Politechnika Wrocławska

Wydział Elektroniki, Fotoniki i Mikrosystemów

Sterowanie Procesami Dyskretnymi

Oliwier Woźniak, Dzmitry Mandrukevich

kierunek studiów: Automatyka i robotyka

specjalność: Robotyka

Wpływ różnych czynników na efektywność pracy algorytmu Carliera

Prowadzący: dr inż. Radosław Grymin

Wrocław 8 maja 2024

Spis treści

1 Opis problemu **2**

1.1 Algorytm Carliera 2

2 Przedstawienie naszego rozwiązania **4**

2.1 Badania działania algorytmu 5

3 Analiza czynników wpływających na efektywność algorytmu **8**

3.1 Wpływ strategii przeglądania 8

3.1.1 Deep left 9

3.1.2 Wide left 10

3.1.3 Deep greedy 11

3.1.4 Greedy 11

3.2 Wpływ dodatkowych testów eliminacyjnych 12

4 Podsumowanie **13**

1 Opis problemu

Celem zadania jest znalezienie algorytmu, znajdującego optymalną permutację zbioru zadań, dla których suma czasów wykonywania jest minimalna. Każde z zadań ma 3 parametry: r, p oraz q. Pierwszy z nich oznacza czas przygotowania zadania, p to czas przez jaki zadanie będzie wykonywane, natomiast q to czas stygnięcia, czyli ile zadanie musi czekać przed jego zakończeniem.

1.1 Algorytm Carliera

Działanie algorytmu Carliera jest oparte o Algorytm Schrange, oraz Schrange z podziałem. Jest to pierwszy algorytm rekurencyjny omawiany na zajęciach, który przeszukuje drzewo rozwiązań, poprzez rekurencyjne wywoływanie samego siebie. Wariacje algorytmu Schrange służą jako wskaźniki mówiące o kontynuacji, lub zaprzestaniu przeszukiwania możliwych rozwiązań. Do prawidłowego działania algorytmu należy wyznaczyć 5 podstawowych parametrów (i wiele innych, pomocniczych), są to:

- UB – górne ograniczenie wyznaczane na postawie algorytmu Schrange, jest to suma czasu wykonania zadań w wynikowym ułożeniu. Najgorszy przypadek, którego wartość staramy się zredukować.
- LB – dolne ograniczenie wyznaczane na podstawie algorytmu Schrane z podziałem. Jeśli ta wartość jest mniejsza od UB to powinniśmy kontynuować przeszukiwanie drzewa.
- b – numer zadania w obecnej permutacji, które jest ostatnie na ścieżce krytycznej (dokładny opis w [2]).
- a – numer zadania o najniższym indeksie, które nie ma żadnych przestojów do zadania "b".
- c – numer zadania krytycznego, którego czas dostarczania jest mniejszy niż czas dostarczania zadania o numerze b.

Algorytm 1 Pseudokod dla algorytmu Carlier.

