Projektowanie Efektywnych Algorytmow

Projekt

16/11/2021

252690 Karolina Nogacka

(3)Branch & Bound

spis treści	strona

1. **Sformułowanie zadania:**

   Zadanie polega na stworzeniu algorytmu znajdującego rozwiązanie problemu komiwojażera opartego o metodę podziału i ograniczeń.

   Problem komiwojażera (TSP - Traveling salesman problem):

   Zakładając, że ktoś (np. kurier, sprzedawca, listonosz) ma zbiór miast/domów do odwiedzenia szukamy drogi, która zawierać będzie wszystkie wyżej wymienione miejsca i będzie drogą najkrótszą (aby listonosz nie musiał się nachodzić).

   Innymi słowy problem ten jest zagadnieniem optymalizacyjnym , polegającym na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Wszystkich cykli w takim grafie będzie *n*! (w związku z czym złożoność dla większych grafów będzie szybko rosła), pewne z nich będą w rzeczywistości tymi samymi cyklami, zaczynającymi się jednak w różnych węzłach grafu.

2. Metoda:

Metoda podziału i ograniczeń polega na analizie drzewa przestrzeni rozwiązań. Przechodzenie drzewa rozpoczyna się od korzenia i kończy się w liściu (gdy wszystkie wierzchołki dołączone są do rozwiązania). Kluczową częścią tej metody jest ograniczanie przestrzeni przez poprawnie postawione ograniczenia. Jeśli wiemy już, że rozwijanie pewnej ścieżki w drzewie nie będzie opłacalne (bo przekraczane są ograniczenia, np. aktualny koszt ścieżki jest większy od kosztu minimalnego cyklu), nie rozwijamy dalej tej ścieżki.

Można wyróżnić trzy sposoby przechodzenia przestrzeni:

- **breadth search**, przeszukiwanie drzewa wszerz (pomocna struktura danych: kolejka)
- **depth search**, przeszukiwanie drzewa w głąb (pomocna struktura danych: stos)
- **low cost**, best search (kolejne decyzje poruszania się po drzewie podejmujemy na podstawie kosztów)

Jako dolne ograniczenie przyjęta jest tutaj wartość MST (minimalnego drzewa spinającego) wyznaczona algorytmem Prima. Jako górne ograniczenie przyjęta jest dwukrotność MST.

Złożoność czasowa algorytmów opartych na metodzie BxB (Brancz & Bound) w dużej części zależy od narzuconych ograniczeń (górnego i dolnego). Gdyby takie ograniczenia były bardzo słabe, bądź w ogóle by ich nie było, algorytmy oparte na tej metodzie sprowadzamy by się do brute force (**O(*n*!)**). Złożoność nie powinna być więc wyższa niż c * n!.

