Techniki optymalizacji

Dokładne algorytmy optymalizacji

Maciej Hapke
maciej.hapke at put.poznan.pl

Problem optymalizacji kombinatorycznej

- Problem optymalizacji kombinatorycznej jest problemem **minimalizacji** bądź **maksymalizacji** i charakteryzuje się zbiorem **instancji**
- Instancja jest parą (S, f)
  - S oznacza skończony zbiór wszystkich możliwych rozwiązań
  - f jest odwzorowaniem definiowanym jako

$$f : S \rightarrow \boldsymbol{R}$$

Minimalizacja

- W przypadku minimalizacji należy znaleźć takie rozwiązanie $i_{opt} \in S$ które spełnia

$f(i_{opt}) \leq f(i)$ dla wszystkich $i \in S$

Maksymalizacja

- W przypadku maksymalizacji należy znaleźć takie rozwiązanie $i_{opt} \in S$ które spełnia

$$f(i_{opt}) \geq f(i) \quad \text{dla wszystkich } i \in S$$

- Takie rozwiązanie i_{opt} jest nazywane *globalnie optymalne* (*minimalne* lub *maksymalne*)

Klasyfikacja algorytmów

- algorytmy optymalizacyjne dokładne
  - przeszukiwanie wyczerpujące (exhaustive),
  - podziału i ograniczeń (B&B),
  - programowanie dynamiczne
- algorytmy heurystyczne
  - specjalizowane
  - losowego przeszukiwania (random search)
  - lokalnego przeszukiwania i odmiany
    - symulowane wyżarzanie
    - przeszukiwanie tabu
  - genetyczne
  - hybrydy

Klasyfikacja algorytmów (2)

- Ponadto w obu klasach można wyróżnić
  - algorytmy ogólne (general algorithms) – niezależne od rozwiązywanego problemu (np. B&B)
  - oraz algorytmy dopasowywane (tailored algorithms) – wykorzystujące specyficzną wiedzę o problemie (np. heurystyki priorytetowe w szeregowaniu)

Co jest potrzebne do zaprojektowania algorytmu

- Reprezentacja rozwiązania
- Cel
- Funkcja oceny

Reprezentacja rozwiązania

- Ciąg binarny, np. SAT
- Reprezentacja zmiennoprzecinkowa, np. programowanie nieliniowe
- Permutacja, np. TSP, QAP
- Macierz
- Drzewo

Exhaustive search

- Wymaga wygenerowania i sprawdzenia każdego rozwiązania dopuszczalnego – wady oczywiste
- Zaleta – proste
- Jedyne wymaganie – systematyczny sposób przeszukiwania *S*
- Jak to zrobić, zależy od reprezentacji

Exhaustive search - SAT

0000 – 0

0001 – 1

0010 – 2

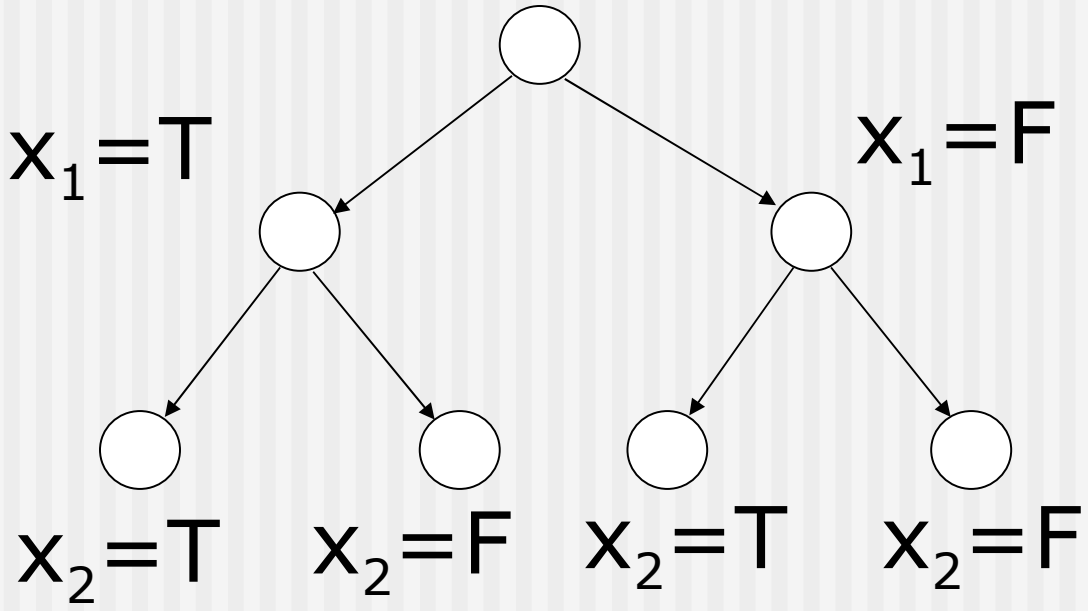
0011 – 3

…

Ale można inaczej dzieląc S. Jak?

Exhaustive search - SAT (2)