procedure CARLIER(n, R, P, Q)
  $U \leftarrow$ SCHRAGE(n, R, P, Q)
  if $U < UB$ **then**
    $UB \leftarrow U$, $\pi^* \leftarrow \pi$
  end if
  $b \leftarrow \max\left\{j \in \mathcal{N} : 1 \leq j \leq n \wedge C_{max}(\pi) = C_{\pi(j)} + q_{\pi(j)}\right\}$
  $a \leftarrow \min\left\{j \in \mathcal{N} : 1 \leq j \leq n \wedge C_{max}(\pi) = r_{\pi(j)} + \sum_{s=j}^{b} p_{\pi(s)} + q_{\pi(b)}\right\}$
  $c \leftarrow \max\left\{j \in \mathcal{N} : a \leq j \leq b \wedge q_{\pi(j)} < q_{\pi(b)}\right\}$
  if $c = \emptyset$ **then**
    return π^*
  end if
  $\mathcal{K} \leftarrow \{c+1, c+2, \ldots, b\}$
  $r(\mathcal{K}) \leftarrow \min_{j \in \mathcal{K}} r_{\pi(j)}$, $q(\mathcal{K}) \leftarrow \min_{j \in \mathcal{K}} q_{\pi(j)}$, $p(\mathcal{K}) \leftarrow \sum_{j \in \mathcal{K}} p_{\pi(j)}$
  $r_{\pi(c)} \leftarrow \max\left\{r_{\pi(c)}, r(\mathcal{K}) + p(\mathcal{K})\right\}$
  $LB \leftarrow$ SCHRAGEPMTN(n, R, P, Q)
  $LB \leftarrow \max\{h(\mathcal{K}), h(\mathcal{K} \cup \{c\}), LB\}$
  if $LB < UB$ **then**
    CARLIER(n, R, P, Q)
  end if
  odtwórz $r_{pi(c)}$
  $q_{\pi(c)} \leftarrow \max\left\{q_{\pi(c)}, q(\mathcal{K}) + p(\mathcal{K})\right\}$
  $LB \leftarrow$ SCHRAGEPMTN(n, R, P, Q)
  $LB \leftarrow \max\{h(\mathcal{K}), h(\mathcal{K} \cup \{c\}), LB\}$
  if $LB < UB$ **then**
    CARLIER(n, R, P, Q)
  end if
  odtwórz $q_{\pi(c)}$
end procedure

Rysunek 1: Pseudokod algorytmu Carliera

Algorytm jest skomplikowany i można założyć, że skomplikowany obliczeniowo. W związku z tym stosuje się wszelkie sposoby, aby ograniczyć stos wywołań.

2 Przedstawienie naszego rozwiązania

Nasza implementacja algorytmu Cariera jest prymitywna i zachłanna, jednak znajduje prawidłowe rozwiązania w skończonym czasie. Poniżej znajdują się funkcje realizujące algorytm.

```
void Blok (int n, Dane* dane, int* Kolejnosc, int& Id, int& R, int& Q)
{
    int pos=-1, m=0, Cmax=0;
    int tmp[n];
    for (int i=0; i<n; i++){
        int j=Kolejnosc[i];
        tmp[i] = (m>=dane[j].r);
        m = max(m, dane[j].r) + dane[j].p;
        if(Cmax < m+dane[j].q){
            Cmax = m+dane[j].q;
            pos = i;
        }
    }
    int i=pos, j=-1;
    Dane pomoc = dane[Kolejnosc[pos]];
    while (tmp[i]){
        if (dane[Kolejnosc[--i]].q < pomoc.q){
            j = Kolejnosc[i];
            break;
        }
        pomoc.r = min(pomoc.r, dane[Kolejnosc[i]].r);
        pomoc.p += dane[Kolejnosc[i]].p;
    }
    Id = j;
    R = pomoc.r + pomoc.p;
    Q = pomoc.q + pomoc.p;
}
```

Rysunek 2: Funkcja odnajdująca ścieżkę krytyczną

```
void carlier (int n, Dane* dane, int* Kolejnosc, int& UB)
{
    if(schragePodzial(n, dane) >= UB){
        return;
    }
    int Cmax = schrage(n, dane, Kolejnosc);
    if(Cmax < UB){
        UB = Cmax;
    }
    int Id, R, Q;
    Blok(n, dane, Kolejnosc, Id, R, Q);
    if(Id < 0){
        return;
    }
    int pomR = dane[Id].r;
    int pomQ = dane[Id].q;
    dane[Id].r = R;
    carlier(n, dane, Kolejnosc, UB);
    dane[Id].r = pomR;
    dane[Id].q = Q;
    carlier(n, dane, Kolejnosc, UB);
    dane[Id].q = pomQ;
}
```

Rysunek 3: Funkcja realizująca algorytm Carliera

2.1 Badania działania algorytmu

Wykonaliśmy jedynie testy czasowe na danych dostępnych na stronie dr. Mariusza Makuchowskiego, Poniżej znajdują się wyniki testów.