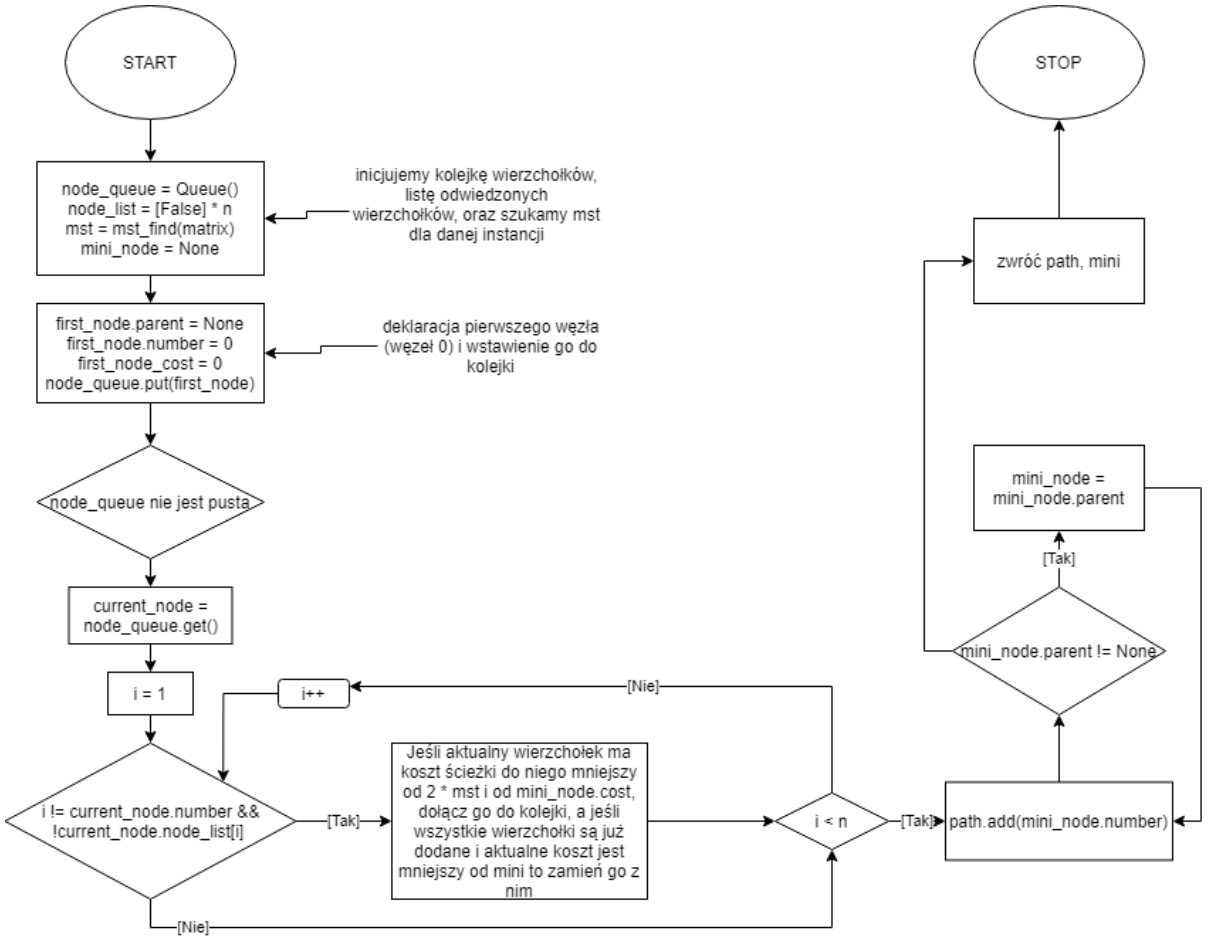
3. Algorytmy:

Algorytm z przeszukiwaniem **wszerz**, ustawia wierzchołki w kolejce zaczynając od pierwszego wierzchołka. Odwiedza jego wszystkich sąsiadów, dodając je w kolejności rosnącej do kolejki.