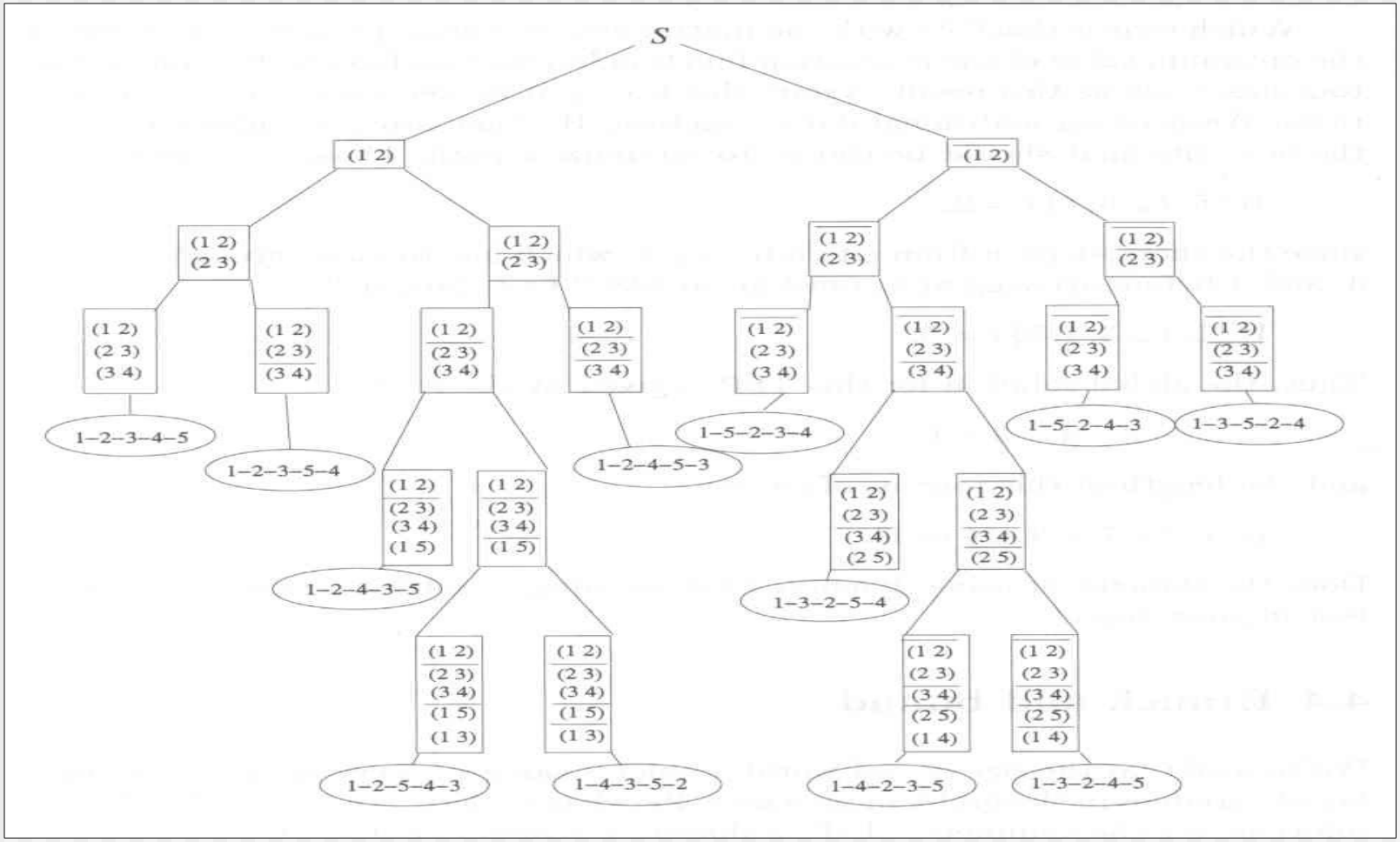
- Wielokrotny podział na dwie przestrzenie, $x_1 = T$, $x_1 = F$, ...



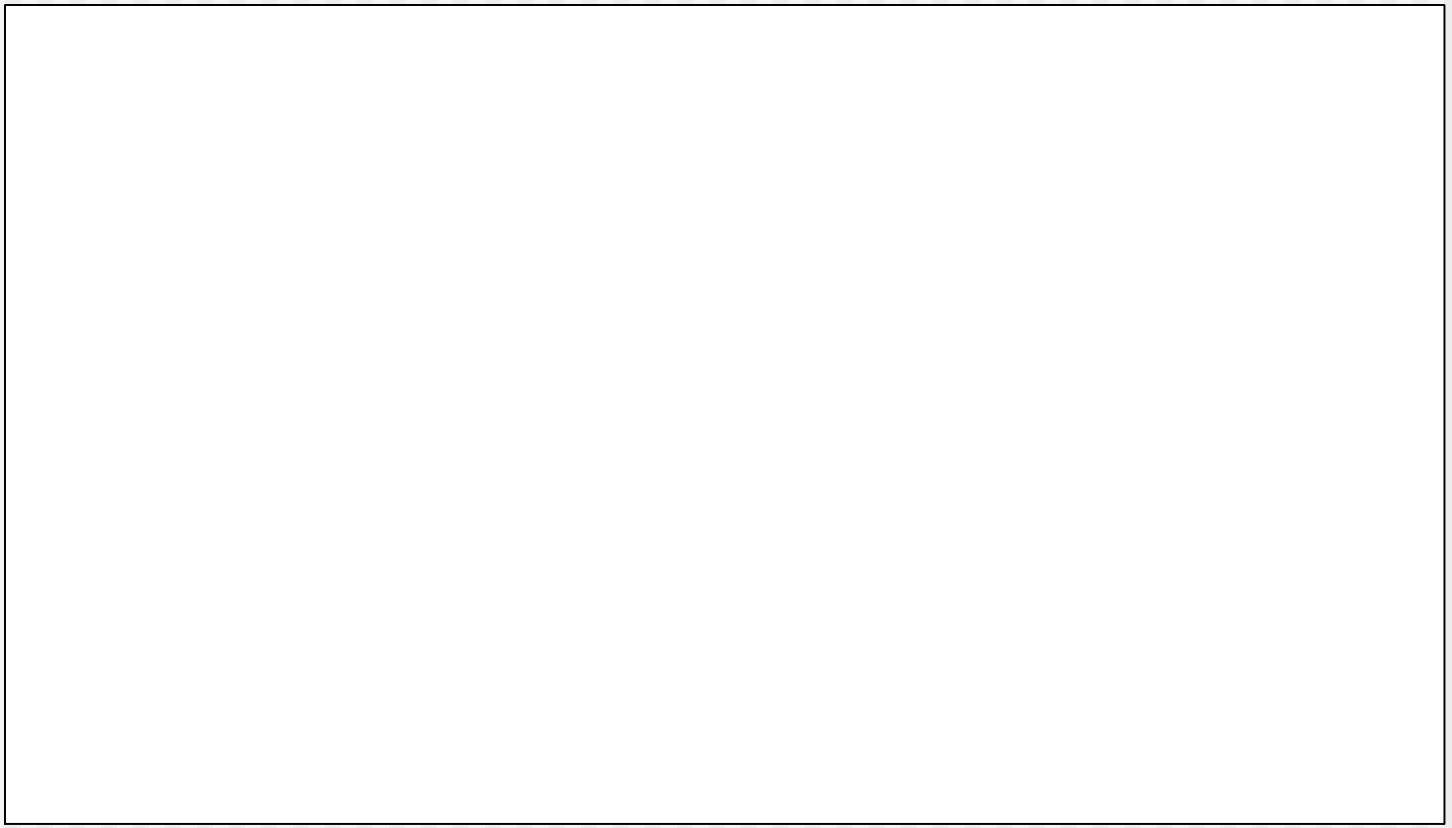
Exhaustive search - TSP

- Permutacja dopuszczalna, np. nie
  1-2-3-1-4
- Liczba permutacji – n!
- Rekurencyjne generowanie permutacji
  - ustalenie pierwszej pozycji
  - generowanie (n-1)! permutacji dla ustalonej pierwszej pozycji ...

