**Grzegorz Zając 200664** Data oddania sprawozdania: 7.11.2014

**Wojciech Biniek 200632** Termin zajęć: Poniedziałek TN 11:15

Projektowanie efektywnych algorytmów

Temat ćwiczenia:

***Metoda podziału i ograniczeń***

***dla problemu komiwojażera (TSP)***

Prowadzący:

**dr inż. Dariusz Banasiak**

1. **Wstęp teoretyczny**
   1. **Opis problemu**

**Problem komiwojażera** to zagadnienie optymalizacyjne, polegające na znalezieniu minimalnego cyklu Hamiltona w pełnym grafie ważonym. Mówiąc prościej, rozwiązaniem problemu komiwojażera jest znalezienie najkrótszej ścieżki w grafie skierowanym, rozpoczynającej się w danym wierzchołku, odwiedzającej wszystkie wierzchołki dokładnie raz i kończącej się w wierzchołku początkowym. Ścieżka taka nazywa się optymalną trasą. Ponieważ nie ma znaczenia, gdzie zaczynamy, wierzchołek początkowy może być po prostu pierwszym wierzchołkiem w grafie.

* 1. **Opis algorytmu**

**Metoda podziału i ograniczeń** 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 (ograniczenie), która pozwala określić go jako obiecujący bądź nie. W dalszej fazie algorytm przegląda tylko potomków węzłów obiecujących. Pozwala to, razem z dobraniem odpowiedniej strategii odwiedzania wierzchołków oraz liczenia granicy, zmniejszyć ilość odwiedzonych wierzchołków i szybciej znaleźć rozwiązanie problemu.

**Bardziej szczegółowy opis algorytmu znajduje się w punkcie 2.1**

* 1. **Pseudokod**

Przeszukiwanie typu najpierw najlepszy z metodą podziału i ograniczeń:

initialize(Q) // inicjalizacja kolejki priorytetowej  
v ← korzeń drzewa // pierwszy w. będzie w. startowym  
best ← value(v)  
best items ← items(v)  
insert(Q, v) //umieszczamy w. pocz. w kolejce pryj.  
**while** !empty(Q) **do** //tak długo jak kolejka ma elementy  
 v ← remove(Q) //usuń wierzchołek z najlepsza granica  
 **if** bound(v) jest lepszy od best **then**  
 **for all** dziecko u węzła v **do** //ustaw u na potomka v **if** value(u) jest lepsza od best **then** //jeżeli wartość dla u jest najlepsza  
 best ← value(u) //zapamiętujemy ją i u  
 best items ← items(u)  
 **end if**  
 **if** bound(u) jest lepszy od best **then** //jeżeli ograniczenie u  
 insert(Q, u) //umieszczamy u w kolejce pryjorytetowej  
 **end if  
 end for  
 end if  
end while**

* 1. **Złożoność obliczeniowa algorytmu**

• Czas działania metody **bound** wynosi

• Czas działania metody **boundBranch** wynosi

• Zapamiętanie i odtworzenie macierzy to operacja rzędu

• Zapamiętywanie nowego najlepszego rozwiązania w czasie

• Oznacza to, że realizacja algorytmu w obrębie jednego węzła wymaga czasu

• Złożoność całego algorytmu można oszacować zatem przez ,   
gdzie ***f(n)*** jest liczbą węzłów drzewa poszukiwań odwiedzanych przez algorytm

• Liczba wykonanych obliczeń zależy od konkretnych danych wejściowych i w pesymistycznym przypadku jest wykładnicza

1. **Przykład praktyczny**
   1. **Opis działania algorytmu „krok po kroku” dla przykładowej instancji problemu komiwojażera**

Reprezentacja macierzy sąsiedztwa grafu, na podstawie którego zostanie krok po kroku zaprezentowane działanie algorytmu B&B zaimplementowanego w programie stanowiącym realizację ćwiczenia:

Aby zastosować przeszukiwanie typu najpierw najlepszy musimy być w stanie określić granicę dla każdego wierzchołka. Musimy określić dolną granicę długości tras uzyskiwanych przez rozwinięcie bieżącego węzła. Będziemy nazywać węzeł obiecującym, jeżeli jego granica będzie mniejsza niż bieżące znalezione minimum.

