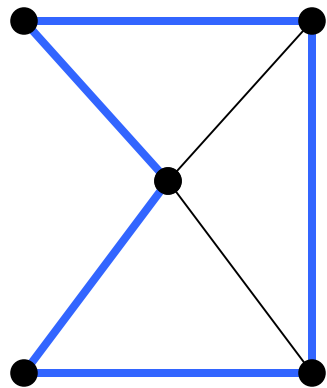


Grafy hamiltonowskie,
problem komiwojażera –
algorytm optymalny

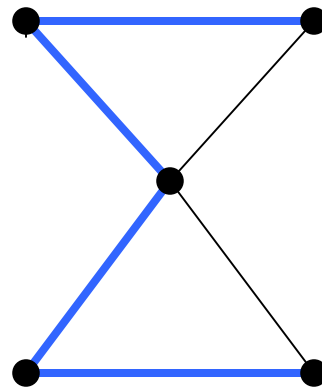
Grafy hamiltonowskie

Def. *Cykl (droga) Hamiltona* jest to cykl (droga), w którym każdy wierzchołek grafu występuje dokładnie raz. Graf jest *hamiltonowski* (*półhamiltonowski*), o ile posiada cykl (drogę) Hamiltona.

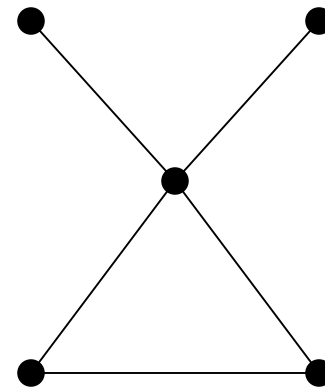
Przykład



Graf hamiltonowski



Graf półhamiltonowski

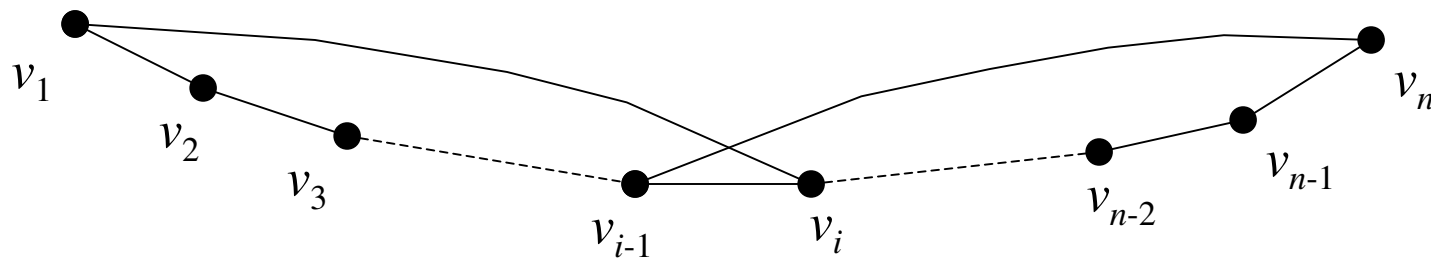


Graf nie jest ani hamiltonowski
ani półhamiltonowski

Grafy hamiltonowskie

Tw. (Ore, 1960) *Jeśli G jest grafem prostym o $n \geq 3$ wierzchołkach i $\deg(u) + \deg(v) \geq n$ dla każdej pary niesąsiednich wierzchołków u i v , to graf G jest hamiltonowski.*

Dowód: Załóżmy, że istnieje graf G o podanych założeniach ale nie jest hamiltonowski. Możemy założyć, że G posiada drogę Hamiltona $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n$ oraz $\{v_1, v_n\} \notin E(G)$. Stąd wynika, że $\deg(v_1) + \deg(v_n) \geq n$ a to oznacza, że istnieje indeks i taki, że $\{v_1, v_i\} \in E(G)$ oraz $\{v_{i-1}, v_n\} \in E(G)$, co pokazano na rysunku. To prowadzi do sprzeczności, gdyż $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_{i-1} \rightarrow v_n \rightarrow v_{n-1} \rightarrow \dots \rightarrow v_i \rightarrow v_1$ jest cyklem Hamiltona.



