Paweł Biel

nr indeksu 225949

Sprawozdanie

Problem komiwojażera metodą podziału i ograniczeń

1. Informacje teoretyczne

Problem klasy NP jest problemem, dla którego rozwiązanie można zweryfikować w czasie wielomianowym – natomiast samo znalezienie rozwiązania problemu prawie na pewno wymaga czasu ponadwielomianowego – nie znaleziono dotąd algorytmu o wielomianowej złożoności czasowej. Dla rozwiązania problemów klasy NP można zastosować przegląd zupełny – otrzymamy wtedy dokładne rozwiązanie, jednak ze względu na bardzo dużą złożoność czasową, nadaje się on jedynie dla bardzo małych rozmiarów problemu. Dla nieco większych instancji problemów również możemy otrzymać dokładny wynik stosując algorytm oparty na programowaniu dynamicznym lub metodzie podziału i ograniczeń (branch and bound). Do implementacji w ramach projektu wybrałem metodę podziału i ograniczeń dla problemu komiwojażera.

2. Problem komiwojażera

Zagadnienie optymalizacyjne, polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Często można się spotkać że problem jest dedykowany dla miast o odległość między tymi miastami to koszt podróży czy czasu, w grafie pełnym ten koszt odpowiada krawędziom. Problem komiwojażera można rozdzielić na dwa przypadki asymetryczny i symetryczny. Ten pierwszy polega na tym, że np. z miasta A do miasta B mamy 2 drogi jednostronne o różnych kosztach przejazdu (mogą być też takie same), natomiast symetryczny polega na tym że droga jest jedna w jedną stronę z jednego do drugiego miasta. Złożoność obliczeniowa:

- złożoność czasowa algorytmu wykorzystującego metodę podziału i ograniczeń jest bardzo mocno zależna od danych; w pesymistycznym przypadku wynosi O(n!), gdzie n – ilość miast, ale dla większości instancji problemu będzie w stanie znaleźć rozwiązanie szybciej[1]

3. Początkowe uwagi i problemy

Program wykonujący powyższy algorytm napisałem w języku Python, co pod koniec prac nad algorytmem nasuwa pierwszy wniosek, mianowicie mamy bardzo ograniczone możliwości kontroli nad pamięcią. Implementację algorytmu wzorowałem na paru dokumentach[2][3]. Niestety gdzieś popełniłem błąd przy blokowaniu cykli czy też drogi powrotnej i program nie wskazuje dobrych wyników – kosztów. Algorytm działa dla tych 3 macierzy, uważam że dla większej macierzy właśnie

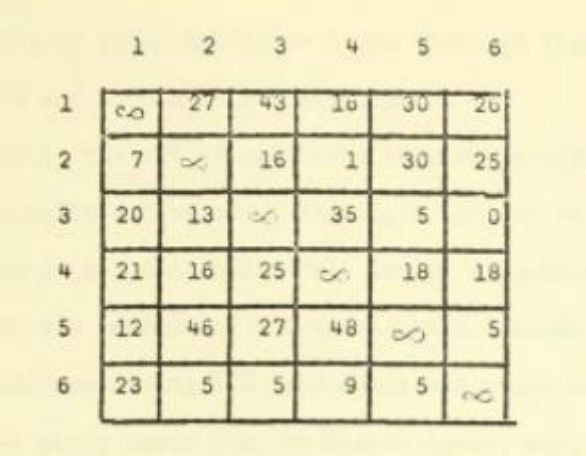
[1] http://www.geeksforgeeks.org/branch-bound-set-5-traveling-salesman-problem/
[2] https://archive.org/stream/algorithmfortrav00litt#page/n15/mode/2up
[3] https://www.ii.uni.wroc.pl/~prz/2011lato/ah/opracowania/met_podz_ogr.opr.pdf

blokowanie cykli gdzieś zawodzi i jest to jakiś szczegół (bardzo ważny) w kodzie, którego niestety na dzień dzisiejszy nie potrafię znaleźć, a poświęciłem ponad 3 dni nad debuger'em .