**Uzyskiwanie granicy:**

W każdej z tras długość krawędzi musi być co najmniej tak długa, jak najdłuższa krawędź wychodząca z węzła. Dlatego dolna granica wartości długości wybranej krawędzi dla wierzchołka **v1 jest określana przez minimum z wszystkich niezerowych pozycji wiersza 1**. w tablicy sąsiedztwa. Dolna granica kosztu wyjścia z wierzchołka v2 jest określana przez minimum niezerowych pozycji wiersza 2 i tak dalej. Dolna granica kosztu wyjścia z pięciu wierzchołków analizowanego w przykładnie grafu jest następująca:

Ponieważ trasa musi przechodzić przez węzeł dokładnie raz , dolna granica długości trasu jest sumą tych minimów. Dlatego dolna granica długości cyklu wynosi:

Nie można powiedzieć, że istnieje trasa o tej długości, ale jesteśmy pewni, że nie istnieje trasa o mniejszej długości.

Załóżmy, że dotarliśmy do węzła 2 pokonując trasę: 1→2. Wtedy zatwierdziliśmy wierzchołek 2 jako drugi odwiedzany wierzchołek i koszt dojścia do v2 jest waga krawędzi z v1 do v2 i wynosi 14. Dowolna z tras uzyskanych przez rozwinięcie tego węzła posiada dolną granicę kosztu:

Aby uzyskac minimum dla v2, nie dołączamy krawędzi do v1, ponieważ z v2 nie można wrócić do v1. Aby uzyskać minima dla pozostałych wierzchołków, nie dołączamy krawędzi do v2, ponieważ już jesteśmy w wierzchołku v2. Dolna granica długości dowolnej trasy, uzyskana przez rozwinięcie jest sumą tych minimów:

Aby jeszcze dokładniej zobrazować algorytm załóżmy, że dotarliśmy do węzła 3 pokonując trasę: 1→2→3. Zatwierdziliśmy już v2 jako drugi wierzchołek oraz v3 jako trzeci wierzchołek. Dowolna trasa otrzymana przez rozwinięcie tego wezła ma poniższą dolną granicę kosztu:

Aby uzyskać minima dla v4 i v5 , nie bierzemy pod uwagę krawędzi do v2 oraz v3 ponieważ już odwiedzliśmy te wierzchołki. Dolna grainca długości dowolnej trasy uzyskanej przez rozwinięcie węzła wynosi:

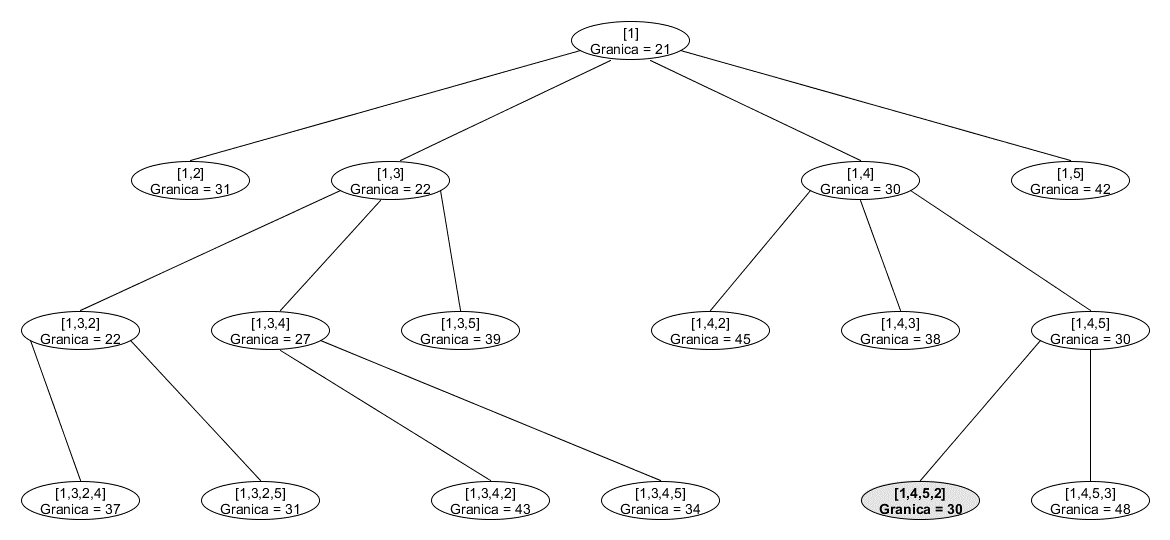
W analogiczny sposób możemy znaleźć dolną granicę długości trasy, jaka może być uzyskana przez rozwinięcie dowolnego węzła w drzewie.

Obliczane w ten sposób granice dolne są wykorzystywane w przeszukiwaniu najpierw najlepszy.

**Przeszukiwanie najpierw najlepszy:**

Działanie przeszukiwania najpierw najlepszy 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 (ograniczenie), która pozwala określić go jako obiecujący bądź nie. W dalszej fazie algorytm przegląda tylko potomków węzłów obiecujących. Pozwala to, razem z dobraniem odpowiedniej strategii odwiedzania wierzchołków oraz liczenia granicy, zmniejszyć ilość odwiedzonych wierzchołków i szybciej znaleźć rozwiązanie problemu.

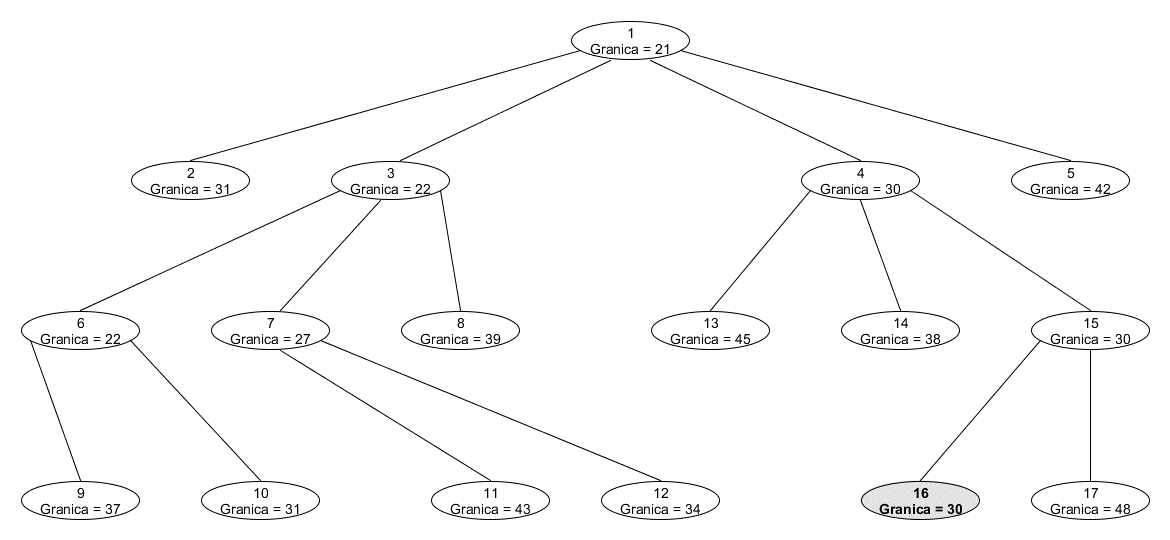
Granice dla poszczególnych węzłów w drzewie przeszukiwań są obliczane w sposób zaprezentowany powyżej.