Grafy hamiltonowskie

Wniosek (Dirac, 1952) *Jeśli G jest grafem prostym o $n \geq 3$ wierzchołkach i $\deg(u) \geq n/2$ dla każdego wierzchołka v , to G jest hamiltonowski.*

Dowód: Wynika z poprzedniego twierdzenia, gdyż $\deg(u) + \deg(v) \geq n$ dla każdej pary (również niesąsiednich) wierzchołków.

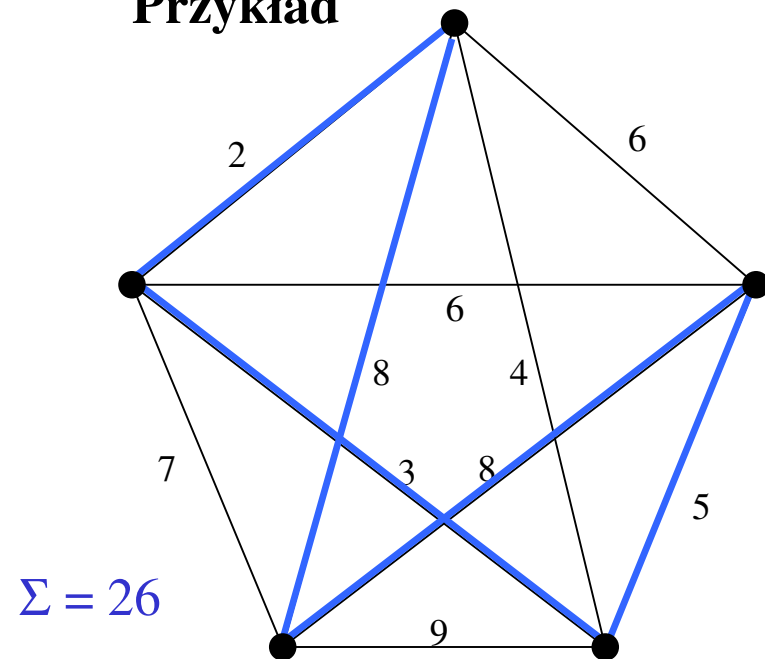
Uwaga Problem polegający na stwierdzeniu czy dany graf G jest hamiltonowski jest NP-zupełny. Oznacza to, że nie są znane efektywne (działające w czasie wielomianowym) algorytmy rozwiązujące ten problem. Nie jest również znane twierdzenie podające warunki konieczne i dostateczne na to, aby G był hamiltonowski.

Problem komiwojażera

Dany jest zbiór miast. Komiwojażer chce odwiedzić wszystkie miasta (każde dokładnie raz) i powrócić do punktu wyjścia. Problem polega na znalezieniu najkrótszej trasy o tej własności.

Zdefiniujemy powyższy problem w języku teorii grafów. Niech będzie dany graf pełny G . Zakładamy, że z każdą krawędzią e_i jest skojarzona jej waga (długość) oznaczana dalej przez w_i . Rozwiązaniem problemu komiwojażera jest taki cykl Hamiltona, którego suma wag krawędzi jest minimalna.