Macierz kosztów:

	1	2	3	4	5
1	∞	12	3	45	6
2	78	∞	90	21	3
3	5	56	∞	23	98
4	12	6	8	∞	34
5	3	98	3	2	∞

Macierz C	*1*	*2*	*3*	*4*
1	∞	2	7	3
2	7	∞	8	5
3	9	4	∞	6
4	3	8	5	∞



	1	2	3	4	5	6
1	∞	27	43	16	30	26
2	7	∞	16	1	30	25
3	20	13	∞	35	5	0
4	21	16	25	∞	18	18
5	12	46	27	48	∞	5
6	23	5	5	9	5	∞

4. Opis algorytmu

W moim programie zaimplementowałem algorytm oparty na metodzie podziału i ograniczeń opisany w pliku , w wersji stosującej strategię przeszukiwania typu najpierw najlepszy.

Lista kroków algorytmu:

- **a.** Znajdź minimalny element w każdy wierszu i go odejmij, następnie wykonaj to samo dla kolumn, suma tych elementów to jest lower bound czyli najlepsze rozwiązanie w danym stanie. Upper bound można wyliczyć np. z wzięcia po kolei wszystkich wierzchołków i z sumowania kosztów z 1 do 2, 2 do 3 etc. To też jest już jakieś ograniczenie górne.
- **b.** Następnie dla danej macierzy(wierzchołka, miasta) szukamy maksymalnego kosztu wyłączenia oraz dla tego kosztu wyłączenia rozgałęziamy nasze drzewo tworząc nową macierz z zablokowaną drogą i zwiększonym lower bound o ten koszt wyłączenia. Oznacza to że tego miasta o tym koszcie wyłączenia nie odwiedziliśmy.
- **c.** Natomiast druga odnoga to jest macierz z usuniętym wierszem i kolumną tegoż kosztu, potem następuje redukcja i ewentualny koszt redukcji jest dodawany do lower bound dla danej macierzy. Blokujemy drogę powrotną, aby nie wykonać ruchu z miasta 1 do miasta 2 i z miasta 2 do miasta 1.
- **d.** Dla macierzy która została przez nas „okrojona" wykonujemy na nowo punkt b i c, następnie blokujemy cykl o najdłuższej możliwej drodze [Rysunek 1].
- **e.** Następnie wykonujemy punkty b, c, d dla miasta o najniższym lower bound, wykonujemy algorytm tak długo aż dojedziemy do macierzy 2 x 2.