*Rys. 1   
Rysunek przedstawiający drzewo przestrzeni stanów dla analizowanej instancji problemu. Obliczono granice dla poszczególnych węzłów. Zapis [a,b,c] oznacza, że zatwierdziliśmy trasę przez kolejne wierzchołki:* ***a→b→c*** *i poszukujemy kolejnego wierzchołka grafu do którego przejdziemy. Węzeł z optymalnym rozwiązaniem został zaznaczony.*

**Wykonując przeszukiwanie najpierw najlepszy wykonywane są następujące operacje:**

1. Przechodzimy do węzła [1] (korzenia)
2. Obliczamy granicę, jej wartość to 21
3. Ustawiamy minlength = ∞
4. Przechodzimy do węzła [1,2]
   1. Obliczamy granicę, jej wartość to 31
5. Przechodzimy do węzła [1,3]
   1. Obliczamy granicę, jej wartość to 22
6. Przechodzimy do węzła [1,4]
   1. Obliczamy granicę, jej wartość to 30
7. Przechodzimy do węzła [1,5]
   1. Obliczamy granicę, jej wartość to 42
8. Szukamy obiecującego nierozwiniętego węzła o najmniejszej granicy
   1. Jest to węzeł [1,3]. Przechodzimy do jego potomków
9. Przechodzimy do węzła [1,3,2]
   1. Obliczamy granicę, jej wartość to 22
10. Przechodzimy do węzła [1,3,4]
    1. Obliczamy granicę, jej wartość to 27
11. Przechodzimy do węzła [1,3,5]
    1. Obliczamy granicę, jej wartość to 39
12. Szukamy obiecującego nierozwiniętego węzła o najmniejszej granicy
    1. Jest to węzeł [1,3,2]. Przechodzimy do jego potomków
13. Przechodzimy do węzła [1,3,2,4]
    1. Ponieważ jest to liść obliczamy długośc trasy, wynosi ona 37
    2. Ponieważ wartość 37 < ∞, czyli wartoścminlength, ustawamy minlength = 37
    3. Węzły [1,5] oraz [1,3,5] stały się niobiecujące, ponieważ ich granice równe 42 i 39 są >= 37, czyli nowej wartości minlength
14. Przechodzimy do węzła [1,3,2,5]
    1. Ponieważ jest to liść obliczmy długość trasy , wynosi ona 31
15. Ponieważ 31 < 37, czyli wartości minlength, ustawiamy minlength = 31
16. Węzeł [1,2] staje się nieobiecujący, ponieważ jego dolna granica = 31 jest >= nowej wartości minlength
17. Szukamy obiecującego nierozwiniętego węzła o najmniejszej granicy
    1. Jest to węzeł [1,3,4]. Przechodzimy do jego potomków
18. Przechodzimy do węzła [1,3,4,2]
    1. Ponieważ jest to liść obliczmy długość trasy , wynosi ona 43
19. Przechodzimy do węzła [1,3,4,5]
    1. Ponieważ jest to liść obliczmy długość trasy , wynosi ona 34
20. Szukamy obiecującego nierozwiniętego węzła o najmniejszej granicy
    1. Jedynym nierozwiniętym obiecującym węzłem jest [1,4]. Przechodzimy do jego potomków.
21. Przechodzimy do węzła [1,4,2]
    1. Obliczmy granicę, jej wartość to 45
    2. Węzeł jest nieobiecujący ponieważ jego granica 45 >= 31, czyli aktualnej wartości minlength
22. Przechodzimy do węzła [1,4,3]
    1. Obliczmy granicę, jej wartość to 38
    2. Węzeł jest nieobiecujący ponieważ jego granica 38 >= 31, czyli aktualnej wartości minlength
23. Przechodzimy do węzła [1,4,5]
    1. Obliczmy granicę, jej wartość to 30
24. Szukamy obiecującego nierozwiniętego węzła o najmniejszej granicy
    1. Jedynym nierozwiniętym obiecującym węzłem jest [1,4,5]. Przechodzimy do jego potomków.
25. Przechodzimy do węzła [1,4,5,2]
    1. Ponieważ jest to liść obliczamy długośc trasy, wynosi ona 30
    2. Ponieważ wartość 30 < 31, czyli aktualna wartość minlength, ustawamy minlength = 30
26. Przechodzimy do węzła [1,4,5,3]
    1. Ponieważ jest to liść obliczamy długośc trasy, wynosi ona 48
27. Szukamy obiecującego nierozwiniętego węzła o najmniejszej granicy
    1. Nie ma więcej obiecujących nierozwiniętych węzłów. Kończymy pracę.

