Projektowanie efektywnych algorytmów
Projekt

263934 Michał Pawlus

08.11.2023

Branch and Bound

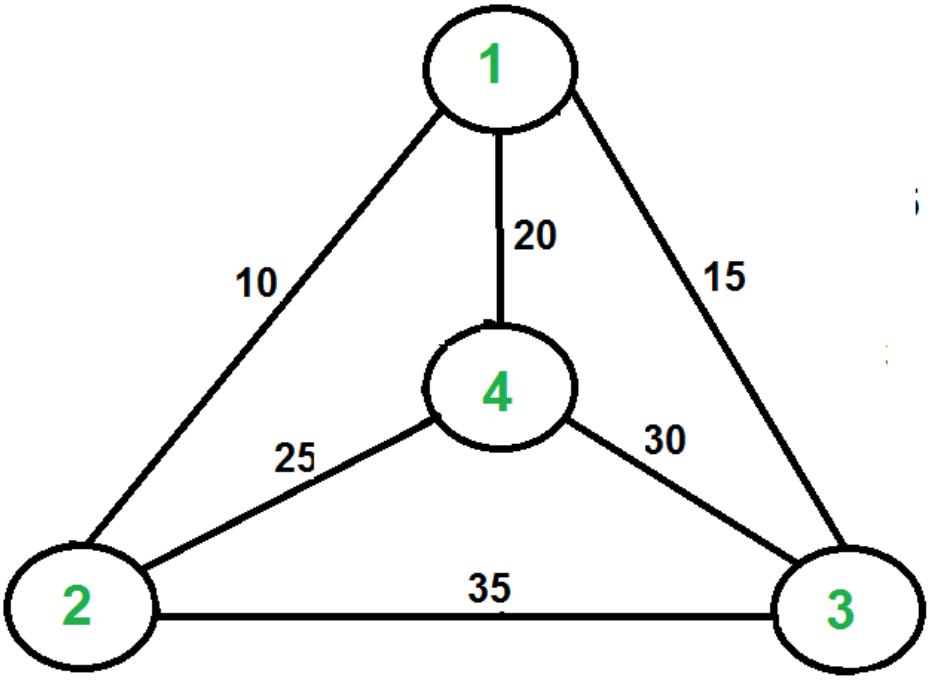
Spis treści	strona

1. Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu przeglądu zupełnego rozwiązującego problem komiwojażera w wersji optymalizacyjnej. Problem komiwojażera Polega na znalezieniu najkrótszej zamkniętej trasy, która odwiedza wszystkie zadane punkty (miasta) dokładnie raz i wraca do punktu początkowego.Taka trasa nazywana jest również ścieżką Hamiltona. Problem ten jest problemem NP-trudnym, co oznacza, że nie istnieje znany efektywny algorytm rozwiązujący ten problem w czasie wielomianowym.

Celem tego zadania jest badanie algorytmu Branch & Bound w kontekście rozwiązywania problemu komiwojażera.Algorytm ten zostaje zaimplementowany w języku Python, a następnie zbadany pod kątem jego efektywności i zdolności znajdowania optymalnych rozwiązań.

Przykładowa instancja problemu(najkrótsza ścieżka 1-2-4-3-1 ma koszt 80)