Dane nr.0				
	Obliczenia	Odpowedź	Poprawność	Czas wykonania [s]
Podzial:	221	221	true	2,00E-06
Schrage:	283	283	true	2,00E-06
Carlier:	228	228	true	4,00E-06

Dane nr.1				
	Obliczenia	Odpowedź	Poprawność	Czas wykonania [s]
Podzial:	3026	3026	true	3,90E-05
Schrage:	3109	3109	true	3,60E-05
Carlier:	3026	3026	true	9,92E-04

Dane nr.2				
	Obliczenia	Odpowedź	Poprawność	Czas wykonania [s]
Podzial:	3654	3654	true	2,90E-05
Schrage:	3708	3708	true	2,90E-05
Carlier:	3665	3665	true	8,75E-04

Dane nr.3				
	Obliczenia	Odpowedź	Poprawność	Czas wykonania [s]
Podzial:	3309	3309	true	2,00E-05
Schrage:	3353	3353	true	2,00E-05
Carlier:	3309	3309	true	9,50E-04

Dane nr.4				
	Obliczenia	Odpowedź	Poprawność	Czas wykonania [s]
Podzial:	3172	3172	true	3,10E-05
Schrage:	3235	3235	true	3,00E-05
Carlier:	3191	3191	true	7,55E-04

Dane nr.5				
	Obliczenia	Odpowedź	Poprawność	Czas wykonania [s]
Podzial:	3618	3618	true	3,10E-05
Schrage:	3625	3625	true	3,10E-05
Carlier:	3618	3618	true	8,60E-05

Dane nr.7				
	Obliczenia	Odpowedź	Poprawność	Czas wykonania [s]
Podzial:	3820	3820	true	2,90E-05
Schrage:	3862	3862	true	3,00E-05
Carlier:	3821	3821	true	5,64E-04

Dane nr.8				
	Obliczenia	Odpowedź	Poprawność	Czas wykonania [s]
Podzial:	3633	3633	true	3,60E-05
Schrage:	3645	3645	true	3,90E-05
Carlier:	3634	3634	true	8,06E-04

Jak widać na podstawie tabel, Carlier ma zdecydowanie większą złożoność obliczeniową niż algorytmy w nim używane co jest spodziewane. Jednak wszystkie wyniki są zgodne z wynikami podanymi w zadaniu.

3 Analiza czynników wpływających na efektywność algorytmu

Ze względu na ograniczoną implementację z naszej strony, oraz szeroki zakres narzędzi dostarczonych przez dr. Mariusza Makuchowskiego, postanowiliśmy dokonać analizy na podstawie programu dostarczonego w ramach przygotowania do wykonania laboratorium.

3.1 Wpływ strategii przeglądania

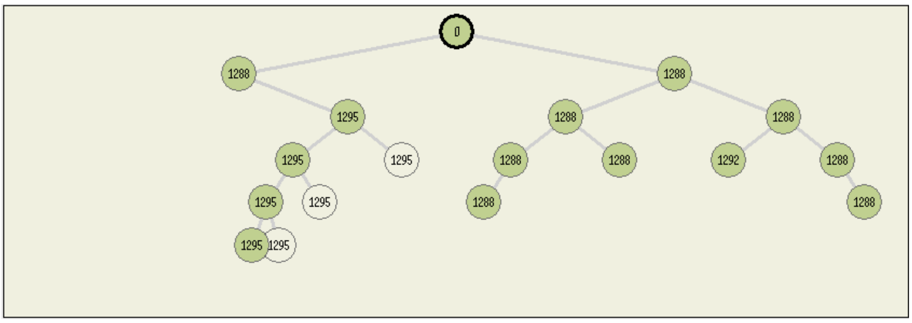
W programie dostarczonym przez dr. Makuchowskiego, dostępne są 4 strategie przeglądania:

- Deep left (first) – przeszukiwanie głębokie, skupiające swoją uwagę na pierwszym odnóżu grafu. Jest on przeszukiwany "w głąb"do momentu zastania jakiegoś warunku końcowego. Dopiero po przeszukaniu całej "lewej"strony grafu, przechodzi on do analizy alternatywnych ścieżek.
- Wide left (first) – przeszukiwanie alternatywne do przeszukiwania głębokiego. Na każdym kroku przed przystąpieniem do analizy kolejnego poziomu, sprawdzane są wszystkie możliwości na obecnym poziomie. Podobnie jak wcześniej graf przeszukiwany jest od lewej strony.
- Deep greedy – przeszukiwanie w głąb, wybierające ścieżki do analizy na podstawie wartości w następnych węzłach.
- Greedy – przeszukiwanie analogiczne do poprzedniego, tym razem jednak nie skupiamy się na przeszukiwanie każdej możliwej alternatywy w głąb. Pomimo niepochlebnej nazwy w wielu przypadkach jest najszybszy.

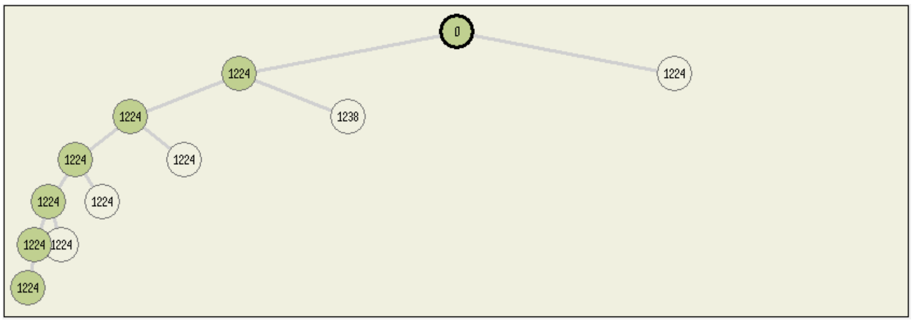
Wszystkie metody mają przypadki gdzie sprawdzają się lepiej bądź gorzej, wszystko zależy od tego gdzie w drzewie rozwiązań leży przypadek optymalny. We wszystkich przypadkach zdjęcia są ustawione zgodnie z kolejnością wykonywanych testów, więc drzewa przeszukiwań tyczą się tych samych problemów.

3.1.1 Deep left

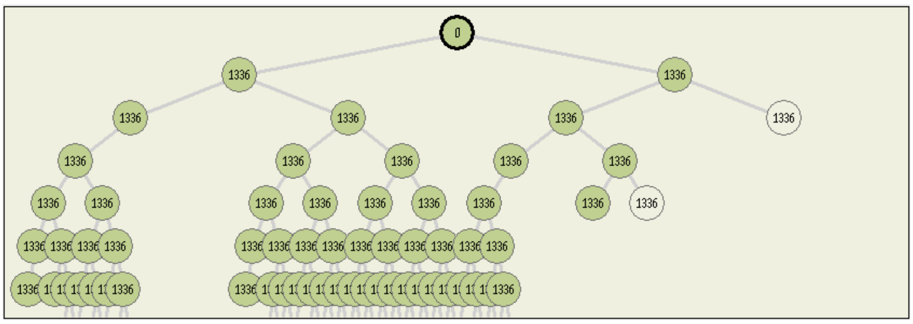
Ta metoda jest dobra dla drzew, gdzie rozwiązanie leży nisko i po lewej stronie. Pozwala to na szybkie wykluczenie wszystkich następnych ścieżek i zakończenie przeszukiwania. Im dalej na prawo jest rozwiązanie, tym więcej ścieżek trzeba przeszukać, jest to szczególnie złe dla drzew których wartości są malejące w prawą stronę, w takim przypadku algorytm przeszukuje całość. Poniżej znajduje się wynik przeszukiwania dla przykładowych wartości.