Określiliśmy, że węzeł [1,4,5,2], który reprezentuje trasę *1→4→5→2→3→1,* reprezentuje **trasę optymalną, której długość wynosi 30.** Kolejność odwiedzania węzłów drzewa przestrzeni stanów ilustruje poniższy diagram:

*Rys. 2   
Kolejność odwiedzania węzłów drzewa przestrzeni stanów.*

W drzewie przestrzeni stanów przedstawionym na rysunku znajduje się **17** węzłów, natomiast liczba węzłów w drzewie przeglądu zupełnego wyniosłaby 1+4+4\*3+4\*3\*2 = **41**.

1. **Implementacja**

Działanie funkcji ograniczeń zostało bardzo dokładnie opisane powyżej, dlatego w tej części skupiamy się jedynie na wykorzystanych strukturach danych.

* 1. **Wykorzystane struktury istotne dla działania algorytmu**

Podczas implementowania algorytmu wykorzystano nastepujące struktury danych:

1. **Wektor** dostępny w bibliotece „vector”
2. Struktura **wezel** stworzona na potrzeby programu.

struct wezel

{

int level;

std::vector <int> path;

int bound;

};

Struktura przechowuje informacje o poziomie drzewa na którym znajduje się dany węzeł w zmiennej level. Oprócz tego przechowuje dane o dolnym ograniczeniu wyznaczonym dla danego węzła, a także ścieżkę jaką trzeba przebyć (wektor path), aby dotrzeć do tego węzłą od korzenia drzewa.

1. **Kolejka priorytetowa** dostępną w bibliotece queue, w celu przechowywania najbardziej obiecujących ścieżek w postaci wierzchołków (elementów drzewa zbudowanego z obiektów struktury wezel)
2. **Analiza przeprowadzonych badań**
   1. **Opis przeprowadzonych eksperymentów**

Wszystkie testy zostały wykonane na komputerze typu notebook - HP ProBook4540s

z procesorem Intel(R) Core(TM) i3-2370M CPU @ 2.40GHz. Do pomiaru czasu wykorzystano funkcję clock(), która znajduje się w bibliotece ctime. Jako wynik działania zwraca ona czas wykonywania programu w postaci taktów procesora, które minęły od uruchomienia. Zmierzenie czasu wykonywania algorytmu polega na wywołaniu funkcji clock() w miejscu rozpoczęcia i zakończenia algorytmu oraz wyliczeniu różnicy pomiędzy tymi wartościami - w ten sposób dowiadujemy się ile faktycznie taktów wykonywał się badany kawałek kodu programu. Następnie dzięki wykorzystaniu makra CLOCKS\_PER\_SECOND, które reprezentuje ilość taktów procesora na sekundę, otrzymujemy ostateczny czas trwania algorytmu w sekundach.