Branch & Bound



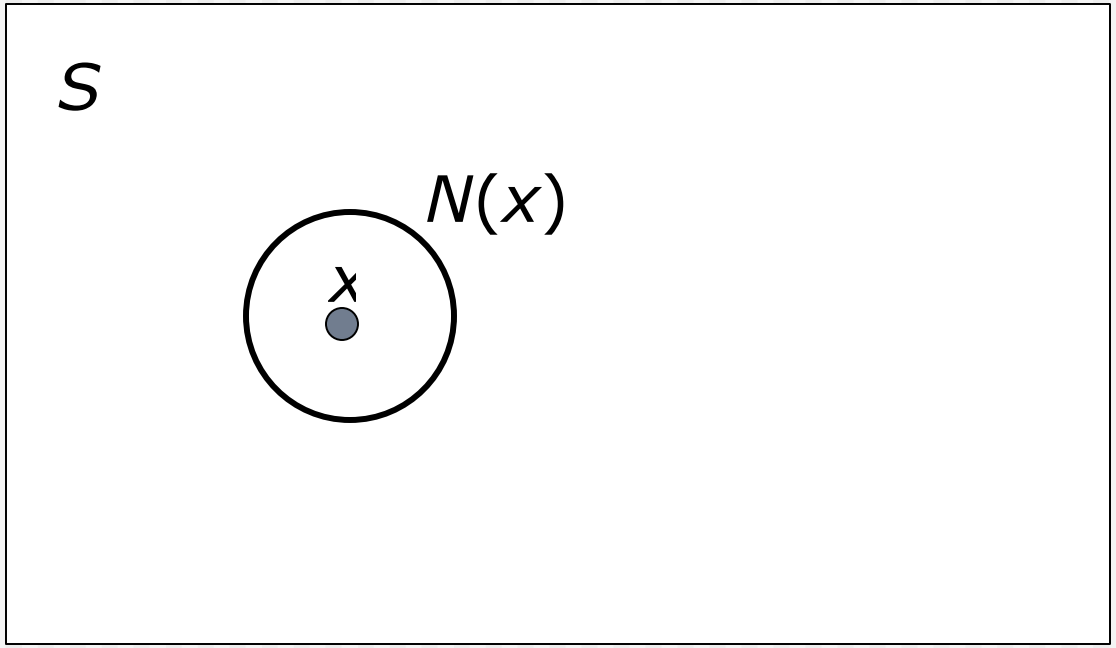
Przeszukiwanie losowe



Modelowanie i metaheurystyki

Przeszukiwanie lokalne

Idea sąsiedztwa



Definicja sąsiedztwa

- $x \in S$
- zbiór $N(x) \subseteq$ S rozwiązań, które leżą „blisko” rozwiązania x
- funkcja odległości

$$dist: S \times S \rightarrow \boldsymbol{R}$$

- sąsiedztwo

$$N(x) = \{y \in S : dist(x, y) \leq \varepsilon\}$$

- każde rozwiązanie $y \in N(x)$ jest nazywane rozwiązaniem sąsiednim lub po prostu sąsiadem x

Definicja sąsiedztwa (2)

- zakładamy, że $y \in N(x) \Leftrightarrow x \in N(y)$
- $N(\mathbf{x})$ można uzyskać z **x** wykonując jeden ruch (modyfikację) m z pewnego zbioru możliwych ruchów $M(\mathbf{x})$
- ruch m jest pewną transformacją, która zastosowana do rozwiązania **x** daje rozwiązanie **y**
- Sąsiedztwo można więc zdefiniować następująco:

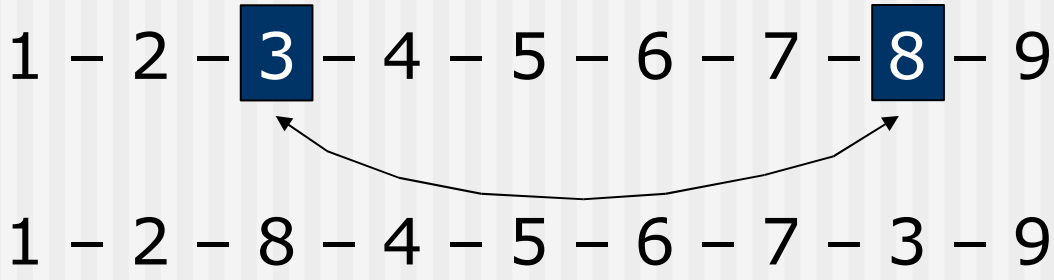
$$N(\mathbf{x}) = \{\mathbf{y} \mid \exists m \in M(\mathbf{x}) \mid \mathbf{y} = m(\mathbf{x})\}$$

Cechy sąsiedztwa

- ograniczenie na rozmiar
  - dla każdego x jego $N(x)$ zawiera co najmniej jedno rozwiązanie y rożne od x
  - nie może obejmować całej przestrzeni rozwiązań dopuszczalnych (nie może być dokładna)
- podobieństwo sąsiadów
  - $y \in N(x)$ niewiele różni się od x, tak by przejście (elementarny ruch) od x do y nie wymagało za każdym razem konstruowania nowego rozwiązania „od podstaw”
- równouprawnienie
  - niezależnie od wyboru rozwiązania początkowego powinno być osiągalne każde rozwiązanie należące do S

