Karol Wojtachnia 243702

Projektowanie efektywnych algorytmów

Sprawozdanie z etapu 1:

Algorytmy przeglądu zupełnego oraz podziału i ograniczeń dla problemu komiwojażera

Prowadzący:
dr inż. Jarosław Rudy

Cel zadania

Zadanie polegało na implementacji zadanych algorytmów oraz zbadanie czasu wykonania algorytmów i pokazanie wpływu najważniejszych parametrów na szybkość ich działania.

Wstęp teoretyczny

Problem komiwojażera (ang. travelling salesman problem – TSP) polega na znalezieniu w grafie ważonym najkrótszej ścieżki łączącej wszystkie wierzchołki. Przez każdy wierzchołek można przejść tylko raz.

Algorytm przeglądu zupełnego (ang. brute force – BF) wyznacza wszystkie możliwe ścieżki w grafie i zwraca najkrótszą z nich. Dzięki temu zawsze dostajemy poprawne rozwiązanie, ale złożoność algorytmu rośnie bardzo szybko wraz ze wzrostem wielkości grafu. Ilość możliwych ścieżek w pełnym grafie to n!, gdzie n jest ilością węzłów. Dla TSP algorytm brute force musi przejrzeć (n-1)! możliwości.

Algorytm podziału i ograniczeń (ang. branch and bound – B&B) pozwala ograniczyć ilość przeglądanych rozwiązań. Na jego potrzeby definiujemy drzewo przestrzeni stanów, które w korzeniu zawiera tylko wierzchołek początkowy tworzonego cyklu Hamiltona. Na drugim poziomie znajdują się węzły zawierające dwa wierzchołki – dodajemy do ścieżki każdy wierzchołek poza pierwszym. Na kolejnych poziomach tworzy się dla węzłów dzieci, które zawierają o jeden więcej odwiedzony wierzchołek. Rozwijając takie drzewo dla każdego węzła należy podjąć decyzję, czy jego rozwinięcie będzie opłacalne. Tworząc drzewo trzeba przyjąć jakąś formę wyliczania ograniczenia - w przypadku TSP należy sprawdzić, czy wchodząc do kolejnego węzła jesteśmy w stanie otrzymać ścieżkę krótszą niż najlepsza dotychczasowo znaleziona. Jeżeli nie ma możliwości znalezienia ścieżki krótszej takiego węzła się nie rozwija i przechodzi do następnego. Liście na dole drzewa reprezentują ścieżki Hamiltona w badanym grafie.
Złożoność tego algorytmu zależy od badanego grafu oraz przyjętego sposobu obliczania ograniczenia. Czas wykonania algorytmu może być różny podczas badania instancji o takich samych rozmiarach. W najbardziej niekorzystnych przypadkach złożoność czasowa może się zbliżać do algorytmu BF.

Szczegóły implementacji algorytmów

a) Algorytm przeglądu zupełnego

Zaimplementowany algorytm korzysta z następujących zmiennych:

- int tab[][] – tablica reprezentująca graf wczytana z pliku instancji
- size – ilość wierzchołków w grafie
- int path[] – tablica zawierająca najlepszą znalezioną ścieżkę
- int temp_path[] – tablica zawierająca aktualnie badaną ścieżkę
- bool visited[] – tablica wartości logicznych do zapisywania, czy wierzchołek został odwiedzony
- int temp_path_counter – licznik odwiedzonych wierzchołków
- int path_length – długość najlepszej znalezionej ścieżki
- int temp_path_length – dotychczasowa długość aktualnie badanej ścieżki

Pseudokod funkcji bruteForce(int start)

'start' jest numerem wierzchołka dla którego funkcja jest wywoływana:

```
K1)   dodaj do temp_path wierzchołek o numerze start
K2)    jeżeli temp_path_counter=size to skocz do K12
K3)   oznacz wierzchołek start jako odwiedzony w tablicy visited[]
K4)    zwiększ temp_path_counter o 1
K4)    for (int i =size;i--; )
K5)          Jeżeli wierzchołek o numerze i nie został odwiedzony:
K6)                dodaj tab[start][i] do temp_path_length
K7)                wywołaj bruteForce(i)
K8)                odejmij tab[start][i] od temp_path_length
K9)   oznacz wierzchołek start jako nieodwiedzony w tablicy visited[]
K10)  zmniejsz temp_path_counter o 1
K11)   return
K12)  dodaj tab[start][0] do temp_path_length
K13)   jeżeli temp_path_length < path_length:
K14)               przypisz temp_path_length do path_length
K15)               skopiuj zawartość temp_path[] do path[]
K16)  odejmij tab[start][0] od temp_path_length
K17)  zmniejsz temp_path_counter o 1
```

b) Algorytm podziału i ograniczeń

Sposób wyliczania ograniczenia jest oparty na następujących założeniach:

Do każdego wierzchołka wchodzimy raz i wychodzimy raz. Dla każdego wierzchołka wyznaczymy najkrótszą ścieżkę, która do niego wchodzi oraz najkrótszą ścieżkę, która z niego wychodzi. Sumujemy wszystkie wyznaczone ścieżki. Na koniec dzielimy otrzymaną sumę przez 2, ponieważ każda z krawędzi została policzona dwa razy. Obliczona w ten sposób wartość musi być mniejsza lub równa wartości ścieżki będącej rozwiązaniem TSP.
Wartość ta jest wyliczana przed rozpoczęciem programu, a następnie odpowiednio redukowana przy przejściu do następnego wierzchołka.

Zaimplementowany algorytm korzysta z następujących zmiennych:

- int tab[][] – tablica reprezentująca graf wczytana z pliku instancji
- size – ilość wierzchołków w grafie
- int path[] – tablica zawierająca najlepszą znalezioną ścieżkę
- int temp_path[] – tablica zawierająca aktualnie badaną ścieżkę
- bool visited[] – tablica wartości logicznych do zapisywania, czy wierzchołek został odwiedzony
- int temp_path_counter – licznik odwiedzonych wierzchołków
- int path_length – długość najlepszej znalezionej ścieżki
- int temp_path_length – dotychczasowa długość aktualnie badanej ścieżki
- int temp_bound – wartość ograniczenia

Pseudokod funkcji branchAndBound(int start, int current_bound)

‘start’ jest numerem wierzchołka dla którego funkcja jest wywoływana
‘current_bound’ jest wartością ograniczenia dla wywołanego wierzchołka

K1) dodaj do temp_path wierzchołek o numerze start
K2) jeżeli temp_path_counter=size to skocz do K15
K3) oznacz wierzchołek start jako odwiedzony w tablicy visited[]
K4) zwiększ temp_path_counter o 1
K4) for (int i =0;i < size;i++)
K5) Jeżeli wierzchołek o numerze i nie został odwiedzony:
K6) dodaj tab[start][i] do temp_path_length
K7) zredukuj current_bound o połowę wartości najkrótszej ścieżki wychodzącej z wierzchołka start oraz o połowę wartości najkrótszej ścieżki wchodzącej do wierzchołka i

K8) jeżeli temp_path_length+current_bound < path_length:
K9) wywołaj branchAndBound(i,current_bound)
K10) przywróć currnet_bound do wartości sprzed K7
K11) odejmij tab[start][i] od temp_path_length
K12) oznacz wierzchołek start jako nieodwiedzony w tablicy visited[]
K13) zmniejsz temp_path_counter o 1
K14) return
K15) dodaj tab[start][0] do temp_path_length
K16) jeżeli temp_path_length < path_length:
K17) przypisz temp_path_length do path_length
K18) skopiuj zawartość temp_path[] do path[]
K19) odejmij tab[start][0] od temp_path_length
K20) zmniejsz temp_path_counter o 1

Opis badania algorytmów

Aby sprawdzić działanie algorytmów badanie zostało podzielone na dwa etapy:

1) Wpływ wielkości instancji na czas działania algorytmów
2) Działanie algorytmów na różnych instancjach o tej samej wielkości

Instancje użyte podczas etapu pierwszego:

- m6.atsp 6 wierzchołków
- m8.atsp 8 wierzchołków
- m9.atsp 9 wierzchołków
- m10.atsp 10 wierzchołków
- m11.atsp 11 wierzchołków
- m12.atsp 12 wierzchołków
- m13.atsp 13 wierzchołków
- m14.atsp 14 wierzchołków
- m15.atsp 15 wierzchołków

Instancje użyte podczas etapu drugiego:

- m14.atsp 14 wierzchołków
- burma14.tsp 14 wierzchołków
- m16.atsp 16 wierzchołków
- ulysses16.tsp 16 wierzchołków

Czas działania dla każdej instancji obliczony został jako średnia arytmetyczna z 10 pomiarów. Instancje m16.atsp oraz ulysses16.tsp zostały użyte tylko do działania algorytmu podziału i ograniczeń, gdyż dla algorytmu przeglądu zupełnego czas ich wykonania był zbyt długi.
Pomiary były wykonywane z dokładnością do 1 ms.