Rysunek 4: Przeszukiwanie głębokie, przypadek zwykły



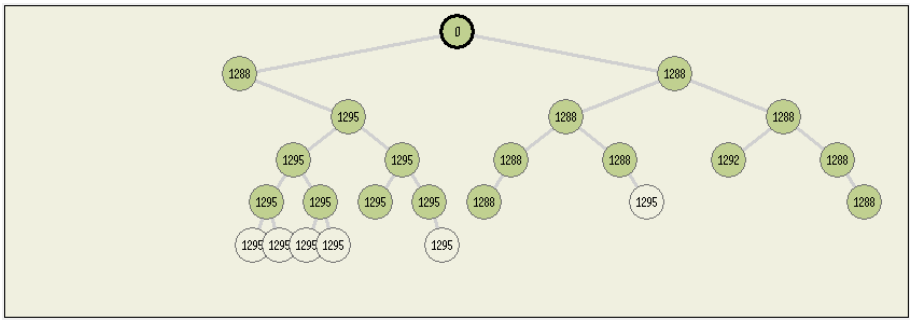
Rysunek 5: Przeszukiwanie głębokie, przypadek nieoptymalny



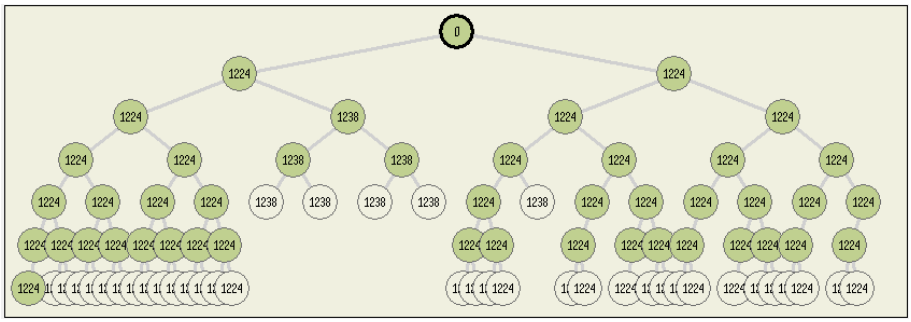
Rysunek 6: Przeszukiwanie głębokie, przypadek zbliżony do najgorszego

3.1.2 Wide left

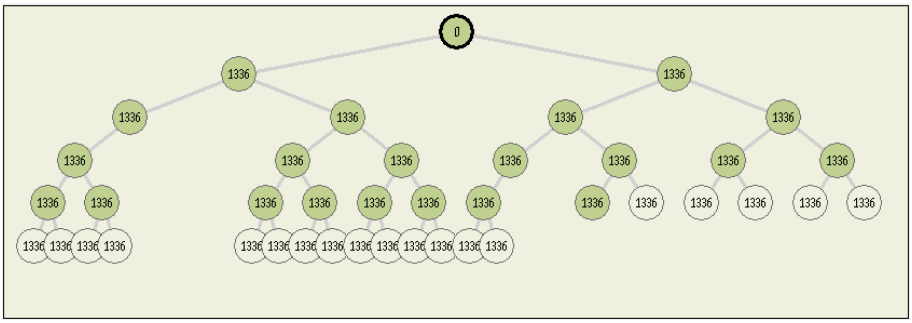
Ta metoda jest dobra dla drzew, gdzie rozwiązanie leży wysoko, aspekt związany z kolejnością przeszukiwań jest mniej widoczny. Dla tej metody przeszukiwań bardzo często przeszukujemy dużą część drzewa, zanim znajdziemy prawidłowe rozwiązanie. Sprawia to, że algorytm ma bardzo wąskie zastosowania.