Przykład



Problem komiwojażera

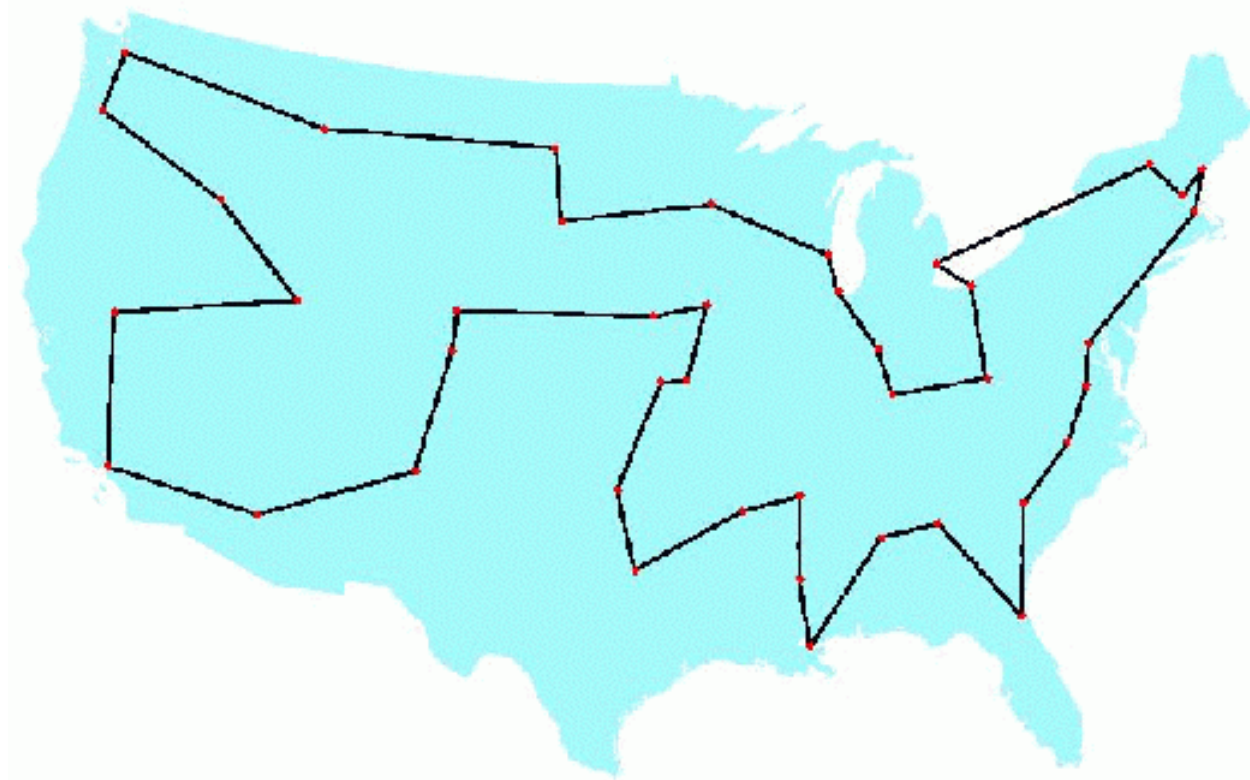
Uwagi

- problem komiwojażera jest NP-trudny, co oznacza, że nie są znane algorytmy o wielomianowej złożoności obliczeniowej rozwiązujące ten problem (przypuszczalnie takie nie istnieją)
- w praktyce jesteśmy zmuszeni posługiwać się wielomianowymi algorytmami przybliżonymi, tzn. takimi, które szybko znajdują rozwiązanie, które jest w przybliżeniu równe optymalnemu

Przykład Jednym z możliwych algorytmów dokładnych jest sprawdzenie wszystkich możliwych cykli Hamiltona i wybranie najkrótszego. Wadą takiego podejścia jest to, że liczba cykli jest zbyt duża, gdyż dla n -wierzchołkowego grafu wynosi $(n!)/2$. Stąd, jeśli dysponujemy komputerem sprawdzającym milion permutacji na sekundę, to:

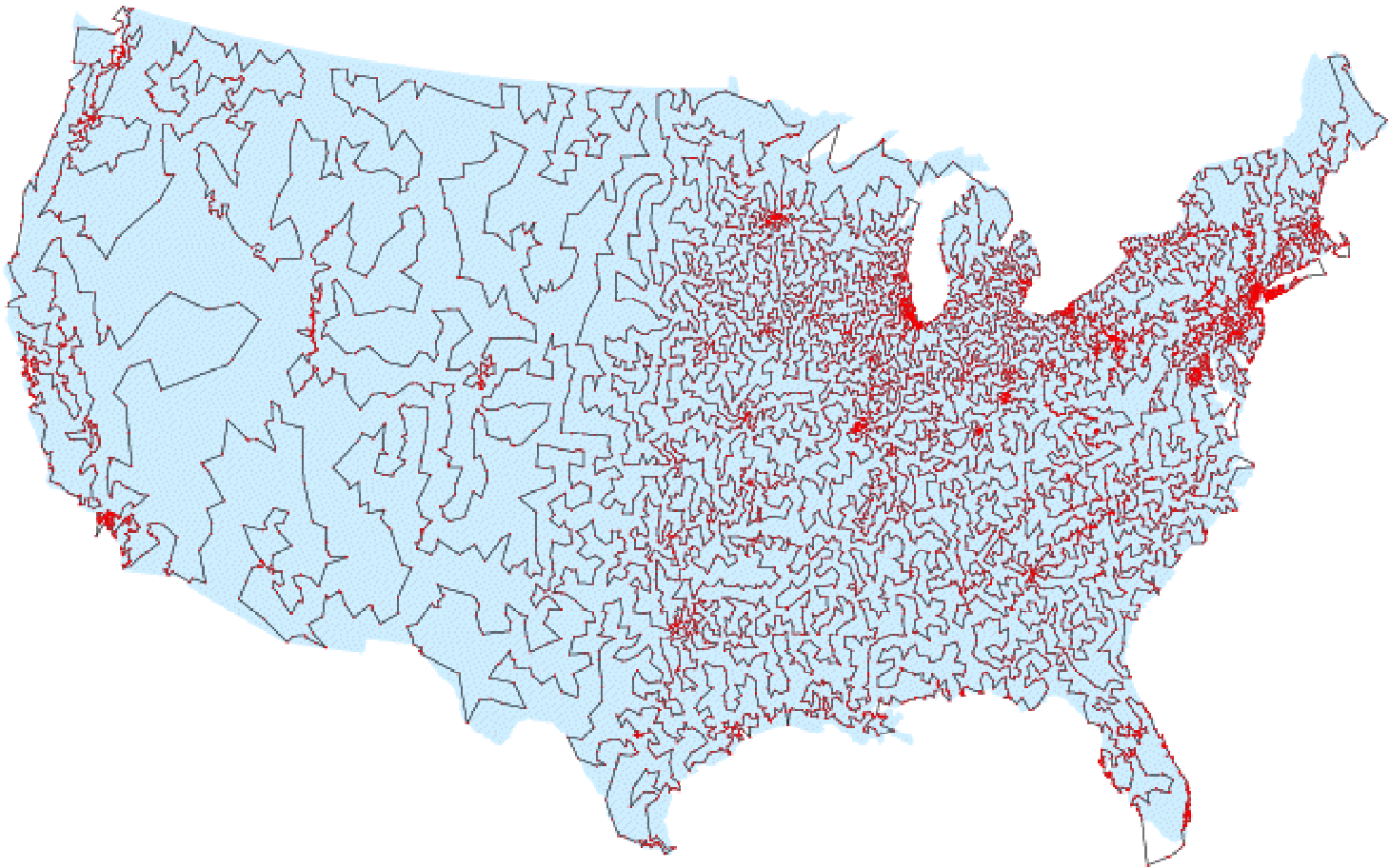
- $n = 10 \Rightarrow$ ilość cykli = $(10!)/2 = 1814400 \Rightarrow$ czas obliczeń = 1.8 s
- $n = 20 \Rightarrow$ ilość cykli = $(20!)/2 \approx 10^{18} \Rightarrow$ czas obliczeń \approx 40 tys. lat

Historia problemu komiwojażera



George Dantzig, Ray Fulkerson i Selmer Johnson (1954) zaprezentowali optymalne rozwiązanie problemu komiwojażera dla 49 amerykańskich miast.

Historia problemu komiwojażera



Rozwiązanie obejmujące 13549 miast amerykańskich, uzyskane w 1998 roku.

Problem komiwojażera – algorytm optymalny

- nie jest znany żaden wielomianowy optymalny algorytm dla tego problemu i jest mało prawdopodobne, że taki algorytm w ogóle istnieje
- omówiony dalej algorytm polega na przeszukiwaniu całej przestrzeni rozwiązań
- podczas obliczeń na bieżąco uaktualniane jest dolne oszacowanie na długość optymalnej trasy, dzięki czemu wiemy, których rozwiązań częściowych na pewno nie da się rozszerzyć na rozwiązania optymalne i część obliczeń można pominąć
- rozważamy przypadek nieco ogólniejszy, w którym dany jest na wejściu obciążony graf skierowany

Drzewo przeszukiwań

Def. *Drzewo przeszukiwań* definiujemy jako zakorzenione drzewo, którego każdy wierzchołek odpowiada pewnemu podzbiorowi rozwiązań. Podzbiory rozwiązań odpowiadające synom węzła wynikają z sposobu podziału zbioru rozwiązań ojca.

