

Heurystyki dla (symetrycznego) problemu komiwojażera

- Algorytm włączania
- Heurystyki lokalnych poszukiwań

Algorytm włączania

Uwaga Rozważamy problem komiwojażera dla obciążonych grafów prostych

Szkic algorytmu:

- wybierz dowolny wierzchołek v_0 grafu jako początkowy wierzchołek cyklu
- założmy, że został wyznaczony fragment cyklu zawierający wierzchołki v_0, \dots, v_k . W celu rozszerzenia takiego częściowego rozwiązania na $(k+1)$ -elementowy fragment cyklu wykonywane są dwa kroki:
 - *krok wyboru*: wybierz wierzchołek v spośród wierzchołków nie należących do cyklu
 - *krok włączania*: określ miejsce w dotychczas utworzonym fragmencie cyklu, w które należy wstawić wierzchołek v

Kryteria wyboru

Przykładowe kryteria stosowane podczas kroku wyboru to:

- wybór losowego wierzchołka spośród wierzchołków nie włączonych jeszcze do cyklu
- wierzchołek leżący najbliżej dotychczas zdefiniowanego fragmentu cyklu
- dla każdego wierzchołka wyznaczyć koszt jego włączenia do cyklu w najbardziej korzystnym dla niego miejscu i wybrać wierzchołek o minimalnym koszcie
- wybór wierzchołka znajdującego się najdalej od cyklu

Uwagi:

- brak teoretycznych analiz porównujących powyższe podejścia
- testy komputerowe wskazują, że włączanie najdalszego wierzchołka okazuje się być techniką najskuteczniejszą w praktyce

Algorytm włączania

1) *krok wyboru:*

- w celu zwiększenia efektywności wykorzystujemy tablicę $d[1, \dots, n]$ taką, że $d[i]$ jest najkrótszą drogą z wierzchołka v_i do fragmentu dotychczas utworzonego cyklu
- w kroku wyboru wybieramy wierzchołek v , któremu odpowiada największa wartość w tablicy d
- uaktualnienie tablicy d : dla każdego u , $d[u] = \min\{d[u], w(\{u, v\})\}$

2) *krok włączania:*

- obliczamy długość każdego z następujących częściowych rozwiązań:

$$v_0, v, v_1, \dots, v_k$$

$$v_0, v_1, v, v_2, \dots, v_k$$

$$\vdots$$

$$v_0, \dots, v_k, v$$

- wybieramy rozwiązanie częściowe o najmniejszym koszcie

Pseudokod algorytmu

```

procedure FSTP(  $G, n$  );
begin
   $V(C) := \{u\}; E(C) := \{\{u,u\}\}; len := 0;$ 
  for each  $v \neq u$  do
     $d[v] := w(\{u,v\});$ 
  while  $V(C) \neq V(G)$  do begin
     $v$  – wierzchołek o największej wartości  $d$  spośród  $V(G) \setminus V(C)$ ;
     $min := +\infty;$ 
    for each  $\{x,y\} \in E(C)$  do
      if  $w(\{x,v\}) + w(\{v,y\}) - w(\{x,y\}) < min$  then begin
         $min := w(\{x,v\}) + w(\{v,y\}) - w(\{x,y\});$ 
         $\{x',y'\} := \{x,y\};$ 
      end;
    for each  $z \in V(G) \setminus V(C)$  do
       $d[z] := \min\{d[z], w(\{v,z\});$ 
       $len := len + w(\{x',v\}) + w(\{v,y'\}) - w(\{x',y'\});$ 
    end
  end
end

```

Zmienne:

G – obciążony graf prosty

n – rząd grafu G

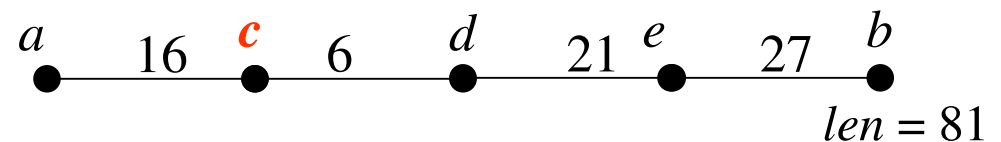
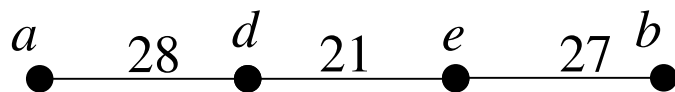
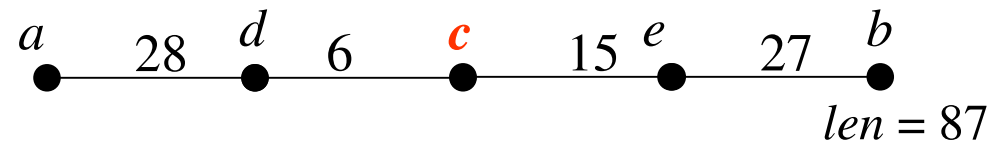
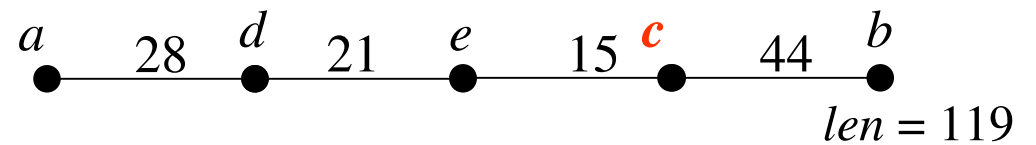
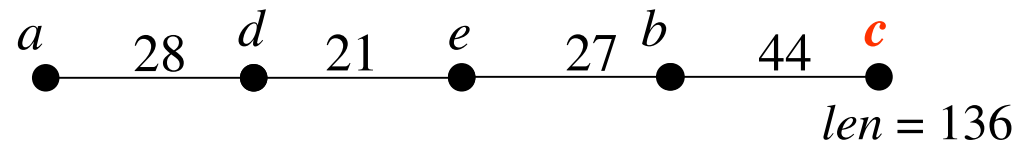
u – dowolny wierzchołek grafu G

len – długość

znalezionego cyklu

Przykład c.d.

$$M = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} \infty & 11 & 16 & 28 & 42 \\ 11 & \infty & 44 & 31 & 27 \\ 16 & 44 & \infty & 6 & 15 \\ 28 & 31 & 6 & \infty & 21 \\ 42 & 27 & 15 & 21 & \infty \end{bmatrix} \end{matrix}$$



4) $len = 87$ $d = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} 0 & 0 & 6 & 0 & 0 \end{bmatrix} \end{matrix}$

