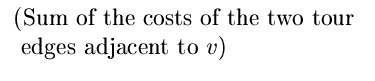
Prowadzący: dr inż. Dariusz Banasiak  
Termin: czwartek 9:15

Filip Mazur 226018

**Projektowanie Efektywnych Algorytmów – Projekt  
Problem komiwojażera – metoda podziału i ograniczeń  
Sprawozdanie**

1. **Wstęp teoretyczny**  
   Problem komiwojażera (Travelling Salesman Problem) [6.3] to zagadnienie optymalizacyjne, polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Problem można opisać bardziej kolokwialnie: komiwojażer chce odwiedzić wszystkie miasta, w każdym będąc tylko raz i przy okazji przebyć możliwie jak najkrótszą drogę.  
   Problem ten podzielić możemy na:   
   Symetryczny (STSP) – droga z miasta A do B jest taka sama jak z B do A.  
   Asymetryczny (ATSP) – drogi A-B oaz B-A mogą mieć różne koszty.  
     
   Metoda podziału i ograniczeń[6.4] służy do rozwiązywania problemów optymalizacyjnych. Jej działanie opiera się na analizie drzewa przestrzeni stanów. Drzewo to reprezentuje wszystkie możliwe ścieżki jakimi może pójść algorytm rozwiązując dany problem. Algorytm zaczyna w korzeniu drzewa i przechodząc do któregoś liścia konstruuje rozwiązanie. Przeglądanie całego drzewa byłoby bardzo kosztowne ze względu na jego wykładniczy rozmiar, dlatego metoda podziału i ograniczeń w każdym węźle oblicza granicę (w przypadku problemu minimalizacyjnego, jakim jest TSP jest to tzw. lower bound), która pozwala określić go jako obiecujący bądź nie. W dalszej fazie algorytm odcina węzły, która uzna za nieobiecujące i przegląda tylko pozostałe. W ten sposób możemy potencjalnie znacznie skrócić czas wykonywania programu względem metody Brute Force.  
     
   Zastosowany wzór na lower bound każdej podróży[6.1]:  
   $\displaystyle \sum_{v \in V}^{}$  
     
   Złożoność czasowa: w najgorszym wypadku (nie odetniemy żadnego węzła) złożoność czasowa metody podziału i ograniczeń będzie równa metodzie Brute Force, czyli 2n. W najlepszym przypadku przejdziemy przez wszystkie poziomy drzewa poszukiwań tylko raz (koszt równy n),   
   zaś resztę węzłów odetniemy.
2. **Przykład działania instancji metody** [6.5]   
   Travelling salesman Problem Example
   Row Minimum
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6 -
   ...Travelling salesman Problem Example
   Column Minimum
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6...  
   Branch and Bound-Step
   31
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6 -
   35
   X12
   X13
   X14
   X15
   For ...Branch and Bound-Step
   31
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6 -
   35 32
   X12
   X13
   X14
   X15
   F...Branch and Bound-Step
   31
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6 -
   35 32 34
   X12
   X13
   X14
   X1...Branch and Bound-Step
   31
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6 -
   35 32 34 31
   X12
   X13
   X14...Branch and Bound-Step
   31
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6 -
   35 32 34 31
   X12
   X13
   X14...Branch and Bound-Step
   31
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6 -
   35 32 34 31
   X12
   X13
   X14...Branch and Bound-Step
   31
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6 -
   35 32 34 31
   X12
   X13
   X14...Branch and Bound-Step
   31
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6 -
   35 32 34 31
   X12
   X13
   X14...Branch and Bound-Step
   31
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6 -
   35 32 34 31
   X12
   X13
   X14...Branch and Bound-Steps
   31
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6 -
   35 32 34 31
   X12
   X13
   X1...Branch and Bound-Steps
   31
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6 -
   35 32 34 31
   X12
   X13
   X1...Branch and Bound-Steps
   31
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6 -
   35 32 34 31
   X12
   X13
   X1...Branch and Bound-Steps
   31
   1 2 3 4 5
   1 - 10 8 9 7
   2 10 - 10 5 6
   3 8 10 - 8 9
   4 9 5 8 - 6
   5 7 6 9 6 -
   35 32 34 31
   X12
   X13
   X1...
3. **Implementacja**  
   Projekt składa się z dwóch klas:
   1. Stopwatch, korzysta z biblioteki chrono do pobierania czasu systemowego oraz wykonywania działań na zmiennych przechowujących czas systemowy. Dzięki tej klasie tworzę obiekt timer odpowiedzialny za pomiar czasu trwania algorytmu.
   2. tspBB, klasa główna służąca do wykonywania obliczeń. Dzięki tej klasie tworzę obiekt testUnit odpowiedzialny za testowanie i wykonywanie algorytmu.  
        