Uwaga: Dla problemu komiwojażera przyjmujemy następującą postać drzewa przeszukiwań:

- każdy wierzchołek odpowiada rozwiązaniom problemu, które zawierają pewne łuki i jednocześnie innych wybranych łuków nie zawierają (np. pewnemu wierzchołkowi odpowiadają optymalne trasy zawierające łuki (a,b) , (e,h) oraz nie zawierające łuków (a,d) , (d,e) i (b,e))
- Każdy węzeł ma dwóch synów. Po wybraniu nowego łuku e , jeden z synów odpowiada rozwiązaniom o ograniczeniach nałożonych w ojcu oraz zawierających e , natomiast drugi – nie zawierających e .

Oszacowanie dolne

Uwaga: Podczas realizacji algorytmu (tzn. podczas trawersowania drzewa przeszukiwań) pamiętamy wartość najlepszego znalezione dotychczas rozwiązania. Oznaczmy ją przez min_sol .

Uwaga: Z każdym wierzchołkiem v drzewa przeszukiwań jest związana zmienna LB . Jest to liczba, która stanowi oszacowanie dolne na wartość każdego rozwiązania należącego do tego wierzchołka. Wówczas:

- jeśli $LB > min_sol$, to wiemy, że nie warto przeszukiwać poddrzewa zakorzenionego w wierzchołku v ,
- jeśli $LB = min_sol$, to poddrzewo być może zawiera rozwiązania dorównujące dotychczasowemu najlepszemu. Jeśli zadanie polega na wyznaczeniu dowolnego rozwiązania optymalnego, to nie przeszukujemy poddrzewa zakorzenionego w wierzchołku v
- jeśli $LB < min_sol$, to należy przeszukiwać poddrzewo zakorzenione w v (być może nie całe).

Redukcja macierzy

Lemat *Jeśli M jest macierzą sąsiedztwa grafu G , to:*

- *do dowolnego cyklu Hamiltona należy dokładnie jeden element z każdego wiersza M i dokładnie jeden z każdej kolumny*
- *jeśli od wszystkich elementów w wybranym wierszu (kolumnie) odejmiemy stałą d , to długość każdego cyklu Hamiltona jest o d mniejsza od długości tego samego cyklu, lecz przed odjęciem stałej*
- *jeśli od wierszy i i kolumn j wielokrotnie odejmiemy stałe tak, aby każdy wiersz i i kolumna j zawierały co najmniej jedno zero, to suma odejmych liczb stanowi dolne oszacowanie optymalnego rozwiązania.*

Def. Proces odejmowania stałych od wierszy (kolumn) macierzy sąsiedztwa nazywamy *redukcją*.

Wniosek *Jeśli łuk (i,j) należy do optymalnej trasy komiwojażera znalezionej na podstawie zredukowanej macierzy sąsiedztwa, to (i,j) należy również do optymalnej trasy w wyjściowym grafie.*

Algorytm redukcji

```

procedure Reduce(  $M$  )
begin
   $r := 0$ ;
  for  $i := 1$  to  $n$  do begin
     $min\_row :=$  najmniejszy element w  $i$ -tym wierszu;
    if (  $min\_row > 0$  ) then begin
      odejmij  $min\_row$  od każdego elementu w wierszu  $i$ ;
       $r := r + min\_row$ ;
    end
  end;
  for  $i := 1$  to  $n$  do begin
     $min\_col :=$  najmniejszy element w  $i$ -tej kolumnie;
    if (  $min\_col > 0$  ) then begin
      odejmij  $min\_col$  od każdego elementu w wierszu  $i$ ;
       $r := r + min\_col$ ;
    end
  end;
  return  $r$ ;
end

```

Zmienne:

M – macierz sąsiedztwa
rozmiaru n
 r – suma odejtych
wartości od wierszy i
kolumn (jak wynika z
poprzedniego lematu,
jest to dolne
oszacowanie na
długość cyklu w M)