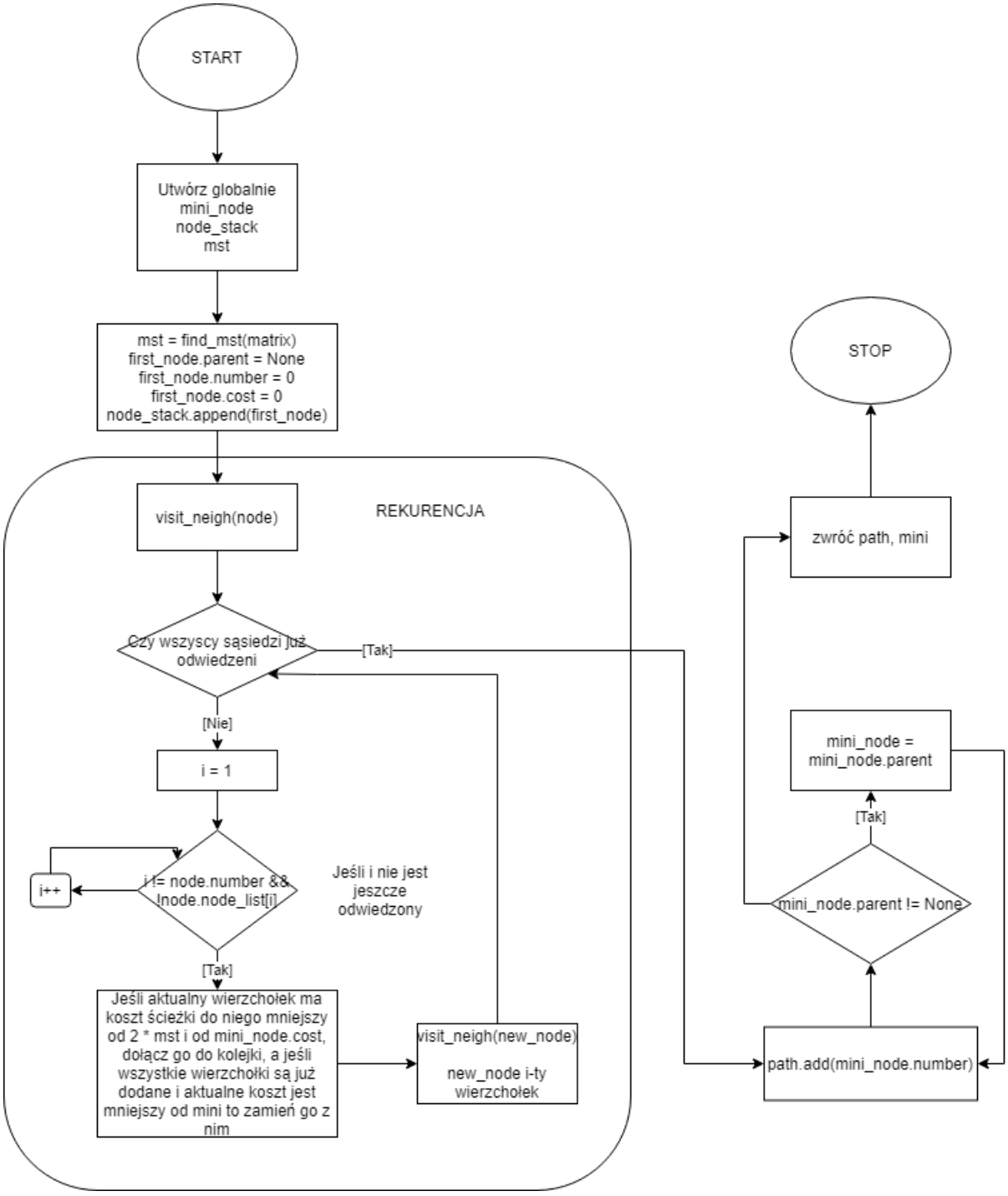
Algorytm z przeszukiwaniem w **głąb** wykorzystuje stos. Zaczyna od pierwszego wierzchołka, a następnie wykorzystując rekurencję odwiedza pierwszego sąsiada, a potem pierwszego sąsiada tego sąsiada. Dodaje wierzchołki na stos, a gdy przejdzie już taką ścieżkę, która zawiera wszystkie wierzchołki, cofa się po stosie, aż znajdzie pierwszego nieodwiedzonego sąsiada w wierzchołków na tym stosie.

Algorytm z przeszukiwaniem przestrzeni **best search\low cost** wykorzystuje kolejkę priorytetową. Zaczynami od pierwszego wierzchołka, podjęcie decyzji, jaki wierzchołek powinien zostać następnie odwiedzony, przebiega na podstawie wyznaczonego kosztu z obecnie analizowanej ścieżki do tego wierzchołka (wybierany jest ten wierzchołek, który zapewni uzyskanie ścieżki o najmniejszym koszcie).