Do wygenerowania pliku użyto zaimplementowanej przez nas funkcji generujKomiwojazer(int iloscMiast), gdzie argumentem jest ilość danych jakie funkcja ma wygenerować przy pomocy standardowego generatora liczb pseudolosowych. Otrzymane dane zapisywane są w pliku „dane”, przy czym pierwszy wiersz w tym pliku oznacza ilość miast, a kolejne wiersze zawierają macierz sąsiedztwa między tymi miastami. Na przekątnej znajdują się wartości „-1”, które oznaczają brak połączenia między miastami (nie ma połączenia między tym samym miastem).

* 1. **Zestawienie wyników eksperymentu**

Dla wszystkich wykresów obowiązuje następująca legenda (umieszczona raz, aby zmniejszyć objętość sprawozdania):

|  |  |
| --- | --- |
| Metoda | Kolor linii |
| Podziału i ograniczeń |  |
| Przeglądu zupełnego |  |

* + 1. Badanie czasu działania algorytmu B&B od rozmiaru instancji. Pomiary wykonano dla liczby miast z przedziału od 5 do 20. Dla każdej ilości miast wygenerowano 100 różnych instancji, a otrzymane czasy uśredniono. Wyniki obrazuje poniższa tabela oraz wykresy:

|  |  |
| --- | --- |
| **Liczba miast** | **Czas wykonania algorytmu [us]** |
| 5 | 120.91 |
| 6 | 471.29 |
| 7 | 1891.9 |
| 8 | 2396.25 |
| 9 | 3927.5 |
| 10 | 6689.62 |
| 11 | 7723.82 |
| 12 | 17340.9 |
| 13 | 51356.2 |
| 14 | 93145.5 |
| 15 | 161613 |
| 16 | 301976 |
| 17 | 476075 |
| 18 | 938985 |
| 19 | 1462910 |
| 20 | 3035220 |

* + 1. Porównanie **czasu wykonywania algorytmu** B&B z algorytmem przeglądu zupełnego. Dla każdej ilości miast wygenerowano 100 różnych instancji, a otrzymane czasy uśredniono. Wyniki obrazuje poniższa tabela oraz wykresy:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Liczba miast | 5 | 6 | 7 | 8 | 9 | 10 |
| B&B | 151.25 us | 223.67 us | 2041.01 us | 2333.91 us | 3249.22 us | 10903 us |
| Przegląd zupełny | 9.07 us | 51.32 us | 386.83 us | 3277.23 us | 31441.2 us | 348502 us |

* + 1. Porównanie **stopnia ograniczenia drzewa przestrzeni stanów** algorytmu B&B z algorytmem przeglądu zupełnego. Dla każdej ilości miast wygenerowano 100 różnych instancji, a otrzymane czasy uśredniono. Wyniki obrazuje poniższa tabela oraz wykresy:

|  |  |  |
| --- | --- | --- |
| Liczba miast | Liczba węzłów | |
| B&B | Przegląd zupełny |
| 4 | 8 | 10 |
| 5 | 11 | 41 |
| 6 | 69 | 206 |
| 7 | 281 | 1237 |
| 8 | 401 | 8660 |
| 9 | 970 | 69281 |
| 10 | 1330 | 623530 |
| 11 | 1599 | 6235301 |
| 12 | 14671 | 68588312 |
| 13 | 13623 | 823059745 |

* 1. **Analiza wyników eksperymentu**
     1. Badanie czasu działania algorytmu B&B od rozmiaru instancji.

Na podstawie obserwacji wykresów w pierwszej kolejności obserwujemy wzrost czasu wykonania algorytmu wprost proporcjonalnie do rozmiary instancji. Ponadto, można zauważyć że zależności wzrostu czasu od rozmiaru instancji nie można ograniczyć od góry funkcją wielomianową. Czas dla rozmiarów n rośnie zgodnie z przewidywaniami określonymi we wstępie: . Obserwujemy, że dla małych n jest bliska – przyczyną jest niewielka wartość *f(n)*. Natomiast dla najwyższych z badanych wartości n złożoność czasowa wzrasta – zawdzięczamy to szybkiemu wzrostowi liczby węzłów drzewa poszukiwań odwiedzanych przez algorytm ***f(n)*** (możliwe do zaobserwowania na przedostatnim wykresie). Powodem takiej degradacji czasowej mogą być działania na wektorach.