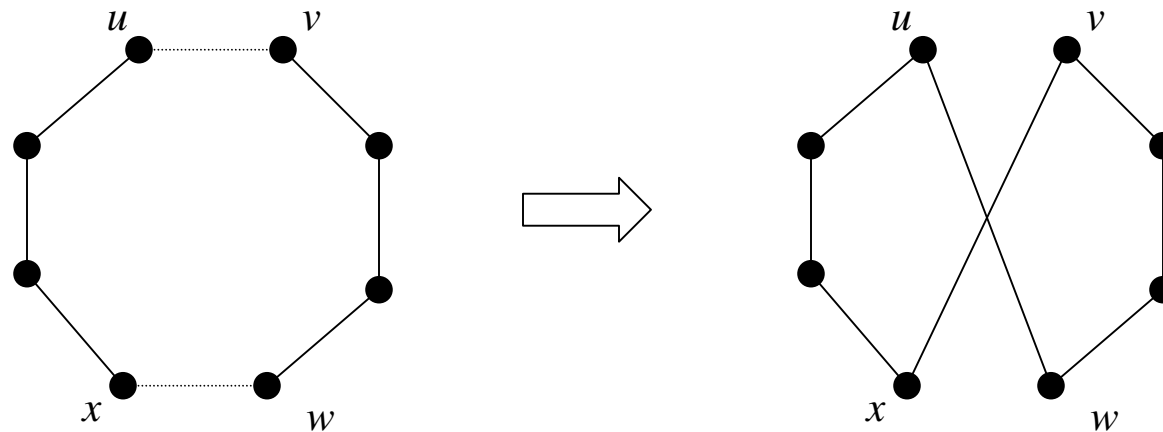
Ostatecznie: $len = 81$

Heurystyki lokalnych
poszukiwań – algorytmy
r-optymalne

Algorytm 2-opt

- Załóżmy, że mamy pewien cykl komiwojażera C dla grafu G złożony z krawędzi e_1, \dots, e_n
- wybieramy dwie krawędzie e_i, e_j , które nie są sąsiednie
- oznaczmy $e_i = \{u, v\}$ oraz $e_j = \{w, x\}$
- cykl C przekształcamy w cykl C' w taki sposób, że

$$C' = (C \setminus \{e_i, e_j\}) \cup \{\{u, w\}, \{v, x\}\},$$
 gdzie wierzchołki u, w należą do różnych składowych podgrafu $C \setminus \{e_i, e_j\}$



Algorytm 2-opt

- przy oznaczeniach z poprzedniego slajdu możemy napisać, że koszt nowego cyklu C' wynosi

$$w(C') = w(C) - w(\{u,v\}) - w(\{w,x\}) + w(\{u,w\}) + w(\{v,x\})$$

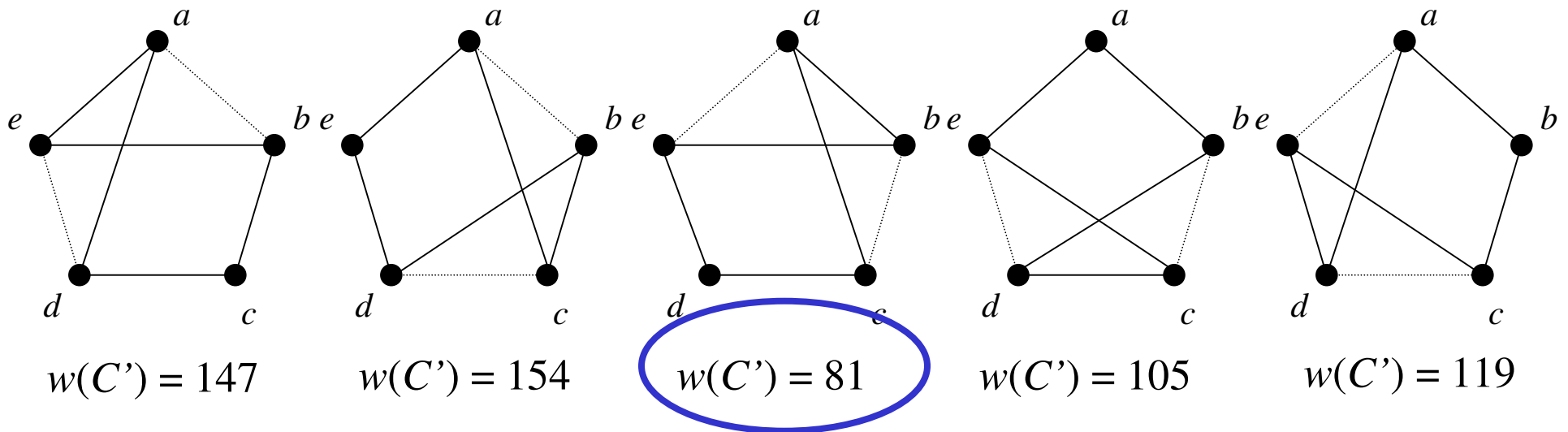
- zatem, jeśli $w(\{u,v\}) + w(\{w,x\}) > w(\{u,w\}) + w(\{v,x\})$, to nowe rozwiązanie jest lepsze od dotychczasowego
- w celu wyznaczenia cyklu C' algorytm szuka takiej pary krawędzi e_i, e_j , aby maksymalnie obniżyć koszt nowego bieżącego rozwiązania
- jeśli zmniejszenie wagi cyklu C nie jest możliwe, to C jest rozwiązaniem *2-optymalnym* i algorytm kończy działanie
- powyższa procedura przekształcania bieżącego cyklu C jest powtarzana dopóki możliwa jest redukcja wagi cyklu dzięki opisanej wcześniej wymianie krawędzi