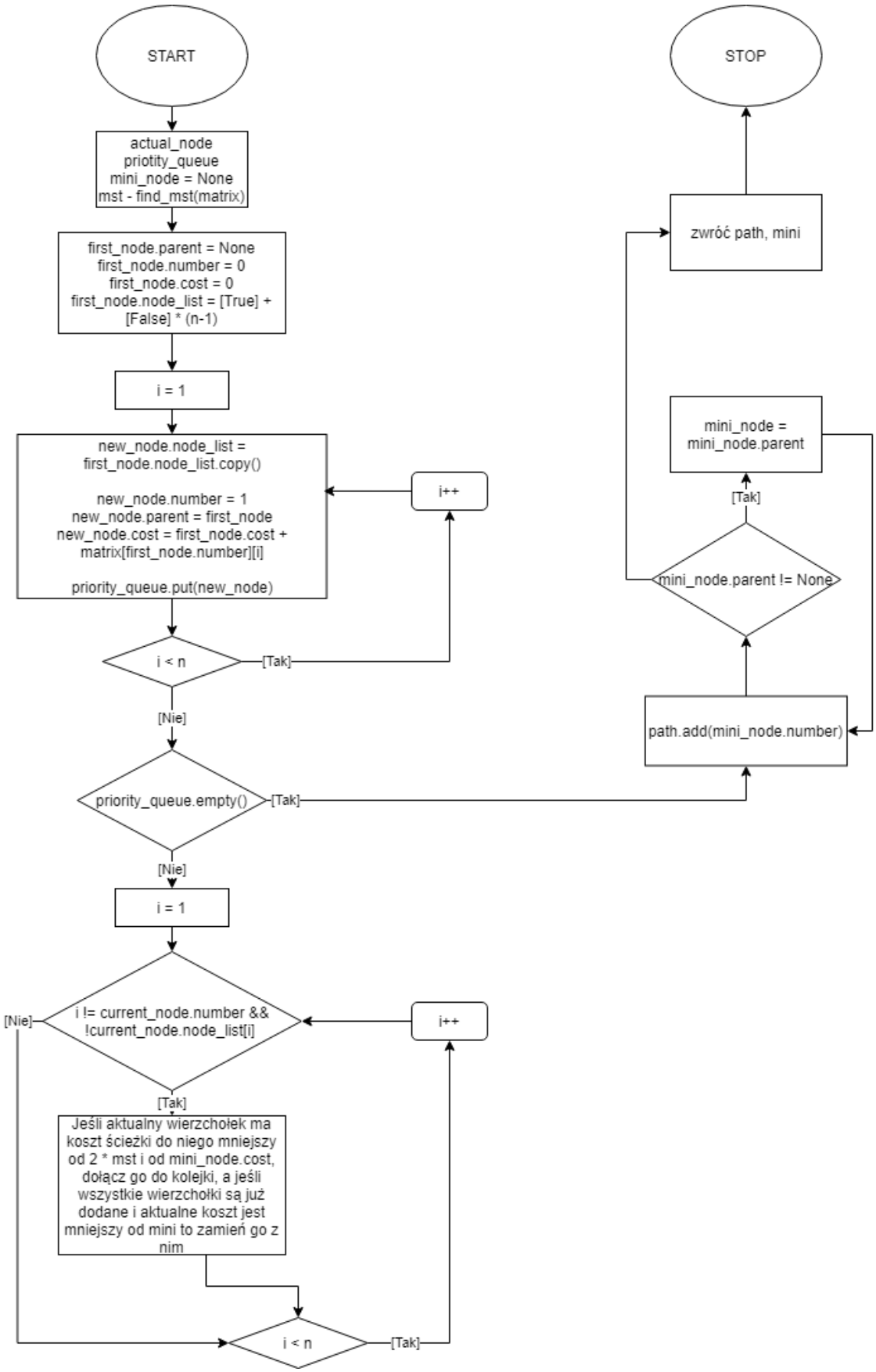
Wszystkie z metod mają wyżej wspomniane ograniczenia, które sprawdzamy, aby skrócić czas wykonywania (dolne ograniczenie - mst, górne ograniczenie – 2 * mst).