      Opis najistotniejszych użytych funkcji:
   3. int findFirstLowest(int adjacencyMatrix[noOfCities][noOfCities], int i) - zwraca najtańszą krawędź wierzchołka i, przyjmuje jako argumenty macierz sąsiedztwa i numer wierzchołka.
   4. int findSecondLowest(int adjacencyMatrix[noOfCities][noOfCities], int i) - znajduje drugą najtańszą krawędz wierzchołka i, argumenty j.w.
   5. void currToBest(int currentPath[]) - zastępuje najlepsze dotychczasowe rozwiązanie obecnym, przyjmuje jako argument obecne rozważaną ścieżkę.
   6. void recurrence(int adjacancyMatrix[noOfCities][noOfCities], int currentBound, int currentCost, int currentHeight, int currentPath[]) – funkcja rekurencji, przyjmuje jako argumenty macierz sąsiedztwa, lower bound dla korzenia, dotychczas obliczony koszt, poziom drzewa poszukiwań, w którym się obecnie znajdujemy oraz dotychczasową ścieżkę.
   7. void TSP(int adjacancyMatrix[noOfCities][noOfCities]) – główna funkcja programu, przyjmuje za argument macierz sąsiedztwa, inicjalizuje tablicę ścieżki oraz odwiedzonych wierzchołków, oblicza lower bound dla korzenia, wywołuje rekurencję.  
        
      Do badania czasu działania algorytmu wykorzystałem następujące funkcje:
   8. Stopwatch() – konstruktor obiektu, przy okazji ustawia zmienną point1, która przechowuje aktualny czas systemowy.
   9. countTimeDiff() – ustawia drugą zmienną point2 oraz odejmuje point1 od point2. Wynik podaje w nanosekundach.
4. **Badania**

Wykonano 50 pomiarów dla róznej ilości miast w wersji release programu. Wykorzystano dwa pliki STSP oraz dwa ATSP. Wszystkie pomiary znajdują się w pliku tsp\_bb.xlsx.   
Platforma testowa: procesor Intel i5-4460, 4 rdzenie – 4 wątki, taktowanie 3.20 GHz.  
  
README.md  
Test Files:  
STSP  
GR17 is a set of 17 cities, from TSPLIB. The minimal tour has length 2085.  
FRI26 is a set of 26 cities, from TSPLIB. The minimal tour has length 937.  
ATSP  
BR17 is a set of 17 cities, from TSPLIB. The minimal tour has length 58.  
FFTV15 is a set of 15 cities, from TSPLIB. The minimal tour has length 55.

Obliczona średnia **AVG = 4227541000 ns ~ 4,23 s**  
Obliczona średnia **AVG = 151263500000 ns ~ 151,13 s**  
Obliczona średnia **AVG = 705793560 ns ~ 0,71 s**Obliczona średnia **AVG = 144793 ns ~ 1,46E-04 s**

1. **Wnioski**  
   Przed przeprowadzeniem badań sądziłem, że metoda podziału i rozgałęzień będzie bardzo mało optymalna, ponieważ zależnie od instancji programu złożoność czasowa algorytmu może być bardzo niska lub bardzo wysoka, w najgorszym razie równa metodzie Brute Force. Spodziewałem się dużego rozrzutu wyników pomiaru czasu, jednak po wykonaniu testów okazało się, że byłem w błędzie. Większość wyników plasuje się bardzo blisko średniej, która również jest zaskakująco niska. Mniej optymalnych wyników jest zauważalnie mniej. Sądzę, że gdyby zrobić bardzo dużą ilość pomiarów rozrzut wyników przypominałby połowę krzywej Gaussa (im dalej od wartości średniej, tym mniej wyników). Nie jestem pewien dlaczego wyniki okazały się być tak uporządkowane. Metoda podziału i rozgałęzień okazała się być lepszą niż początkowo sądziłem. Czas trwania algorytmu mocno zależy nie tylko od ilości miast lecz również od rzędu wielkości krawędzi grafu oraz od typu (synchroniczny, asynchroniczny). Test dla 17 miast dla grafu asynchronicznego wypadł o rząd wielkości szybciej, niż dla synchronicznego. Krawędzie w pliku ATSP były o jeden rząd wielkości mniejsze niż w pliku TSP, zatem wnioskuję, że średnia długość krawędzi może mieć znaczny wpływ na czas wykonywania programu. Niestety nie udało mi się znaleźć pliku ATSP, który umożliwiłby mi jednoznaczne stwierdzenie czy typ (STSP/ATSP) grafu ma wpływ na czas wykonywania progamu, chociaż podejrzewam, że nie ma. Wszystkie wyniki dla wszystkich programów były zgodne z oczekiwaniami (plik README.md oraz tspBB.xlsx).
2. **Literatura**
   1. <http://lcm.csa.iisc.ernet.in/dsa/node187.html> - Optimal Solution for TSP using Branch and Bound.
   2. <https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html> - Data for the Traveling Salesperson Problem.
   3. <https://pl.wikipedia.org/wiki/Problem_komiwoja%C5%BCera> – Problem komiwojażera
   4. https://www.ii.uni.wroc.pl/~prz/2011lato/ah/opracowania/met\_podz\_ogr.opr.pdf - Metoda podziału i ograniczeń.
   5. <https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound> - Przykład instancji działania algorytmu.