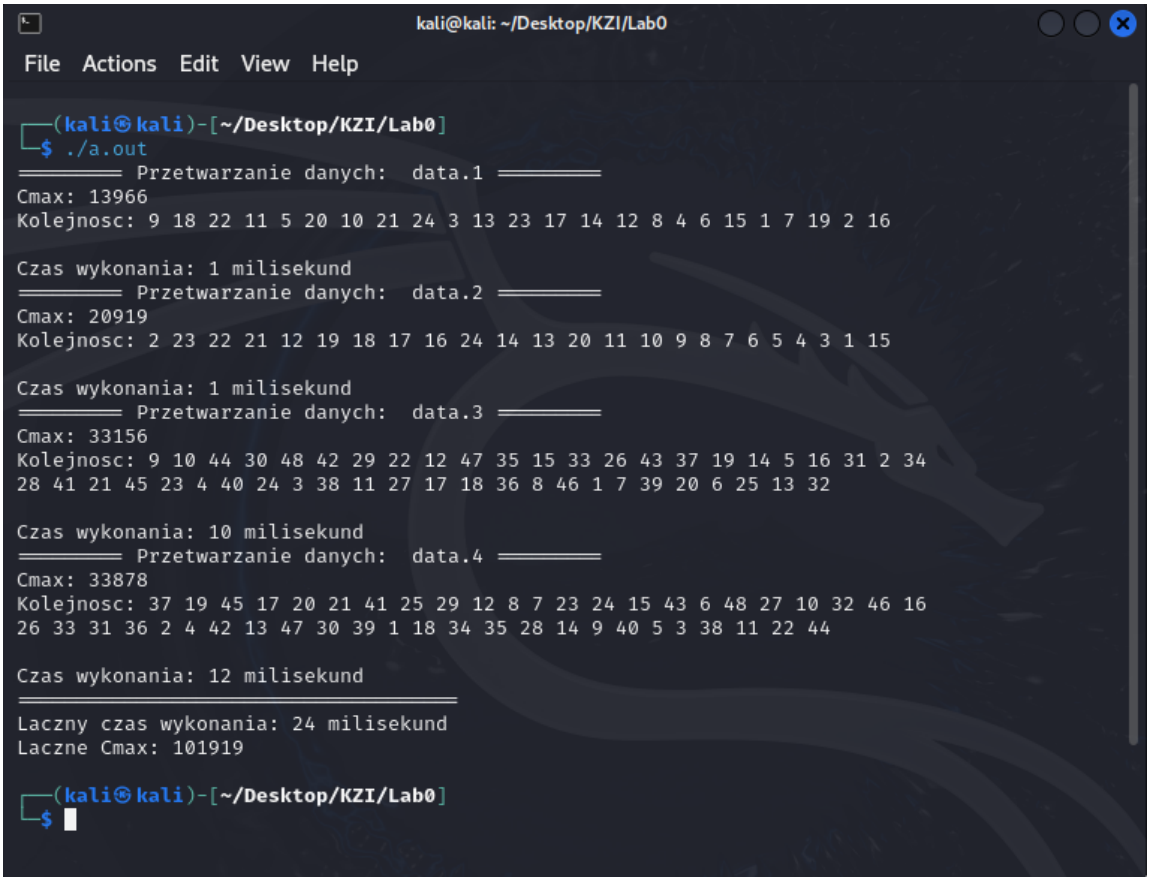

Figure 4: Wyniki wraz z czasem wykonywania algorytmu dla maszyny używającej systemu operacyjnego Linux

4 Sugerowana ocena

Na podstawie otrzymanych wyników oraz przedstawionych nam kryteriów oceny algorytmów uważam, że adekwatną oceną było by **3.5**.