* + 1. Porównanie czasu wykonywania algorytmu B&B z algorytmem przeglądu zupełnego.

Wykres w skali liniowej pozwala nam zaobserwować jedynie gigantyczną różnicę w czasie wykonania algorytmów dla n = 9 oraz n = 10. Algorytm podziału i ograniczeń zyskuje wtedy ogromną przewagę. O wiele ciekawszy jest jednak wykres sporządzony w skali logarytmicznej. Bardzo ciekawą obserwacją jest fakt, że algorytm przeglądu zupełnego jest szybszy dla n < 8, natomiast dla n ≥ 8 szybszy okazuje się algorytm podziału i ograniczeń. Zjawisko to jest najprawdopodobniej spowodowane wysoką złożonością obliczeniową funkcji liczącej dolne granice w algorytmie podziału i ograniczeń. Okazuje się, że dla małych instancji problemu (n < 8) szybciej można sprawdzić wszystkie rozwiązania, niż wyliczać granice dla niektórych z nich i na tej podstawie podejmować decyzję o badanych rozwiązaniach. Jednakże, im większy rozmiar instancji problemu tym większa jest korzyść z wykorzystania metody podziału i ograniczeń. Z obserwacji można wywnioskować, że im większy jest rozmiar instancji tym bardziej opłaca się użyć funkcji zmniejszającej obszar przeszukiwań kosztem wyliczenia granic. Kolejnym wnioskiem jest także, że jeśli nie zależy nam na optymalnym rozwiązaniu, a jedynie na wystarczająco dobrym to dla dużych rozmiarów instancji problemu warto wziąć pod uwagę metodę podziałów i ograniczeń.

* + 1. Porównanie stopnia ograniczenia drzewa przestrzeni stanów algorytmu B&B z algorytmem przeglądu zupełnego.

Wyniki ostatniego badania pokazują, że metoda podziałów i ograniczeń potrafi prowadzić do dużegoograniczenia drzewa przestrzeni stanów. Ograniczenie to jest tym większe, im większy jest rozmiar instancji analizowanego problemu. Różnice te są tak duże, ze dokładne zaobserwowanie ich na wykresie w skali liniowej jest trudne. O wiele bardziej przydatny okazuje się wykres w skali logarytmicznej. Pokazuje on, że statystycznie dla każdego z badanych rozmiarów wystąpił spadek ilości odwiedzanych wierzchołków drzewa (jednakże warto pamiętać że nie dla każdego z badanych rozmiarów wystąpił spadek czasu). Warto pamiętać jednak, że zwiększenie stopnia ograniczenia drzewa przestrzeni stanów nie ma bezpośredniego związku ze skróceniem czasu wykonywania algorytmu. Możemy wyciągnąć takie wnioski łącząc obserwacje z punktu 4.3.2 z obserwacjami z punktu 4.3.3. Pomimo tego, że dla małych instancji doszło do ograniczenia drzewa przestrzeni stanów, czas wykonywania algorytmu wzrósł. Nie należy zatem prowadzić optymalizacji algorytmu pod względem minimalizacji drzewa przestrzeni stanów.

1. **Bibliografia**

1. M.M. Sysło, N.Deo, J.S. Kowalik, Algorytmy optymalizacji dyskretnej z programami w języku Pascal, Wydawnictwo Naukowe PWN , 1999

2. R. Neapolitan, K. Naimipour, Podstawy Algorytmów z przykładami w c++, Helion, 2004

3. Weixiong Zhang, Branch and Bound Search Algorithms and Their Computional Complexity, 1996

4. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Wprowadzenie do algorytmów, Wydawnictwo Naukowe PWN, 2013