2. Metoda

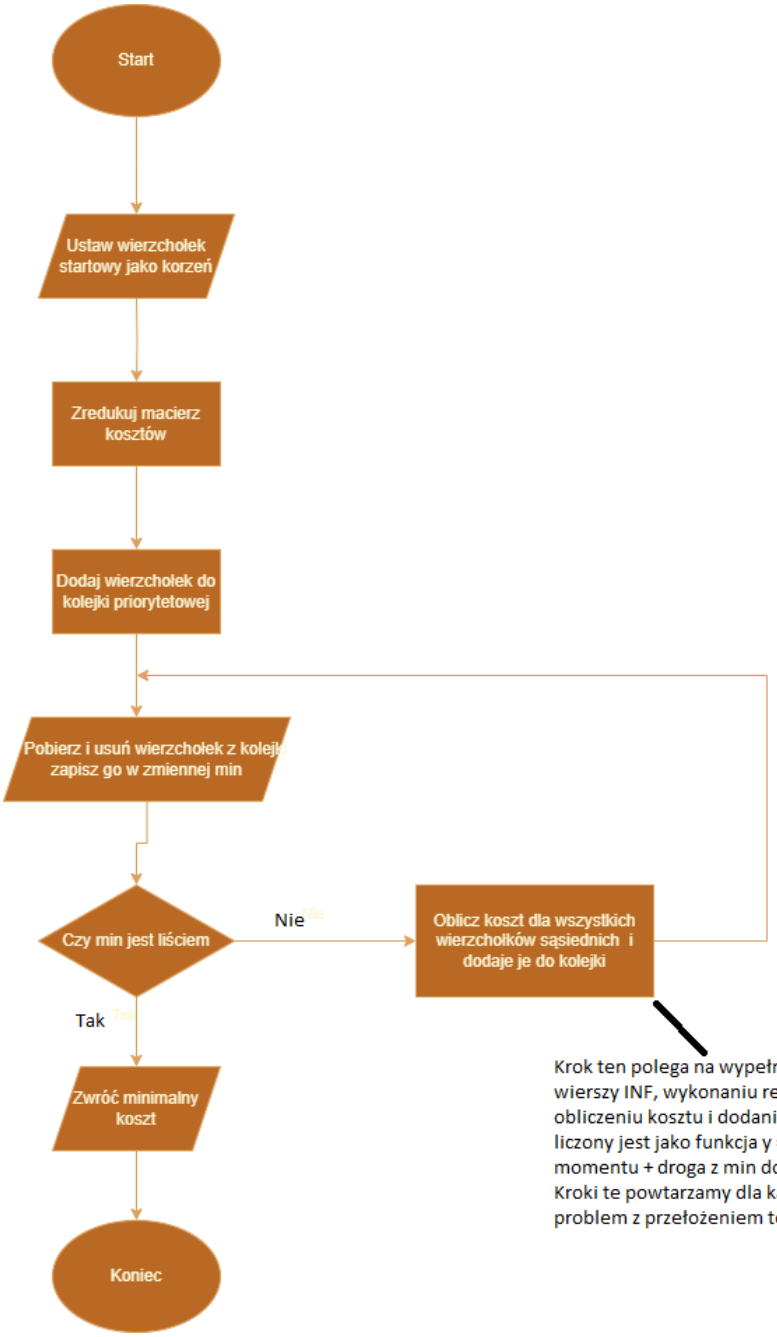
Metoda Branch and Bound (BnB) dla problemu komiwojażera polega na obliczaniu dolnych ograniczeń kosztów rozwinięcia ścieżek, a następnie na zachłannym wyborze ścieżki o najmniejszym dolnym ograniczeniu i jej rozwinięcie. Kolejne kroki powtarzane są aż do uzyskania pełnej trasy. W procesie rozwijania ścieżek wykorzystywana jest metoda przeszukiwania, której działanie może być opisane jako zbliżone do przeszukiwania wszerz (BFS) lub przeszukiwania wszerz z pewnymi modyfikacjami. Istota algorytmu polega na wyborze i rozwijaniu węzłów o najmniejszym dolnym ograniczeniu kosztu.

Dolne ograniczenie jest obliczane poprzez redukcję macierzy kosztów. Wybór optymalnej trasy opiera się na minimalizacji dolnych ograniczeń, nie jest uwzględniane górne ograniczenie, ponieważ uważam je za zbędne w kontekście przeszukiwania w sposób zbliżony do BFS.

Złożoność czasowa:
Złożoność czasowa algorytmu zależna jest w dużym stopniu od instancji problemu. W skrajnych przypadkach może ona wynosić nawet n!(przypadek przejścia przez całe drzewo). Na ogół jednak algorytm ten działa znacznie szybciej niż brute force. Wynika to ze stosowania takich technik jak:

- Redukcja macierzy kosztów
- Ograniczanie drzewa przeszukiwań

3. Algorytm



Krok ten polega na wypełnieniu odpowiednich kolumn i wierszy INF, wykonaniu redukcji macierzy kosztów, obliczeniu kosztu i dodaniu wierzcholka do kolejki. Koszt liczony jest jako funkcja y = przebyta droga do tego momentu + droga z min do wierzchołka + koszt z redukcji. Kroki te powtarzamy dla kazdego z sąsiadów. Miałem problem z przełożeniem tego na schemat

4. Dane testowe

Do sprawdzenie poprawności działania algorytmu wybrano następujące dane testowe:

1. test_6_1.csv;
   http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/tsp_6_1.txt (dane te same tylko rozszerzenie i separator zmieniony)
2. test_10.csv;
   http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/tsp_10.txt (dane te same tylko rozszerzenie i separator zmieniony)