Rysunek 1. Schemat blokowy algorytmu znajdującego rozwiązanie problemu komiwojażera, opartego na metodzie podziału i ograniczeń z przeszukiwaniem wszerz.



Rysunek 2. Schemat blokowy algorytmu znajdującego rozwiązanie problemu komiwojażera, opartego na metodzie podziału i ograniczeń z przeszukiwaniem w głąb.



Rysunek 3. Schemat blokowy algorytmu znajdującego rozwiązanie problemu komiwojażera, opartego na metodzie podziału i ograniczeń z przeszukiwaniem best serach.

4. Dane wejściowe i wyjściowe

Co zawiera program badawczy:

- Pliki wykonywalne tsp_bf.exe, tsp_df.exe, tsp_bs.exe
  Aby działał poprawnie musi być w jednym katalogu z plikiem config.ini i pliki tekstowe z grafami.
  tsp_bf.exe – wszerz
  tsp_df.exe – w głąb
  tsp_bs.exe – best search

- Pliki tsp_6_1.txt, tsp_6_2.txt, tsp_10.txt, tsp_12.txt, itd.
  Zawierają grafy w postaci macierzy sąsiedztwa, które będziemy badać. Źródło: http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/ [data dostępu: 13.11.2021, 17:48]

- Pliki gr21.tsp, ulysses22.tsp, br17.atsp, itd.
  Zawierają grafy w postaci macierzy sąsiedztwa, które będziemy badać. Źródło: http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/pea.php [data dostępu: 13.11.2021, 17:54]

- Plik inicjujący config.ini

```
[data]
tsp_6_1 = tsp_6_1.txt
tsp_6_2 = tsp_6_2.txt
tsp_10 = tsp_10.txt
tsp_12 = tsp_12.txt
tsp_13 = tsp_13.txt
tsp_14 = tsp_14.txt
tsp_15 = tsp_15.txt
tsp_17 = tsp_17.txt
tsp_6_1_opti = 132; [0, 1, 2, 3, 4, 5, 0]
tsp_6_2_opti = 80; [0, 5, 1, 2, 3, 4, 0]
tsp_10_opti = 212; [0, 3, 4, 2, 8, 7, 6, 9, 1, 5, 0]
tsp_12_opti = 264; [0, 1, 8, 4, 6, 2, 11, 9, 7, 5, 3, 10, 0]
tsp_13_opti = 269; [0, 10, 3, 5, 7, 9, 11, 2, 6, 4, 8, 1, 12, 0]
tsp_14_opti = 282; [0, 10, 3, 5, 7, 9, 13, 11, 2, 6, 4, 8, 1, 12, 0]
tsp_15_opti = 291; [0, 12, 1, 14, 8, 4, 6, 2, 11, 13, 9, 7, 5, 3, 10, 0]
tsp_17_opti = 39; [0, 11, 13, 2, 9, 10, 1, 12, 15, 14, 5, 6, 3, 4, 7, 8, 16, 0]
```

```
gr21 = gr21.tsp
ulysses22 = ulysses22.tsp
gr24 = gr24.tsp
fri26 = fri26.tsp
bays29 = bays29.tsp
;att48 = att48.tsp
;eil51 = eil51.tsp
;berlin52 = berlin52.tsp
br17 = br17.atsp
ftv33 = ftv33.atsp
ftv35 = ftv35.atsp
ftv38 = ftv38.atsp
p43 = p43.atsp
```

```
gr21_opti = 2707
ulysses22_opti = 7013
gr24_opti = 1272
fri26_opti = 937
bays29_opti  = 2020
;att48_opti  = 10628
;eil51_opti  = 426
;berlin52_opti  = 7542
br17_opti  = 39
ftv33_opti  = 1286
ftv35_opti  = 1473
ftv38_opti  = 1530
p43_opti  = 5620

[result]
tsp_dp = tsp_dp.csv
tsp_bs = tsp_bs.csv
tsp_bf = tsp_bf.csv
tsp_df = tsp_df.csv
```