Przykład redukcji

$$M = \begin{array}{c} a \\ b \\ c \\ d \\ e \end{array} \begin{array}{ccccc} a & b & c & d & e \\ \left[\begin{array}{ccccc} \infty & 11 & 16 & 28 & 42 \\ 17 & \infty & 44 & 31 & 27 \\ 24 & 33 & \infty & 6 & 15 \\ 13 & 19 & 41 & \infty & 21 \\ 42 & 28 & 25 & 36 & \infty \end{array} \right] \end{array} \Rightarrow \begin{array}{c} a \\ b \\ c \\ d \\ e \end{array} \begin{array}{ccccc} a & b & c & d & e \\ \left[\begin{array}{ccccc} \infty & 0 & 5 & 17 & 31 \\ 0 & \infty & 27 & 14 & 10 \\ 18 & 27 & \infty & 0 & 9 \\ 0 & 6 & 28 & \infty & 8 \\ 17 & 3 & 0 & 11 & \infty \end{array} \right] \end{array} \begin{array}{l} 11 \\ 17 \\ 6 \\ 13 \\ 25 \end{array}$$

$$r = 11 + 17 + 6 + 13 + 25 + 8 \\ = 80$$

Stąd, do wartości LB
potomków węzła dodamy 80

$$\begin{array}{c} a \\ b \\ c \\ d \\ e \end{array} \begin{array}{ccccc} a & b & c & d & e \\ \left[\begin{array}{ccccc} \infty & 0 & 5 & 17 & 23 \\ 0 & \infty & 27 & 14 & 2 \\ 18 & 27 & \infty & 0 & 1 \\ 0 & 6 & 28 & \infty & 0 \\ 17 & 3 & 0 & 11 & \infty \end{array} \right] \end{array} \begin{array}{l} 11 \\ 17 \\ 6 \\ 13 \\ 25 \end{array}$$

$$\begin{array}{ccccc} 0 & 0 & 0 & 0 & 8 \end{array}$$

Kryterium wyboru łuku

procedure FindEdge(M, r, c)

begin

$max := -1;$

for $i := 1$ **to** n **do**

for $j := 1$ **to** n **do**

if $M[i,j] = 0$ **then begin**

$min_r :=$ wartość najmniejszego

elementu w wierszu i z pominięciem $M[i,j];$

$min_c :=$ wartość najmniejszego

elementu w kolumnie j z pominięciem $M[i,j];$

if $min_r + min_c > max$ **then begin**

$max := min_r + min_c;$

$(r,c) := (i,j);$

end

end;

return $max;$

end

Zmienne:

M – macierz sąsiedztwa

n – rozmiar M

(r,c) – łuk do podziału
zbioru rozwiązań

Uwaga: Aby utworzyć potomków w drzewie przeszukiwań, wybieramy taki łuk, który powoduje największy wzrost dolnego oszacowania w prawym poddrzewie. Wartość, o którą wzrośnie LB wyznaczamy w zmiennej max .

Wybór łuku - przykład

$$\min_r + \min_c = 5 + 3 = 8$$

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>		
$2+0=2$ ←	<i>a</i>	∞	0	5	17	23	→ $1+11=12$
Zredukow. $M =$	<i>b</i>	0	∞	27	14	2	
	<i>c</i>	18	27	∞	0	1	→ $0+1=1$
$0+0=0$ ←	<i>d</i>	0	6	28	∞	0	
	<i>e</i>	17	3	0	11	∞	→ $3+5=8$

- do podziału zbioru rozwiązań wybieramy łuk (c,d)
- lewy potomek odpowiada wszystkim rozwiązaniom (cyklom) zawierającym łuk (c,d)
- prawy potomek zawiera wszystkie rozwiązania bez (c,d)

Tworzenie lewego syna

1. założymy, że wybrano łuk (c,d) w celu utworzenia potomków wierzchołka v ,
2. lewy syn zawiera wówczas zbiór rozwiązań o tych samych ograniczeniach, co w przypadku v oraz dodatkowo zawierających łuk (c,d) ,
3. oznacza to, że możemy zmniejszyć rozmiar macierzy sąsiedztwa o 1 poprzez usunięcie c -tego wiersza i d -tej kolumny,
4. kolejne „uproszczenie” macierzy polega na *zablokowaniu* łuku (d,c) tzn. element macierzy na przecięciu d -tego wiersza i c -tej kolumny przyjmuje wartość „nieskończoność”,
5. blokujemy również łuk, który tworzy cykl wraz z łukami dodanymi poprzednio do rozwiązania,
6. wartość LB wyliczamy dodając do wartości LB ojca liczbę r wyliczoną w procedurze Reduce