Do wykonania badań wybrano następujący zestaw instancj(dane o nazwie graph.csv są wygenerowane przeze mnie :

1. graph2.csv
2. graph3.csv
3. graph4.csv
4. graph5.csv
5. test_6_1.csv
6. graph7.csv
7. graph8.csv
8. graph9.csv
9. test_10.csv
10. tsp_12.csv
11. tsp_13.csv
12. tsp_14.csv
13. tsp_15.csv
14. tsp_18_1.csv
15. graph19.csv
16. graph20.csv
17. gr24.csv

5. Procedura badawcza

Należało zbadać zależność czasu rozwiązania problemu od wielkości instancji. W przypadku algorytmu realizującego BxB nie występowały parametry programu, które mogły mieć wpływ na czas i jakość uzyskanego wyniku. W związku z tym procedura badawcza polegała na uruchomieniu programu, załadowaniu pliku inicjalizującego i pomiaru czasu dla każdej instancji danych badawczych. Plik inicjalizujący:

[nazwa pliku][ilość pomiarów][minimalny koszt]

graph2.csv 100 187
graph3.csv 100 148
graph4.csv 100 139
graph5.csv 100 194
test_6_1.csv 100 132
graph7.csv 100 239
graph8.csv 100 266
graph9.csv 100 276
test_10.csv 100 212
tsp_12.csv 100 264
tsp_13.csv 100 269
tsp_14.csv 100 282
tsp_15.csv 100 291
tsp_18_1.csv 100 146
graph19.csv 100 169
graph20.csv 100 148
gr24.csv 1 1272

Każda instancja rozwiązywana była zgodnie z liczbą jej wykonań, np. graph2.csv wykonana została 100 razy. Do pliku wyjściowego output.csv zapisywany był czas wykonania algorytmu. Plik wyjściowy zapisywany był w formacie CSV. Wybrałem 100 pomiarów, gdyż uważam, że jest to wystarczająco duża liczba, aby otrzymać średni wynik zgodny z rzeczywistością.

Maksymalną instancją, na której sprawdzane było działanie algorytmu jest gr24.csv. Dla tej instancji czas wykonywania algorytmu przekroczył 1 godzinę. Przez tak długi czas odrzuciłem ją z analizy.

Pomiar czasu został wykonany prz użyciu biblioteki timeit:

```
  for i in range(int(instance[1])):
            timer   =   timeit.timeit(stmt='BxB(matrix)',
globals=globals(), number=1)
      times.append(timer)
  all_times.append(times)
```