Przykład sąsiedztwa - TSP

- *k*-zamiana (ang. *k*-swap, *k*-opt)
  - *N*(*x*) – zbiór rozwiązań powstałych przez usunięcie *k* miast i wstawienie ich w innej kolejności
- 2-zamiana, 2-opt



$$|N(x)| = n(n\text{-}1)/2$$

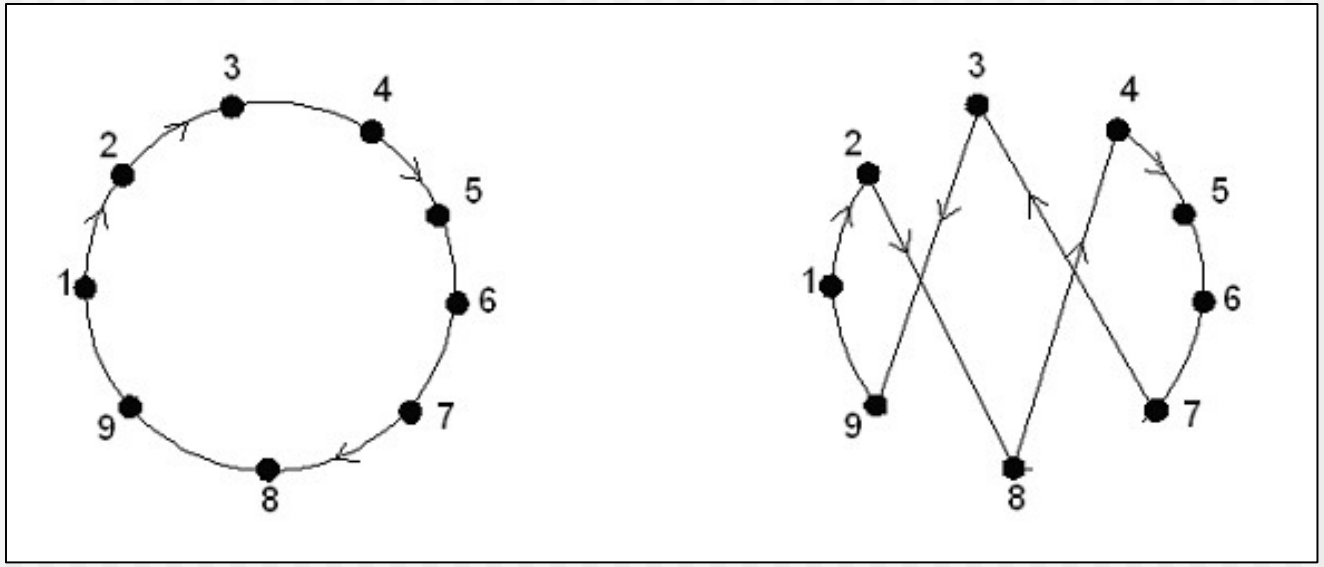
Sąsiedztwo TSP

wymiana miast

$N_2(3, 8)$

1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9

1 – 2 – 8 – 4 – 5 – 6 – 7 – 3 – 9

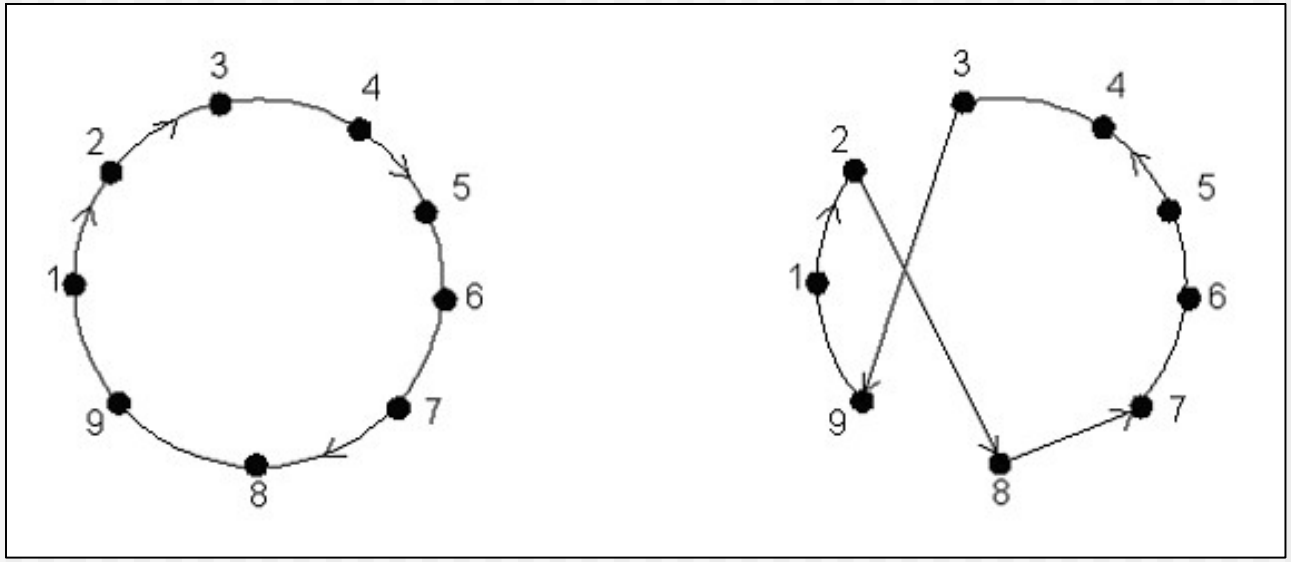


Sąsiedztwo TSP
wymiana łuków

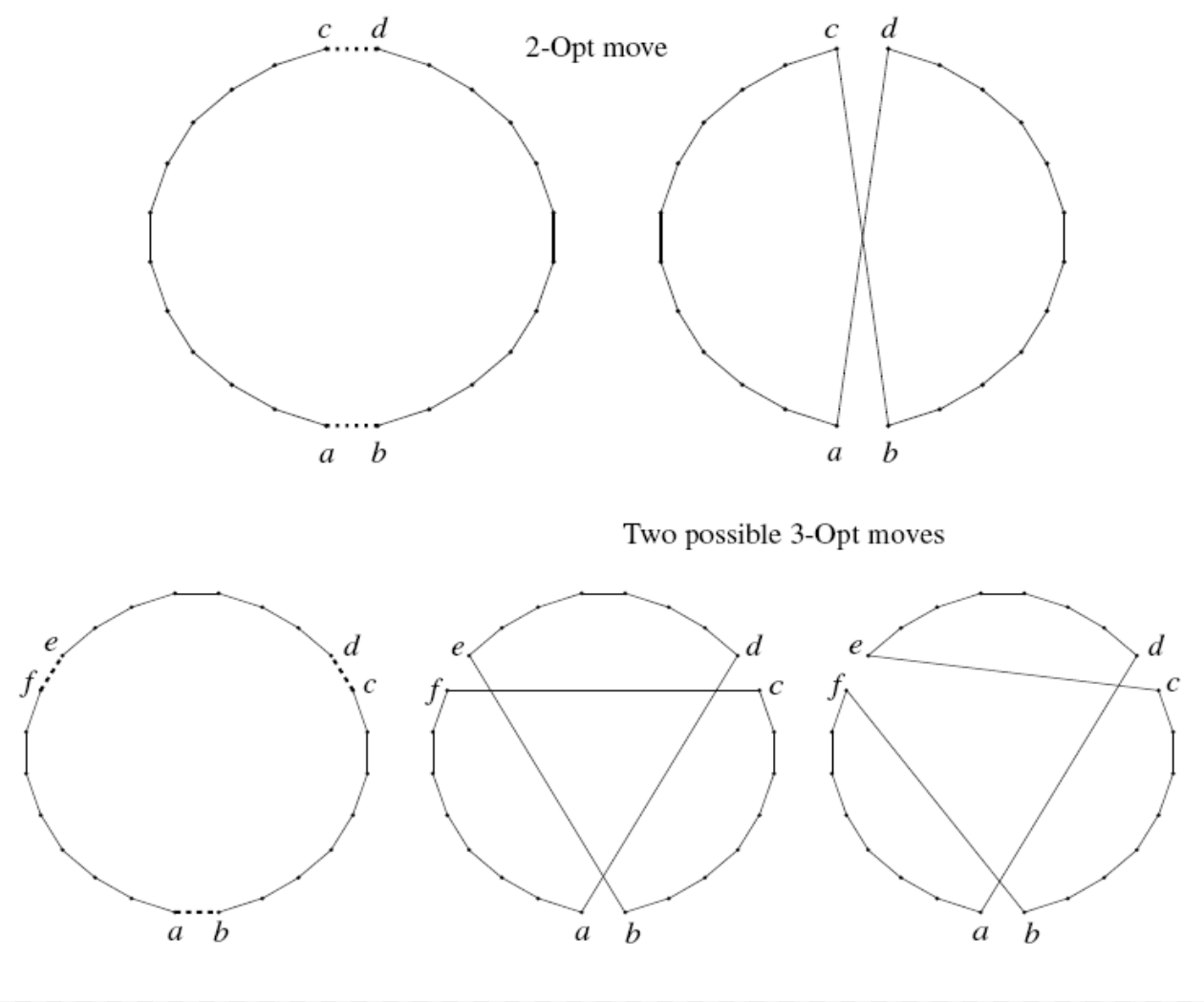
$N_2(3, 8)$

1 – 2 – **3 – 4 – 5 – 6 – 7 – 8** – 9

1 – 2 – **8 – 7 – 6 – 5 – 4 – 3** – 9

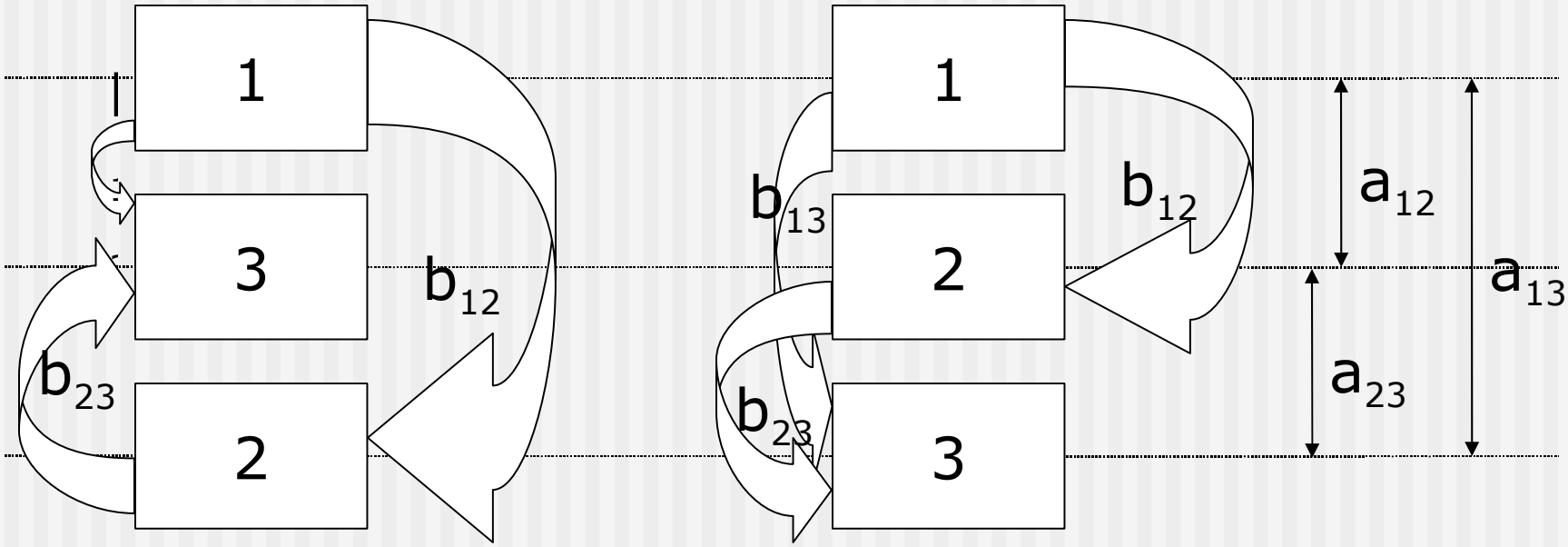


2-opt i 3-opt

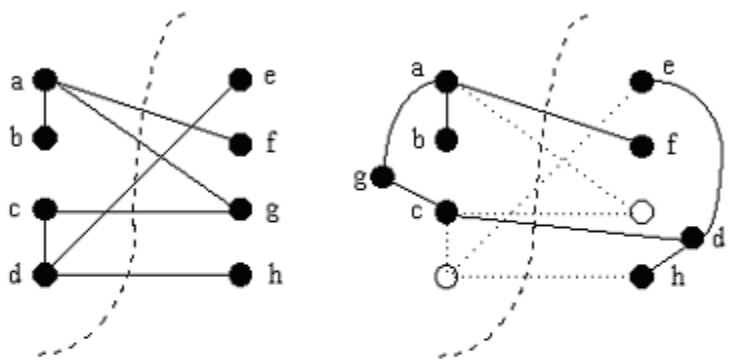

c d
2-Opt move
c d
a b
a b
Two possible 3-Opt moves
e d
f c
a b
e d
f c
a b
e d
f c
a b

Sąsiedztwo QAP

- zamiana lokalizacji
- 1-2-3 $\rightarrow$ 1-3-2



Sąsiedztwo GPP



Step No.	Vertex Pair	Gain	Cutset
0		0	5
1	(d, g)	3	2
2	(c, f)	1	1
3	(b, h)	-2	3
4	(a, e)	-2	5

Podejście alternatywne - generowanie rozwiązań niedopuszczalnych $|V_1| \neq |V_2|$.

$$F(V_1, V_2) = \sum_{i \in V_1, j \in V_2} E_{ij} + \gamma(|V_1| - |V_2|)^2$$