Rysunek 7: Przeszukiwanie szerokie, przypadek przeciętny



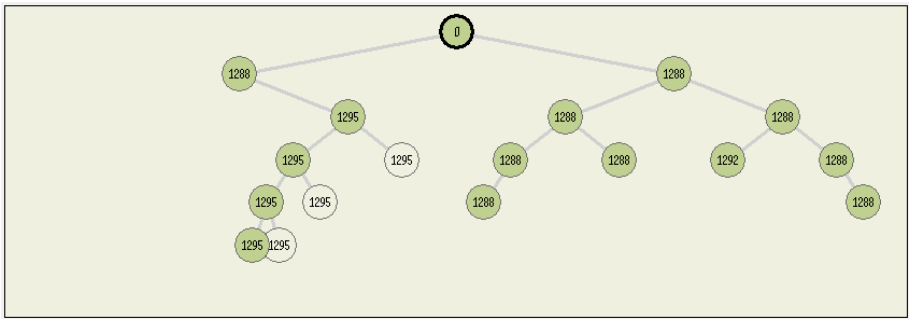
Rysunek 8: Przeszukiwanie szerokie, przypadek zbliżony do najgorszego



Rysunek 9: Przeszukiwanie szerokie, przypadek przeciętny

3.1.3 Deep greedy

Ta metoda jest dobra dla drzew, gdzie rozwiązanie leży nisko, podobnie jak w przypadku pierwszego wyszukiwania. Z resztą przypadki również wydają się mieć podobne wyniki. Tutaj staje się dość wyraźny problem w przeszukiwaniu tych drzew rozwiązań: zmiany są minimalne i względnie losowe, co sprawia że ciężko jest dobrać metodę optymalną dla problemu. Przeszukiwanie takich drzew w sposób usystematyzowany może być problematyczne, a czasem nawet niemożliwe.

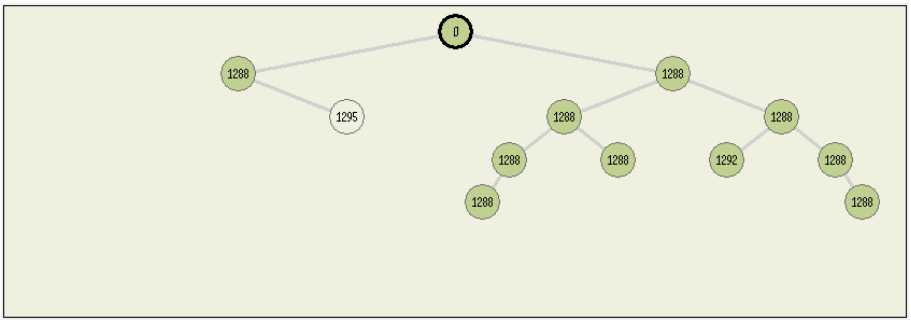


Rysunek 10: Przeszukiwanie chciwe w głąb

W tym przykładzie widać podobieństwo pomiędzy tym rozwiązaniem, a rozwiązaniem za pomocą metody "Deep left". Następne przypadki mają podobną charakterystykę, w związku z tym nie umieszczam ich w sprawozdaniu.

3.1.4 Greedy

Ta metoda jest dobra dla drzew, gdzie rozwiązanie leży za wartościami malejącymi, które niekoniecznie możemy znaleźć w drzewie. Metoda jest zachłanna (jak nazwa mówi), jednak z odpowiednimi testami eliminacyjnymi, oraz zestawami danych może dawać zadowalające wyniki.



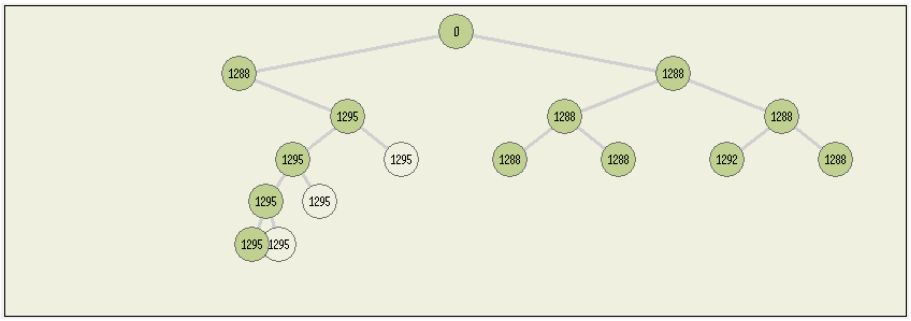
Rysunek 11: Przeszukiwanie chciwe

Podobnie jak wcześniej, rozwiązania są podobne do tych z pierwszej metody, w związku z tym nie umieszczam ich w sprawozdaniu.