Rysunek 2. Zawartość pliku config.ini

Pierwsze zdjęcie zawiera nazwy plików ze strony http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/, oraz optymalne rozwiązania dla grafów w nich zawartych. Dwa pozostałe zdjęcia to nazwy i optymalne rozwiązania dla grafów ze strony http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/pea.php oraz nazwy plików wyjściowych.

Sekcja [data] zawiera nazwę plików wejściowych z grafami.
Sekcja [result] zawiera nazwę pliku wyjściowego.

- Skrypty tsp_bf.py, tsp_df.py, tsp_bs.py, my_writer.py:

Skrypty tsp_bf.py, tsp_df.py, tsp_bs.py są głównymi plikami programu, zawierają one wywołania metod z innych plików. my_writer.py zapewnia poprawny zapis do plików wynikowych.

- Pliki tsp_bf.csv, tsp_df.csv, tsp_bs.csv, analiza.csv:

Zawierają dane wyjściowe programu, a analiza.csv opracowane wyniki wraz z wykresami.

5. Procedura badawcza

- **Kolejność wykonywanych badań:**
    - Pobieranie danych z pliku inicjującego,
    - Wczytanie grafów z plików wejściowych,
    - Uruchomienie badań dla grafów.

Wynikiem działania programu (zapisywanym do pliku) jest czas znalezienia rozwiązania dla każdej instancji, długość najkrótszego cyklu, oraz sam cykl (w postaci listy węzłów).

Wyniki zostały opracowane po 20-krotnym uruchomieniu programu (algorytm wykonał się 20 razy dla każdej instancji). Jako górne ograniczenie działania programu przyjęto 10 minut. Podczas tego czasu algorytm z breadth search zdążył policzyć rozwiązania dla instancji wielkości max 12, depth serach dla wielkości max 14, a best search dla wielkości max 15.

Program nigdy nie wyczerpał pamięci komputera, jednak jego praca ponad 1 godzinę znacznie zakłócała prace komputera, w związku z czym algorytm został przerwany.

- **Specyfikacja sprzętu:**
    a. Procesor Intel i7-10510U, 1.80GHz – 2.30 GHz
    b. 16,0 GB pamięci ram
    c. System Windows 10 Home Edition

- **Metoda badania zużycia pamięci:**

```
127  def work(name, matrix):
128
129      print("Liczę dla: " + name)
130
131      start_time = time.time()
132      path, cost = low_cost(matrix)
133      end_time = time.time() - start_time
134
135      tsp_result = [name, len(matrix), end_time, memory(), cost, path]
136      # tsp_result = [name, len(matrix), end_time, min(paths_costs), paths[low_cost_index]]
137      return tsp_result
```

Rysunek 4. Metoda pomiaru czasu. Dla wszystkich wersji algorytmu wygląda analogicznie.

