PEA

Pierwszy projekt z Projektowania Efektywnych Algorytmów

Implementacja i analiza efektywności algorytmu B&B oraz programowania dynamicznego

Rafał Rodak 252788

Politechnika Wrocławska

Prowadzący: Dr inż. Jarosław Mierzwa
Termin: pt. 11:15

2 listopada 2021
LaTeX

Spis treści

1 Wstęp

1.1 Opis zadania

Zadanie polega na zaimplementowaniu oraz dokonaniu analizy efektywności algorytmu przeglądu zupełnego, podziału i ograniczeń (B&B) oraz programowania dynamicznego (DP) dla asymetrycznego problemu komiwojażera (ATSP).

Problem komiwojażera [1] jest to zagadnienie optymalizacyjne polegające na znalezieniu minimalnego cyklu Hamiltona [2] w pełnym grafie ważonym (pełny graf ważony to taki, w którym wszystkie krawędzi są ze sobą połączone i posiadają etykiety liczbowe przy krawędziach, które wyrażają wagę). Cykl Hamiltona jest to taki cykl w grafie, w którym każdy wierzchołek grafu został odwiedzony dokładnie raz w wyłączeniem pierwszego wierzchołka.

Dodatkowo do projektu utworzone zostało repozytorium na platformie GitHub zawierające wszystkie pliki źródłowe wraz z komentarzami, sprawozdanie oraz pliki przeznaczone do testowania napisanych algorytmów, https://github.com/rodakrafal/PEA1

1.2 Założenia

Podczas realizacji zadania należy przyjąć następujące założenia:

- używane struktury danych są alokowane dynamicznie (w zależności od aktualnego rozmiaru problemu),
- program umożliwia weryfikację poprawności działania algorytmu. W tym celu istnieć możliwość wczytania danych wejściowych z pliku tekstowego,
- po zaimplementowaniu i sprawdzeniu poprawności działania algorytmu dokonano pomiaru czasu jego działania w zależności od rozmiaru problemu N,
- dla każdej wartości N należy wygenerowano po co najmniej 100 losowych instancji problemu (w sprawozdaniu należy umieścić tylko wyniki uśrednione – pamiętać, aby nie mierzyć czasu generowania instancji),
- implementacja algorytmów została dokonana zgodnie z obiektowym paradygmatem programowania,
- program napisany został w wersji okienkowej, obiektowo,
- kod źródłowy powinien jest komentowany,
- algorytmy napisanę są w języku C++,
- testy stworzonych algorytmów przeprowadzone zostały na wersji RELEASE,
- istnieje możliwość ostatnio wczytanych oraz wygenerowanych danych oraz wygenerowania.

[1] https://en.wikipedia.org/wiki/Travelling_salesman_problem
[2] https://en.wikipedia.org/wiki/Hamiltonian_path

2 Teoria

2.1 Brute Force

Metoda przeglądu zupełnego [3], tzw. przeszukiwanie wyczerpujące (eng. Exhaustive serach) bądź metoda siłowa (eng. Brute force), polega na znalezieniu wszystkich potencjalnych rozwiązań problemu oraz ich analizie w celu wybrania tego, które spełnia warunki zadania. Metodę tę stosuje się do rozwiązywania problemów, dla których znalezienie rozwiązania za pomocą innych dokładnych metod jest niemożliwe lub zbyt trudne. Główną wadą metody Brute Force jest to, że w przypadku wielu rzeczywistych problemów liczba naturalnych kandydatów jest zbyt duża.

Algorithm 2.1: BRUTE FORCE(P, c)

$c \leftarrow first(P)$
while $c \neq \wedge$
do $\begin{cases} \textbf{if } valid(P, c) & (1) \\ \quad \textbf{then } output(P, c) & (2) \\ c \leftarrow next(P, c) \end{cases}$

Brute Force oblicza i porównuje wszystkie możliwe premutacje tras w celu określenia najkrótszej ścieżki. Aby rozwiązać problem TSP przy użyciu tej metody, należy obliczyć całkowitą liczbę tras, a następnie wyznaczyć długości każdej z nich wybierając za każdym razem najkrótszą. Złożoność obliczeniowa algorytmu w przypadku problemu TSP przyjmuje postać wykładniczą $O(n!)$.