γ - dodatnia stała (kara za niedopuszczalny podział).

Sąsiedztwo – wszystkie takie podziały (V'_1, V'_2), że

$V'_1 = V_1 \cup \{x\}$ i $V'_2 = V_2 \setminus \{x\}$ lub
$V'_1 = V_1 \setminus \{y\}$ i $V'_2 = V_2 \cup \{y\}$,

$x \in V_2, y \in V_1$.

Przeszukiwanie lokalne

Eksploracja sąsiedztwa x

- Wybierz rozwiązanie w *S* i oceń je, zdefiniuj jako rozwiązanie *bieżące*
- Dokonaj generacji nowego rozwiązania z rozwiązania bieżącego i oceń je
- Jeśli nowe rozwiązanie jest lepsze zdefiniuj jako rozwiązanie bieżące, w przeciwnym wypadku odrzuć
- Powtarzaj kroki 2 i 3 dopóki można uzyskać poprawę

Lokalne optimum

x_{min} jest *lokalnym minimum* jeśli

$$f(x_{min}) \leq f(j), \text{ dla wszystkich } j \in N(x_{min})$$

x_{max} jest *lokalnym maksimum* jeśli

$$f(x_{max}) \geq f(j), \text{ dla wszystkich } j \in N(x_{max})$$

Lokalne przeszukiwanie

Wersja zachłanna

Wygeneruj rozwiązanie **x**

powtarzaj

dla każdego y$\in N(\mathbf{x})$ w losowej kolejności

jeżeli $f(\mathbf{y}) > f(\mathbf{x})$ **to**

$\mathbf{x} := \mathbf{y}$

dopóki nie znaleziono lepszego rozwiązania

Wersja stroma