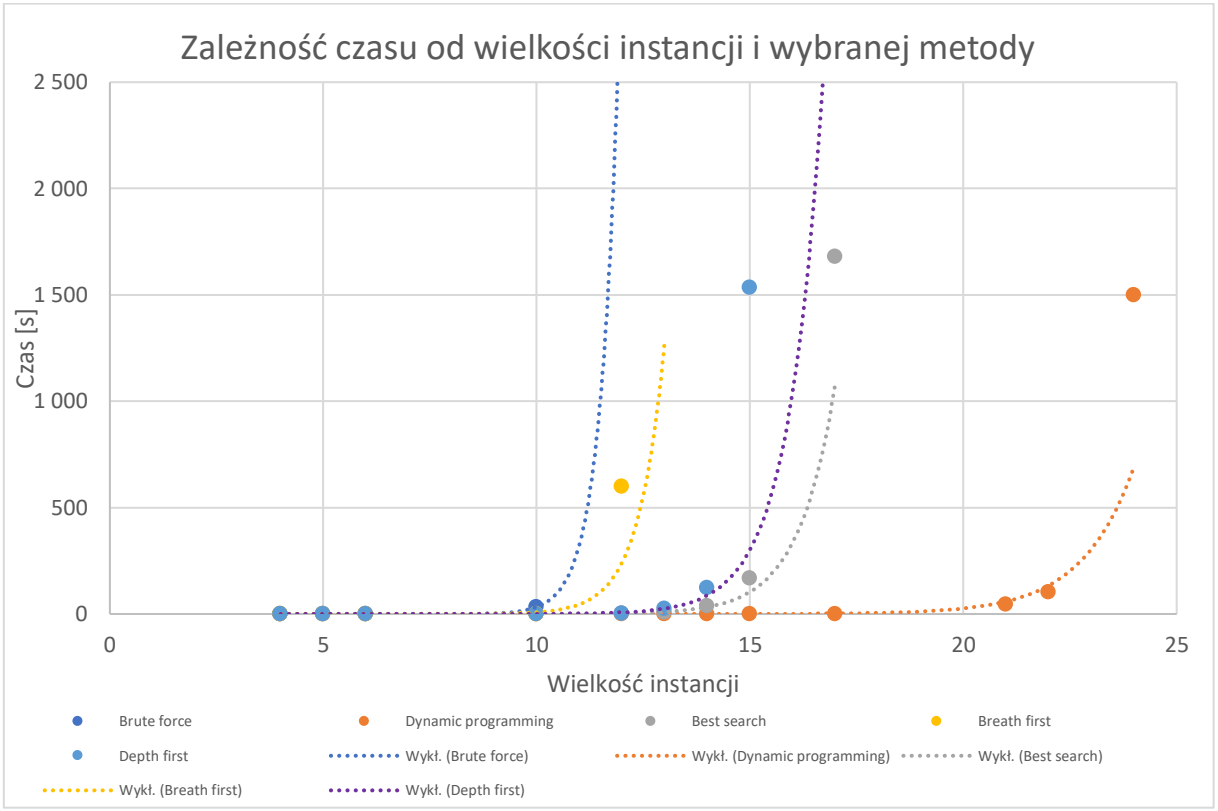
W funkcji *work()* przed i po uruchomieniu funkcji *low_cost()* mierzony jest czas za pomocą funkcji z biblioteki python'a *time. Różnica zapisywana jest do pliku wyjściowego jako czas wykonania algorytmu dla zadanej instancji.*