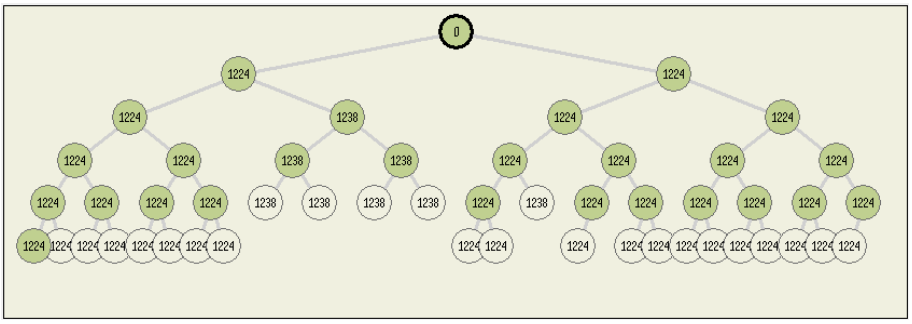
3.2 Wpływ dodatkowych testów eliminacyjnych

Dodatkowe testy eliminacyjne mają na celu ukrócić przeszukiwanie drzewa bez eliminacji rozwiązania optymalnego. Pozwala to na nieznaczne przyśpieszenie wykonywania algorytmu.

Wizualna reprezentacja dla pierwszego zestawu zadań:



Rysunek 12: Przeszukiwanie w głąb, analogiczny do 4



Rysunek 13: Przeszukiwanie szerokie, analogiczne do 7

Uważam że te przykłady są wystarczającą reprezentacją różnic w ucinanych gałęziach, zazwyczaj jest ich niewiele, a wyniki wciąż pozostają identyczne. Ich wpływ na działanie algorytmu jest nieznaczny w porównaniu do wysiłku programistycznego jaki wymagają.

4 Podsumowanie

Poprawnie zaimplementowany algorytm Carliera oferuje poprawę wyników algorytmów Schrange w sposób przewidywalny i stabilny, jednak zadaje duży narzut obliczeniowy na działanie programu, co sprawia że nie nadaje się do rozwiązywania większych problemów.

Metody przeszukiwań drzew nie mają dużego wpływu na średnią szybkość działania algorytmu, ponieważ są zależne od zmiennych wejściowych i tego jakie drzewo one rozpinają. Sprawia to, że ciężko jest jednoznacznie określić która metoda jest najlepsza do tych zastosowań.

Dodatkowa eliminacja ma niewielki wpływ na ilość przeszukiwanych gałęzi, co sprawia że praktycznie przy każdym wywołaniu oszczędzamy trochę czasu. Jednak ten czas jest nieznaczny w porównaniu do wyboru odpowiedniej metody przeszukiwania drzewa.

Bibliografia

[1] Andrzej Gnatowski. “Lab. 03: Algorytm Carlier”. 2019. URL: http://andrzej.gnatowski.staff.iiar.pwr.wroc.pl/SterowanieProcesamiDyskretnymi/lab03_carlier/instrukcja/lab03.pdf.

[2] Andrzej Gnatowski Teodor Nizyński. “STEROWANIE PROCESAMI DYSKRETNYMI Lab 5: Algorytm Carlier”. 2019. URL: http://teodornizynski.com/spd_5_2019.pdf.

[3] “Depth First Search or DFS for a Graph”. 2024. URL: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/.