Wygeneruj rozwiązanie **x**

powtarzaj

znajdź najlepsze rozwiązanie $\mathbf{y} \in N(\mathbf{x})$

jeżeli $f(\mathbf{y}) > f(\mathbf{x})$ **to**

$\mathbf{x} := \mathbf{y}$

dopóki nie znaleziono lepszego rozwiązania

Efektywność lokalnego przeszukiwania

- więcej rozwiązań jest ocenianych niż akceptowanych
  - wersja stroma
  - wersja zachłanna
- **ocena rozwiązań sąsiednich jest operacją krytyczną** z punktu widzenia efektywności algorytmu

Efektywność lokalnego przeszukiwania

- Bezpośrednia implementacja algorytmów LS okazuje się dla wielu typowych problemów bardzo nieefektywna
- Rozwiązania należące do sąsiedztwa są jawnie konstruowane przed ich oceną – kosztowna operacja
- Większość ocenionych rozwiązań nie zostaje akceptowana - nie warto ich jawnie konstruować, wystarczy znać ich wartość f. celu

Efektywność lokalnego przeszukiwania

- Czy można obliczyć wartość f. celu bez konstrukcji rozwiązania?

$$\Delta f_m(\mathbf{x})$$

Efektywne przeglądanie sąsiedztwa

Wersja stroma

Wygeneruj rozwiązanie **x**

powtarzaj

znajdź najlepszy ruch **m** ∈ *M*(**x**)

jeżeli *f*(m(**x**)) > *f*(**x**)