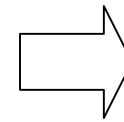
Lewy syn - przykład

Dla węzła
wyjściowego v
(tutaj korzeń
drzewa) $LB(v)=0$

Zredukow. $M =$

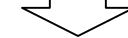
	a	b	c	d	e
a	∞	0	5	17	23
b	0	∞	27	14	2
c	18	27	∞	0	1
d	0	6	28	∞	0
e	17	3	0	11	∞

(usunięcie wiersza
 c i kolumny d)



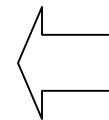
	a	b	c	e
a	∞	0	5	23
b	0	∞	27	2
d	0	6	28	0
e	17	5	0	∞

(zablokowanie
łuku (d,c))



Wartość
oszacowania
dolnego $LB(v_l)$ dla
lewego potomka
wynosi zatem
 $LB(v)+r = 0+80$

	a	b	c	e
a	∞	0	5	23
b	0	∞	27	2
d	0	6	∞	0
e	17	5	0	∞



(blokowanie
łuków tworzących
cykl z dotychczas
wybranymi)

	a	b	c	e
a	∞	0	5	23
b	0	∞	27	2
d	0	6	∞	0
e	17	5	0	∞

Tworzenie prawego syna

1. założymy, że wybrano łuk (c,d) w celu utworzenia potomków wierzchołka v ,
2. prawy syn zawiera wówczas zbiór rozwiązań o tych samych ograniczeniach, co w przypadku v oraz dodatkowo nie zawierających łuku (c,d) ,
3. blokujemy więc łuk (c,d) poprzez wpisanie wartości „nieskończoność” na przecięciu c -tego wiersza i d -tej kolumny w macierzy sąsiedztwa
4. nie następuje zmniejszenie rzędu macierzy sąsiedztwa w tym przypadku
5. wartość LB wyliczamy dodając do wartości LB ojca liczbę r wyliczoną w procedurze Reduce oraz wartość max wyliczoną w procedurze FindEdge

Prawy syn - przykład

Dla węzła
wyjściowego r
(tutaj korzeń
drzewa) $LB(r)=0$

Zredukow. $M =$

	a	b	c	d	e
a	∞	0	5	17	23
b	0	∞	27	14	2
c	18	27	∞	0	1
d	0	6	28	∞	0
e	17	3	0	11	∞

(zablokowanie
łuku (c,d))

	a	b	c	d	e
a	∞	0	5	17	23
b	0	∞	27	14	2
c	18	27	∞	∞	1
d	0	6	28	∞	0
e	17	3	0	11	∞

Wartość oszacowania dolnego
 $LB(r_p)$ dla prawego potomka
wynosi zatem
 $LB(r) + r + max = 0 + 80 + 12 = 92$

Warunki końca rekurencji

Przypadek 1: wartość LB w wierzchołku v jest większa lub równa od najlepszego znalezione dotychczas rozwiązania. Wówczas drzewo zakorzenione w v nie jest przeszukiwane.

Przypadek 2: M jest stopnia 2. Ma ona wówczas jedną z dwóch postaci:

$$M = \begin{bmatrix} +\infty & 0 \\ 0 & +\infty \end{bmatrix} \quad \text{lub} \quad M = \begin{bmatrix} 0 & +\infty \\ +\infty & 0 \end{bmatrix}.$$

Zatem bez względu na postać macierzy nie ma wyboru co do tego jakie łuki należy włączyć do końcowego rozwiązania. Jeśli kolumny odpowiadają wierzchołkom w, x natomiast wiersze u, v to:

- jeśli $M[u, w] = 0$, to do cyklu komiwojażera należą łuki (u, w) , (v, x)
- jeśli $M[u, x] = 0$, to do cyklu komiwojażera należą łuki (v, w) , (u, x)