- **Metoda badania zużycia pamięci:**

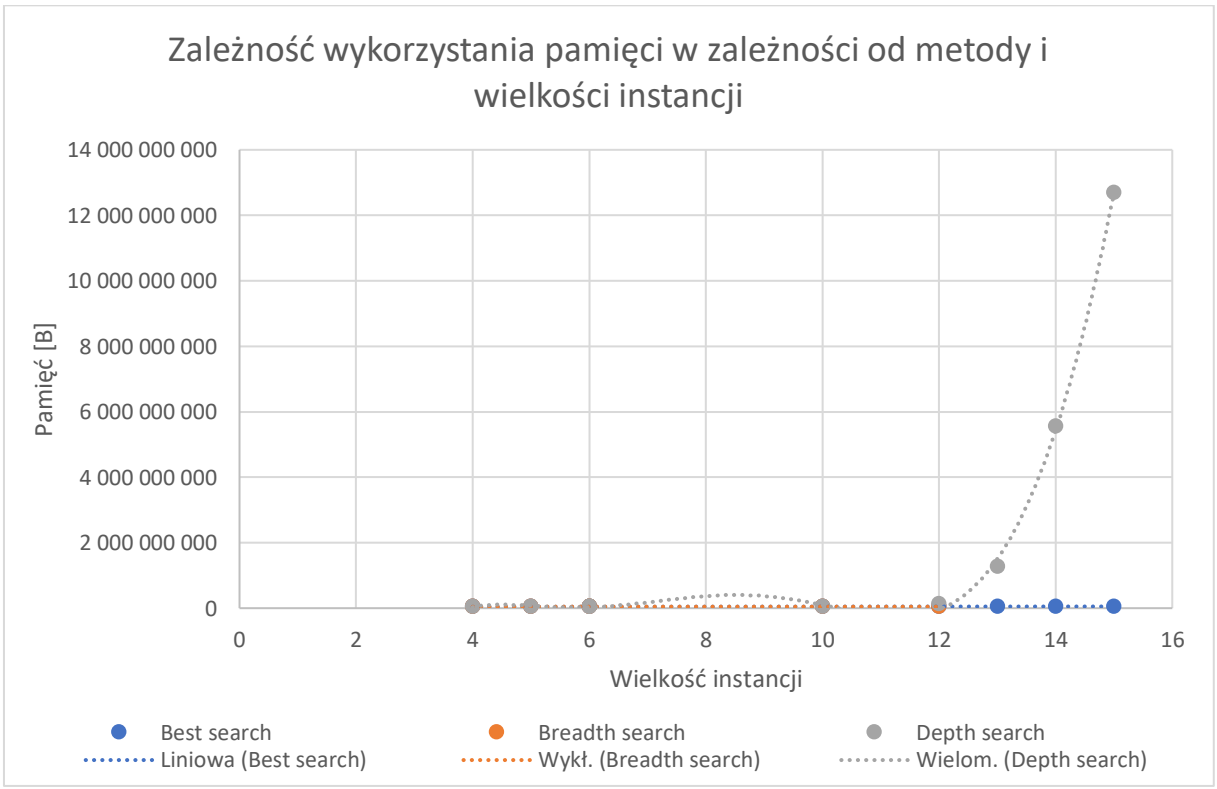
```
14  def memory():
15      w = wmi.WMI('.')
16      result = w.query("SELECT WorkingSet FROM Win32_PerfRawData_PerfProc_Process WHERE IDProcess=%d" % os.getpid())
17      return int(result[0].WorkingSet)
```

Rysunek 5. Metoda badania zużycia pamięci. Sprawdza pamięć zużywaną przez proces (działającą aplikację).

6. Wyniki



Rysunek 6. Zależność czasu pracy algorytmu od wielkości instancji i wybranej metody.



Rysunek 7. Zużycie pamięci w zależności od wybranej metody i wielkości instancji. Na wykresie nakładają się punkty dla breadth search i depth search.

7. Analiza wyników i wnioski

Z algorytmów zaimplementowanych na podstawie metody BxB najbardziej wydajny czasowo okazał się ten wykorzystujący best search (*Rysunek 6*.). Nie jest on jednak szybszy niż algorytm oparty na metodzie programowania dynamicznego z poprzedniego zadania. Może to wynikać z implementacji. Prawdopodobnie można było również znaleźć lepsze ograniczenia (dolne i górne), co spowodowałoby wyeliminowanie większych obszarów z przestrzeni rozwiązań. Zapewniło by to także pewną oszczędność pamięci.

Wszystkie trzy algorytmy opracowane w tym zadaniu (breath search, depth search i best search), zgodnie z założeniami okazały się szybsze od algorytmu opartego na metodzie brute force (*Rysunek 6*.). Czyli również oszacowanie złożoności czasowej okazało się poprawne.

Pod względem wykorzystania pamięci (*Rysunek 7.*) najkorzystniejszy okazała się również wersja best search (Jest ono niemal stałe), w dwóch pozostałych opracowanych algorytmach rośnie ono wykładniczo (na wykresie breath search i depth search nachodzą na siebie).