Przykład działania algorytmu można opisać w oparciu podany powyżej pseudocode. W pętli while obliczane będą kolejne permutacje ścieżki (1), aż do momentu wyczerpania się możliwych połączeń. W przypadku znalezienia krótszej ścieżki będzie ona zapamiętana (2).

2.2 Branch-N-Bound

2.3 Dynamic Programing

Metoda ta określa ogólne podejście polegające na przekształceniu zadania optymalizacyjnego w wieloetapowy proces decyzyjny, w którym stan na każdym etapie zależy od decyzji wybieranej ze zbioru decyzji dopuszczalnych.

[3] https://en.wikipedia.org/wiki/Bruteforce_search

Algorithm 2.2: DYNAMIC PROGRAMING(G, n)

for $k := 2$ **to** n
  do $\{C(\{k\}, k) := d_{1,k}$
for $s := 2$ **to** $n - 1$
  do $\begin{cases} \textbf{for } \forall\ S \in \{2, ..., n\}, |S| = s \\ \quad \textbf{do } \begin{cases} \textbf{for } \forall\ k \in S \\ \quad \textbf{do } C(S, k) := min_{m \neq k, m \in S} \end{cases} \end{cases}$

$opt := min_{k \neq 1}[C(\{2, 3, ..., n\}, k) + d_{k,1}]$
return (opt)

Metoda rozwiązywania problemów optymalizacyjnych dekomponowalnych rekurencyjnie na podproblemy.

1. Scharakteryzowanie struktury rozwiązania optymalnego
2. Zdefiniowanie kosztu rozwiązania optymalnego jako funkcji optymalnych rozwiązań podproblemów (układ równań rekurencyjnych)
3. Wyliczenie optymalnego kosztu metodą bottom-up (rozwiązywanie podproblemów od najmniejszego na największego)
4. Skonstruowanie rozwiązania optymalnego (na podstawie wykonanych obliczeń) - można pominąć ten krok jeżeli interesuje nas tylko koszt rozwiązania optymalnego a nie jego osiągnięcie

Algorytm Helda - Karpa [4], zwany również algorytmem Helda - Bellmana - Karpa jest algorytmem służący do rozwiązywania problemu TSP. Jest on algorytmem dokładnym opartym na programowaniu dynamicznym. Ma on złożoność czasową $O(n^2 2n)$ oraz złożoność pamięciową równą $O(nn^2)$. Jest to co prawda złożoność gorsza od wielomianowej, ale algorytm ten jest znacznie lepszy od algorytmu naiwnego sprawdzającego wszystkie warianty takiego jak Brute Force.

Działanie algorytmu: załóżmy, że mamy graf liczący n wierzchołków ponumerowanych 1, 2, . . . , n. Wierzchołek 1 niech będzie wierzchołkiem początkowym. Jako $d_{i,j}$ oznaczmy odległość między wierzchołkami i oraz j (są to dane wejściowe). Oznaczmy jako D(S, p) optymalną długość ścieżki wychodzącej z punktu 1 i przechodzącej przez wszystkie punkty zbioru S tak, aby zakończyć się w punkcie p (p musi należeć do S). Przykładowo, zapis D(2, 5, 6, 5) to optymalna długość ścieżki wychodzącej z punktu 1, przechodzącej przez punkty 2 i 6, kończącej się w punkcie 5. Liczbę punktów w zbiorze S oznaczmy jako s. Tym co odróżnia metrodę programowania dynamicznego od metody przeglądu zupełnego jest to, że nie musimy wyliczać odległości poszczególnych wierzchołków, a jedynie skorzystać z wcześniej policzonych wyników. W przeglądzie zupełnym każdą drogę liczymy od nowa, nie wykorzystując wcześniejszych wyliczeń, co jest tratą czasu, której unikamy w metodzie programowania dynamicznego.

3 Pomiary

Do sprawdzenia poprawności działania algorytmu, wybrano następujący zostaw instancji:

∗ http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pea-stud/tsp/

[4] https://en.wikipedia.org/wiki/Held%E2%80%93Karp_algorithm