Algorytm

procedure TraverseTree(M, C, LB)

begin

$r := \text{Reduce}(M);$

if $LB + r < min_sol$ **then**

if $|C| = n - 2$ **then begin**

dołącz dwa łuki do C i uaktualnij min_sol oraz zapamiętaj
nowe rozwiązanie jeśli jest lepsze od dotychczasowych;

end else begin

$max := \text{FindEdge}(M, c, d);$

TraverseTree($M^*, C \cup \{(c,d)\}, LB + r$);

if $LB + r + max < min_sol$ **then begin**

$M[c,d] := +\infty;$

TraverseTree($M, C, LB + r$); $M[r,c] := 0;$

end

end;

odtwórz macierz M do postaci sprzed redukcji;

end

Zmienne:

M – macierz sąsiedztwa

C – krawędzie należące do cyklu

LB – wartość dolnego oszacowania
dla danego węzła

M^* powstaje z M poprzez
usunięcie c -tego wiersza, d -tej
kolumny i zablokowania łuku
(d,c) i łuków tworzących cykle z
 $C \cup \{(c,d)\}$

min_sol inicjalnie równe $+\infty$

Przykład

$$M = \begin{matrix} & a & b & c & d & e \\ a & \infty & 42 & 6 & 28 & 35 \\ b & 9 & \infty & 29 & 35 & 38 \\ c & 22 & 13 & \infty & 29 & 34 \\ d & 17 & 21 & 14 & \infty & 2 \\ e & 23 & 24 & 31 & 5 & \infty \end{matrix} \quad \begin{matrix} LB=0 \\ r=35 \\ max=34 \end{matrix}$$

$$\begin{matrix} LB=35 \\ r=0 \\ max=41 \end{matrix} \begin{matrix} & a & b & d & e \\ b & 0 & \infty & 26 & 29 \\ c & \infty & 0 & 16 & 21 \\ d & 15 & 19 & \infty & 0 \\ e & 18 & 19 & 0 & \infty \end{matrix} \quad \begin{matrix} \swarrow (a,c) \\ \searrow (a,c) \end{matrix}$$

$$\begin{matrix} LB=69 \\ r=34 \\ max=23 \end{matrix} \begin{matrix} & a & b & c & d & e \\ a & \infty & 14 & \infty & 0 & 7 \\ b & 0 & \infty & 8 & 26 & 29 \\ c & 9 & 0 & \infty & 16 & 21 \\ d & 15 & 19 & 0 & \infty & 0 \\ e & 18 & 19 & 14 & 0 & \infty \end{matrix}$$

$$\begin{matrix} LB=35 \\ r=35 \\ max=5 \end{matrix} \begin{matrix} & b & d & e \\ c & \infty & 0 & 5 \\ d & 0 & \infty & 0 \\ e & 0 & 0 & \infty \end{matrix} \quad \begin{matrix} \swarrow (b,a) \\ \searrow (b,a) \end{matrix}$$

$$35+0+41 > min_sol$$

$$69+34 > min_sol$$

$$69+34+23 > min_sol$$

$$\begin{matrix} LB=70 \\ r=0 \end{matrix} \begin{matrix} & b & d \\ c & \infty & 0 \\ e & 0 & \infty \end{matrix} \quad \begin{matrix} \swarrow (d,e) \\ \searrow (d,e) \end{matrix}$$

$$LB+r+max = 35+35+5=75 > min_sol=70$$

$$min_sol = 70$$

M – macierz wyjściowa (przed redukcją), natomiast we wszystkich potomkach pokazano macierze po redukcji

Złożoność

- Czas działania procedury Reduce wynosi $O(n^2)$
- Czas działania procedury FindEdge wynosi $O(n^3)$
- Zapamiętanie i odtworzenie macierzy to operacja rzędu $O(n^2)$
- Zapamiętywanie nowego najlepszego rozwiązania w czasie $O(n)$
- Oznacza to, że realizacja algorytmu TraverseTree w obrębie jednego węzła wymaga czasu $O(n^3)$
- Złożoność całego algorytmu można oszacować zatem przez $O(f(n)n^3)$, gdzie $f(n)$ jest liczbą węzłów drzewa poszukiwań odwiedzanych przez procedurę TraverseTree.
- Liczba wykonanych obliczeń zależy od konkretnych danych wejściowych i w pesymistycznym przypadku jest wykładnicza.