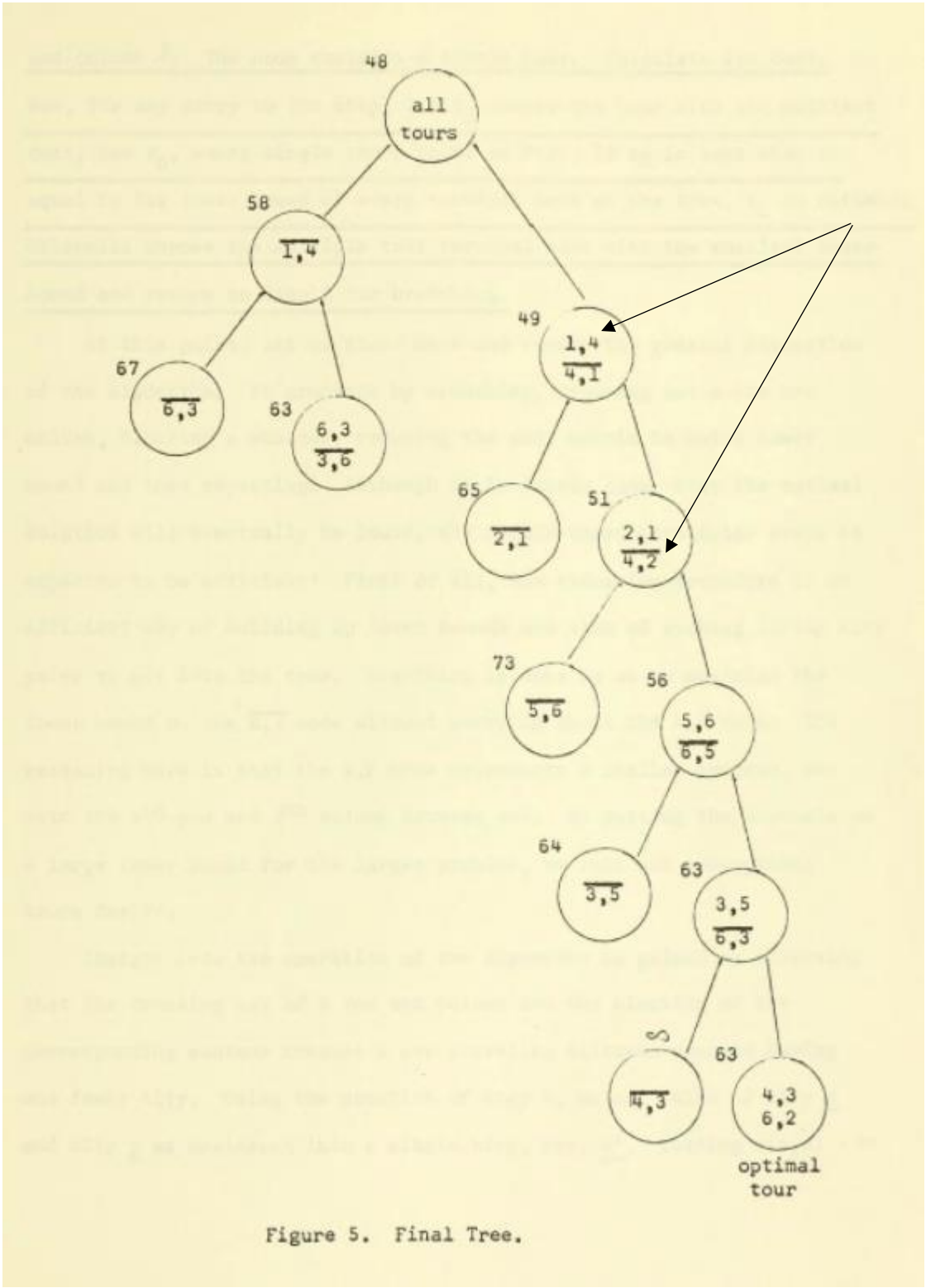


Figure 5. Final Tree.

Rysunek 1

5. Pomiary czasu

Zostały wykonane dla 20 prób, a wynik został uśredniony.

n = 4	0.002392s
n = 5	0.003147s
n = 6	0.003775s

Z racji iż wyniki i ich ilość nie jest zadowalająca postanowiłem też sięgnąć do innych źródeł skąd można coś wywnioskować.

Ilość miast	Przegląd zupełny [s]	Podział i ograniczenia [s]
3	0,00000623	0,00001183
6	0,50833200	0,02339070
8	0,01984770	0,17248000
11	4,20448000	0,03775410
12	52,5361000	–
14	–	0,02233150
18	–	0,44274300
22	–	11,0570000

6. Wnioski

Czas wykonania algorytmu przeglądu zupełnego jest stosunkowo duży nawet dla niewielkich rozmiarów problemu, natomiast przy określonym rozmiarze dla każdej instancji problemu jest stały. Czas wykonania algorytmu opartego na metodzie podziału i ograniczeń dla tych samych rozmiarów problemu jest wyraźnie mniejszy, ale bardzo mocno zależy od konkretnej instancji problemu – im większy rozmiar, tym więcej instancji potrzebuje na wykonanie algorytmu czasu dużo dłuższego od średniej, ale jednocześnie zwykle dużo mniejszego od czasu potrzebnego na wykonanie przeglądu zupełnego. Pewne nieścisłości (dla 8 miast średni czas wykonywania przeglądu zupełnego jest niższy niż sąsiednie, a metody podziału i ograniczeń – wyższy) wynikają prawdopodobnie ze specyfiki środowiska pomiarowego: praca pod kontrolą systemu operacyjnego, a więc możliwy wpływ działania innych uruchomionych w systemie procesów na wyniki pomiarów).