Przykład

Dana jest macierz sąsiedztwa M
 oraz bieżącym cyklem jest $C =$
 $\{\{a,b\}, \{b,c\}, \{c,d\}, \{d,e\}, \{e,a\}\}$.
 Mamy $w(C)=124$.

Szukamy bieżącego cyklu dla
 kolejnej iteracji algorytmu:

$$M = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{bmatrix} \infty & 11 & 16 & 28 & 42 \\ 11 & \infty & 44 & 31 & 27 \\ 16 & 44 & \infty & 6 & 15 \\ 28 & 31 & 6 & \infty & 21 \\ 42 & 27 & 15 & 21 & \infty \end{bmatrix} \end{matrix}$$



Zatem bieżący cykl dla kolejnej iteracji zawiera kraw. $\{a,b\}, \{b,e\}, \{d,e\}, \{c,d\}, \{a,c\}$.

Algorytm 3-opt

- algorytm 3-optimalny jest uogólnieniem algorytmu 2-optimalnego
- w danej iteracji usuwane są trzy krawędzie x, y, z z cyklu C
- następnie do cyklu dodawane są trzy krawędzie e_1, e_2, e_3 w taki sposób, aby graf $C' = (C \setminus \{x, y, z\}) \cup \{e_1, e_2, e_3\}$ tworzył cykl
- istnieje wiele możliwości wyboru krawędzi e_1, e_2, e_3
- algorytm zmiany bieżącego cyklu można opisać następująco:

$d := +\infty;$

dla każdej możliwej trójki krawędzi x, y, z należącej do C :

dla wszystkich e_1, e_2, e_3 t.ż. $(C \setminus \{x, y, z\}) \cup \{e_1, e_2, e_3\}$ tworzy cykl:

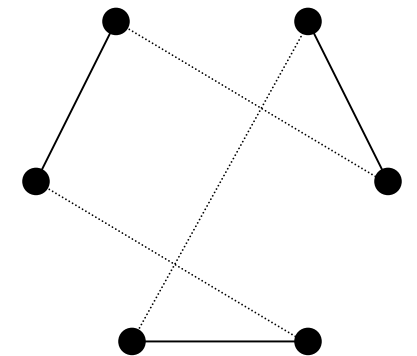
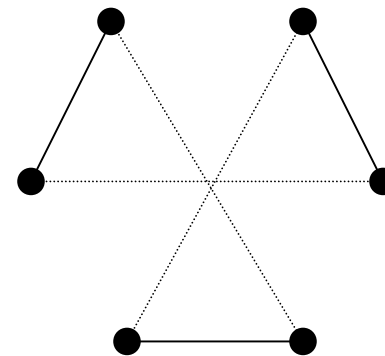
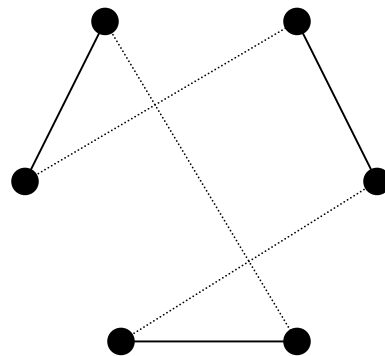
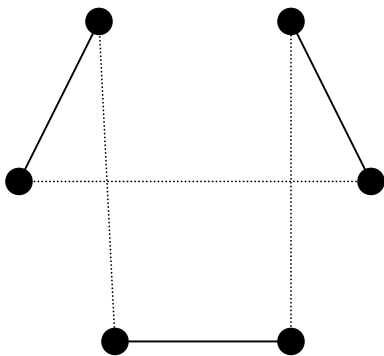
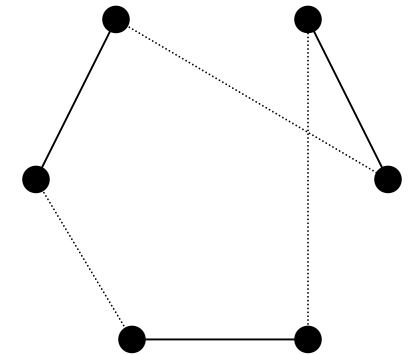
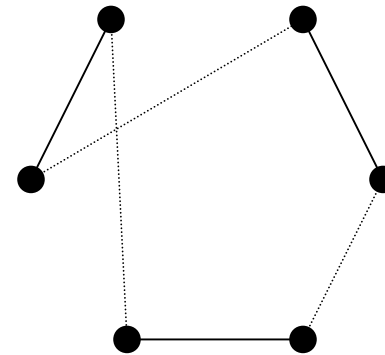
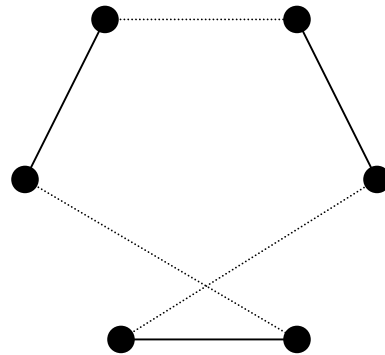
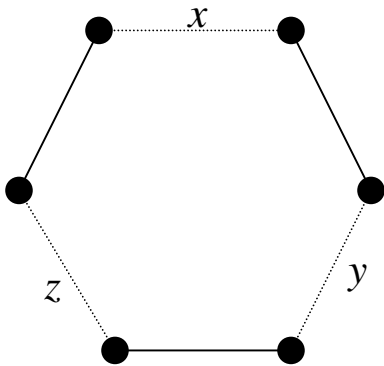
jeśli $w((C \setminus \{x, y, z\}) \cup \{e_1, e_2, e_3\}) < d$ to

$d := w((C \setminus \{x, y, z\}) \cup \{e_1, e_2, e_3\});$

jeśli $w(C) > d$ to utwórz nowy bieżący cykl C ; (*)

- algorytm kończy działanie, gdy warunek (*) jest fałszywy
- jeśli w-k (*) jest spełniony, to podane kroki są powtarzane dla nowego cyklu

8 możliwości uzupełnienia zbioru $C \setminus \{x, y, z\}$ do cyklu



Algorytm r -opt

- w każdej iteracji usuwamy r łuków (ich zbiór oznaczmy przez A) z bieżącego cyklu C
- do zbioru dróg $C - A$ dodajemy r łuków (zbiór dodanych łuków oznaczmy przez B)
- w każdej iteracji badamy wszystkie dopuszczalne zbiory A i B i wybieramy takie rozwiązanie, że $w((C \setminus A) \cup B)$ jest minimalne
- przez dopuszczalne zbiory rozumiemy takie, że $(C \setminus A) \cup B$ jest cyklem

Uwaga *Rozwiązanie r -opt jest również rozwiązaniem $(r - 1)$ -optymalnym.*

Efektywność

- liczba możliwych wyborów krawędzi A wynosi $O(n!/(r!(n-r)!))$
- zbiór CA można uzupełnić do cyklu na $O(2^{r-1}(r-1)!)$ sposobów
- złożoność całego algorytmu wynosi zatem

$$O(p(n)2^{r-1}(r-1)! \binom{n}{r})$$

Uwagi:

- w praktyce algorytm 3-opt okazuje się dużo lepszy niż 2-opt, jednak algorytm 4-opt nie jest znacząco lepszy niż 3-opt
- wynik końcowy zależy od wyboru cyklu początkowego – nie jest prawdą, że lepszy cykl początkowy prowadzi do znalezienia lepszego rozwiązania, jednak testy komputerowe wskazują, że warto wybierać jako punkt startowy dla algorytmu cykl o możliwie małej wadze