Wyniki badania zaimplementowanych algorytmów

1) Wpływ wielkości instancji na czas działania algorytmów

Dla każdej użytej instancji sprawdzono średni czas działania algorytmu BF oraz B&B.

Instancja	m6.atsp	m8.atsp	m9.atsp	m10.atsp	m11.atsp	m12.atsp	m13.atsp	m14.atsp	m15.atsp
czas dla BT [ms]	2	5	10	30	191	1815	21744	301410	4440582
wzrost czasu	-	2,50	2,00	3,00	6,37	9,50	11,98	13,86	14,73
czas dla BnB [ms]	3	13	22	30	31	30	80	125	4401
wzrost czasu	-	4,33	1,69	1,36	1,03	0,97	2,67	1,56	35,21

Wiersz ‘wzrost czasu’ pokazuje ile razy wzrósł czas wykonania w porównaniu z poprzednią instancją.

Dla algorytmu przeglądu zupełnego nie wszystkie wyniki pokrywają się z oczekiwanymi. Dla przykładu:
pomiędzy m6 oraz m8 czas powinien wzrosnąć 6*7=48 razy, a wzrósł 2,5 razy
pomiędzy m8 oraz m9 czas powinien wzrosnąć 8 razy, a wzrósł 2 razy
pomiędzy m10 oraz m11 czas powinien wzrosnąć 10 razy, a wzrósł 6,37 razy

Dopiero dla większych instancji wzrost czasu jest zbliżony do oczekiwanego. Np.:
pomiędzy m12 oraz m13 czas powinien wzrosnąć 12 razy i wzrósł 11,98 razy
pomiędzy m13 oraz m14 czas powinien wzrosnąć 13 razy i wzrósł 13,86 razy
pomiędzy m14 oraz m15 czas powinien wzrosnąć 14 razy i wzrósł 14,73 razy

Zgodnie z oczekiwaniami nie da się wyznaczyć żadnej zależności dla wzrostu czasu wykonania algorytmu podziału i ograniczeń. Otrzymane wyniki potwierdzają, że czas działania tego algorytmu jest inny dla każdej instancji problemu oraz nie zależy wprost od wielkości tej instancji.

Warto zauważyć, że czas wykonania algorytmu BF jest niższy dla małych instancji. Dopiero dla m10 czasy te się zrównują, a następnie używając B&B można uzyskać znacznie lepsze rezultaty.

2) Działanie algorytmów na różnych instancjach o tej samej wielkości

Instancja	m14.atsp	burma14.tsp	m16.atsp	ulysses16.tsp
czas dla BT [ms]	301410	313896	-	-
czas dla BnB [ms]	125	385	369078	89558

Otrzymane wyniki są zgodne z oczekiwanymi. Algorytm BF wykonał się w podobnym czasie dla m14 oraz burma14, co potwierdza, że czas jego działania jest ściśle związany z wielkością instancji. Czasy otrzymane dla algorytmu B&B ponownie potwierdzają, że czas jego wykonania nie jest ściśle związany z

wielkością instancji. Czas wykonania dla m14 i bruma14 różni się około 3 razy, a czas wykonania dla m16 oraz ulysses16 różni się o jeden rząd wielkości, co jest bardzo znaczącą różnicą.

Wnioski

Podczas realizacji tego etapu udało się zaimplementować dwa dokładne algorytmy. Oba działały dobrze i dawały poprawne wyniki. Nie udało się znaleźć dokładnych przyczyn znacznie wolniejszego, niż oczekiwany, wzrostu czasu działania algorytmu przeglądu zupełnego dla małych instancji. Przyczyną może być niedokładność pomiaru czasu wykonania. Uzyskane wyniki pozwalają twierdzić, że użycie algorytmy brute force jest wystarczające dla instancji poniżej 10 wierzchołków i zwraca rozwiązanie szybciej niż algorytm branch and bound. Jest to wynikiem obliczania ograniczenia, które znacząco wpływa na czas wykonania w przypadku małych instancji.