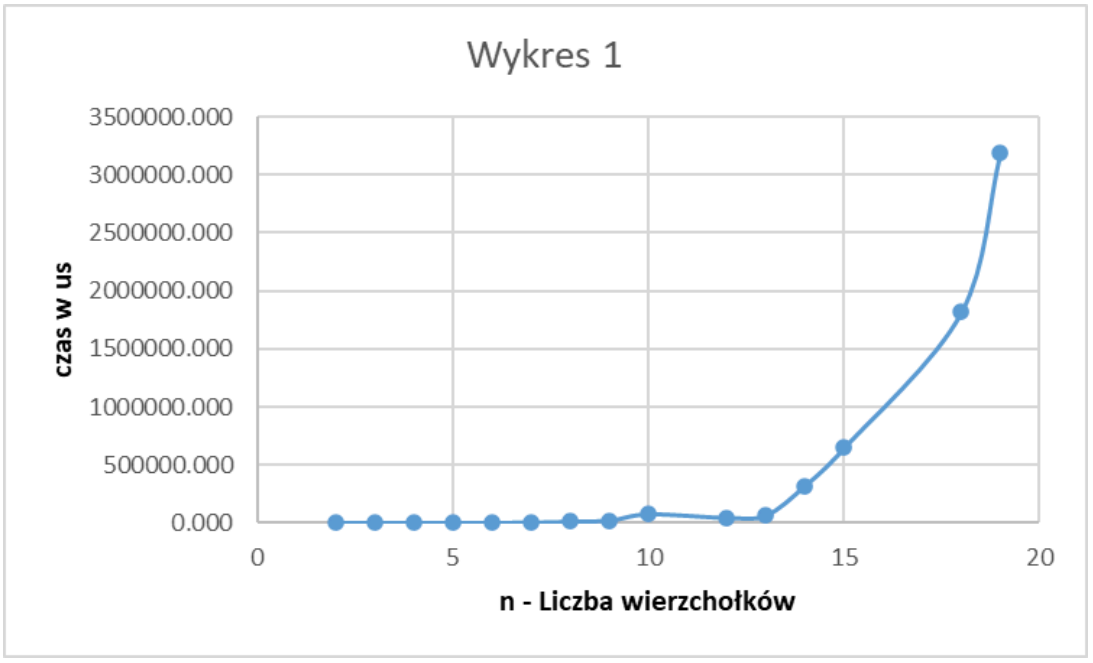
6. Wyniki

Wyniki zgromadzone zostały w plikach: output.csv,

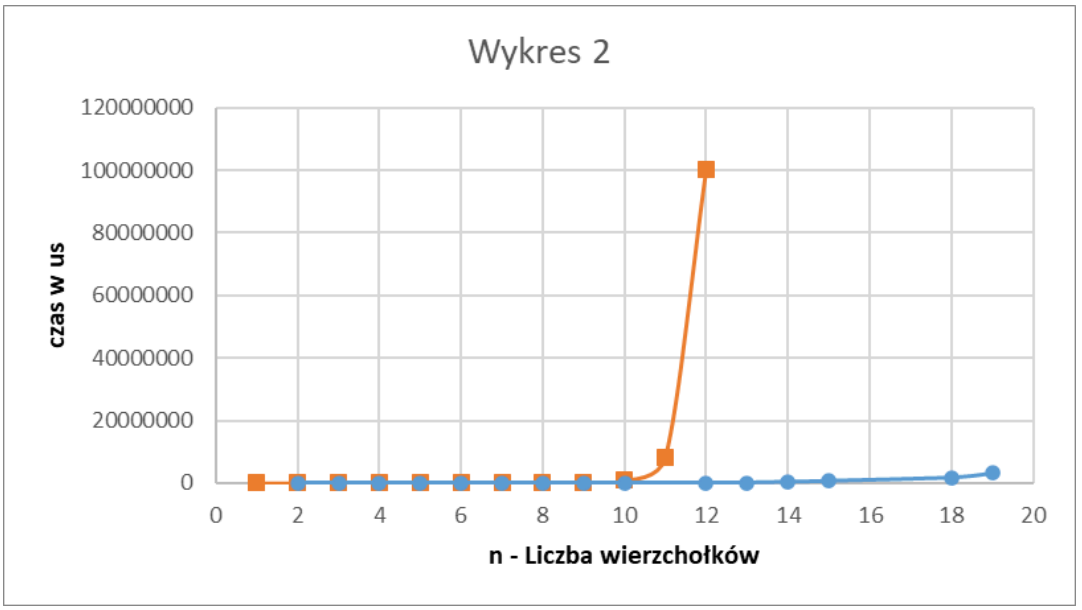
Tabela:

Wierzchołki	czas w ns	koszt
2	207.573	187
3	558.772	148
4	1192.312	139
5	2371.233	194
6	2985.984	132
7	4365.152	239
8	12480.617	266
9	22001.485	276
10	76804.817	212
12	45658.982	264
13	65876.537	269
14	321214.258	282
15	646412.485	291
18	1815226.059	146
19	3188456.142	169
20	746747.994	148

Wykres 1(złożoność czasowa BxB):



Wykres 2(złożoność czasowa BxB i BF porównanie):



7. Analiza wyników i wniosków

Wykres 1 stworzony jest na podstawie tabeli z wyłączeniem instancji o 20 wierzchołkach, ponieważ zakłócałaby ona jego płynność.

Jak widać na Wykresie 2 po krzywych wzrostu czasu(niebieska-BxB, pomarańczowa-BF) algorytm BxB ma znacząco niższe czasy rozwiązywania problemu TSP w stosunku do BF. Nie zawsze jest jednak tak dobrze. W przypadku, niektórych instancji grafów algorytm BxB musi “przejść przez całe drzewo” co sprowadza się do przeglądu zupełnego. Czas działania tego algorytmu nie zależy jedynie od ilości wierzchołków lecz także od połączeń między nimi. Widać to w tabeli gdzie czas dla instancji z 20 wierzchołkami jest ponad 4 razy krótszy od instancji z 19. Podsumowując algorytm Breach and Bound jest efektywniejszy od Brute Force, ale wciąż nadaje się jedynie do rozwiązywania małych instancji problemu komiwojażera. Złożoność czasowa opracowanego algorytmu wynosi w najbardziej pesymistycznym wariancie O(n!)