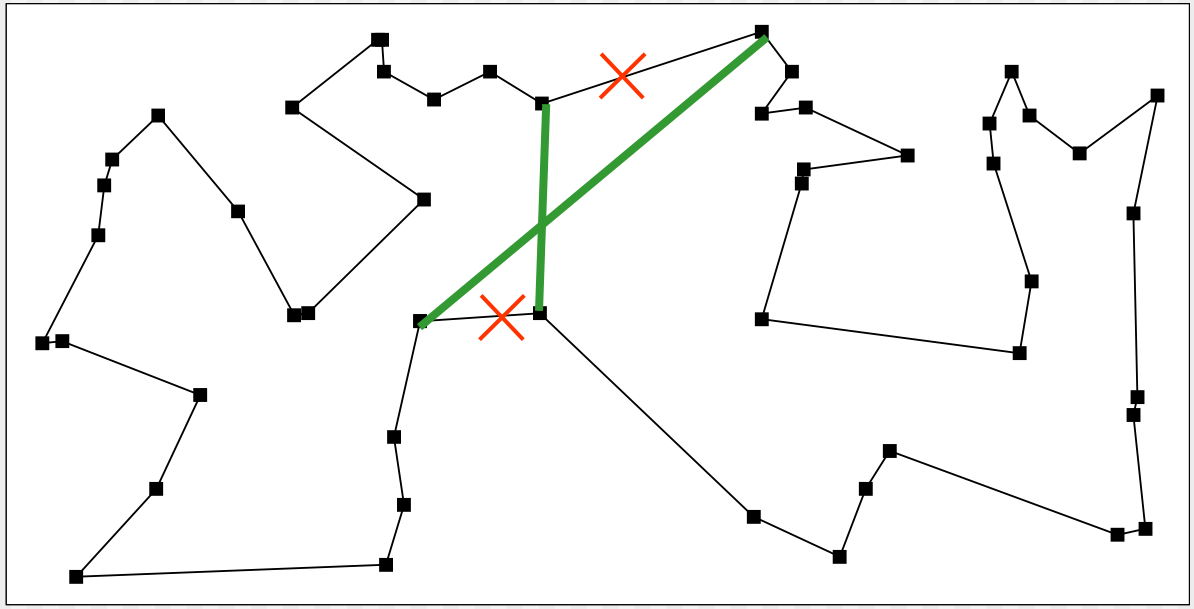
to

x := m(**x**)

dopóki nie znaleziono lepszego rozwiązania

- **Ocenia się ruchy, nie rozwiązania!**
- **Rozwiązania sąsiednie nie muszą być jawnie konstruowane!**

Ocena ruchów –problem komiwojażera (TSP)



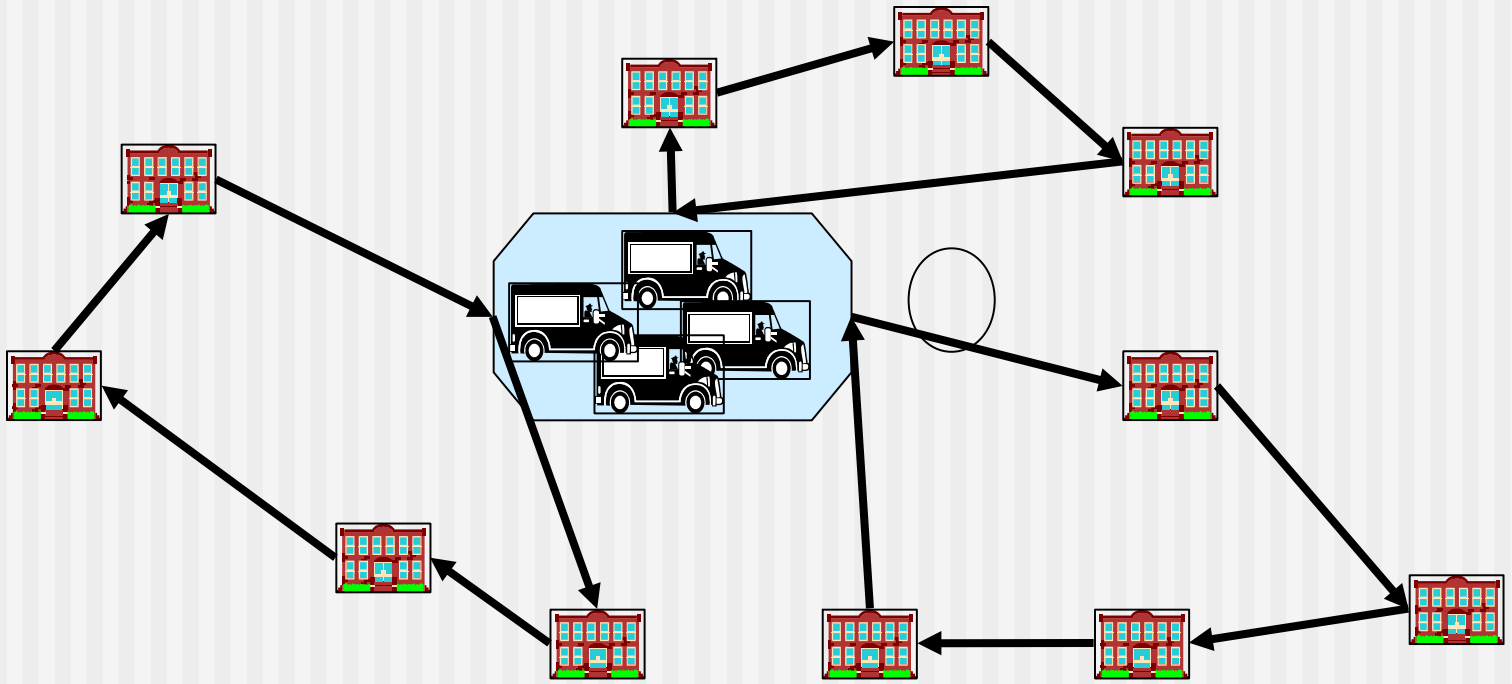
Ocena ruchu:

Różnica kosztów dwóch dodawanych i dwóch usuwanych łuków

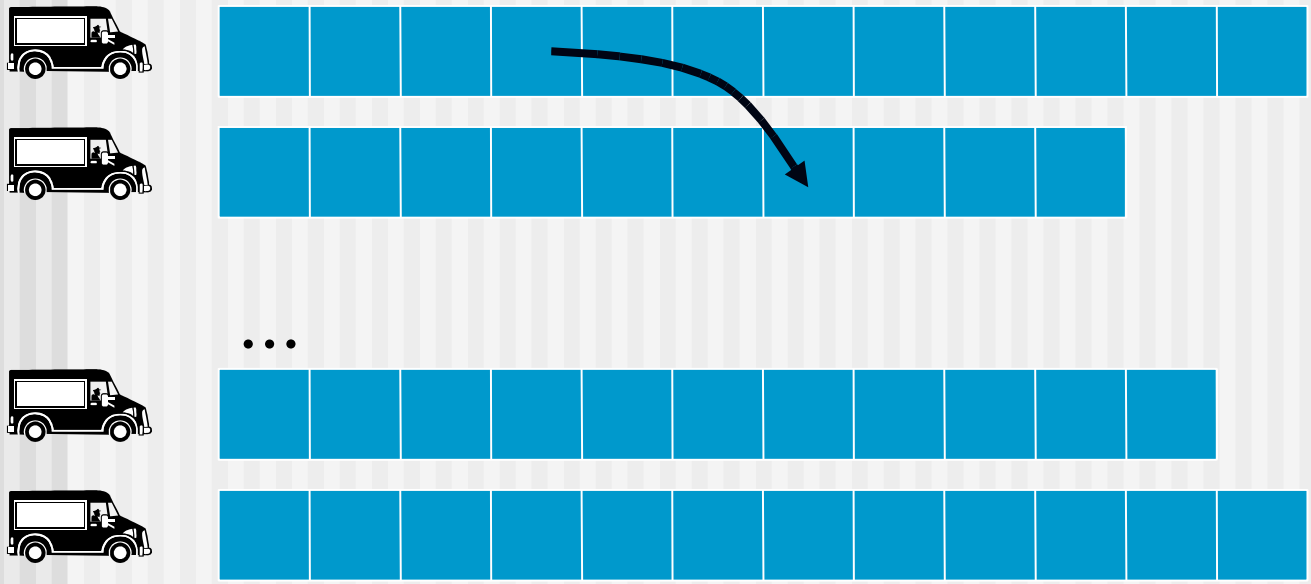
Efektywne obliczanie kosztu zamiany - TSP

Problem marszrutyzacji pojazdów – (*VRP*)

- Należy odwiedzić wszystkie wierzchołki korzystając ze zbioru pojazdów



Ocena ruchów – vehicle routing problem (VRP)



- Należy przeliczyć koszty tylko dwóch tras

Efektywne obliczanie kosztu zamiany - VRP

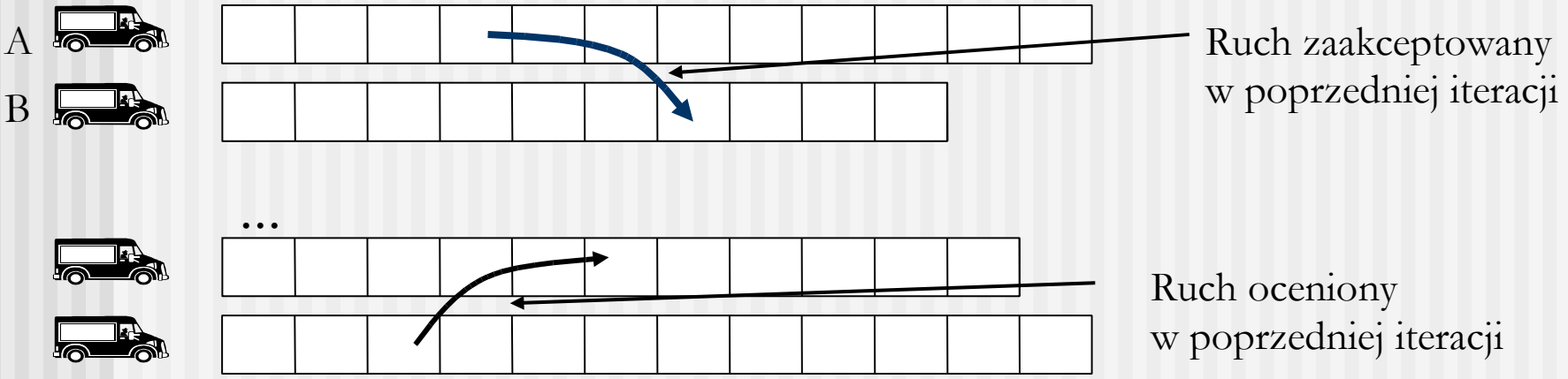
Efektywne ocenianie ruchów

- Większość ruchów jest tylko oceniana, a nie wykonywana
- Ocena ruchu wymaga zmodyfikowania dwóch tras
- Nakład na ocenienie ruchu – *2*
- Złożoność jednej iteracji – 2 $W(T-1)$

Wykorzystanie ocen ruchów z poprzednich iteracji

- Dla $\mathbf{y} \in N(\mathbf{x})$ z reguły $N(\mathbf{x}) \cap N(\mathbf{y}) = \emptyset$ lub
  $|N(\mathbf{x}) \cap N(\mathbf{y})| << |N(\mathbf{x})|$
- Ale
  $M(\mathbf{x}) \cap M(\mathbf{y})$ może być liczny tzn. wiele ruchów pozostaje takich samych po wykonaniu ruchu

Wykorzystanie ocen ruchów, z poprzednich iteracji



- Oceny wszystkich ruchów **nie dotyczących tras A i B** pozostają te same

Wykorzystanie ocen ruchów, z poprzednich iteracji

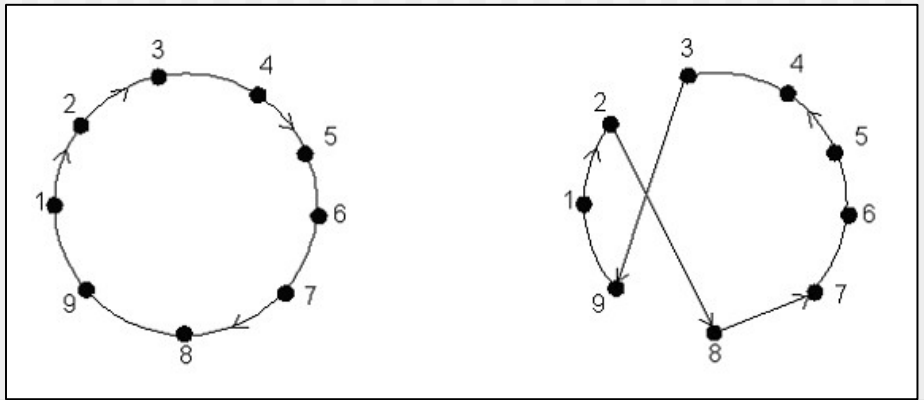
- W wersji zachłannej redukcja jest mniejsza ponieważ nie wszystkie ruchy musiały być przejrzane w poprzedniej iteracji
- W wersji zachłannej można pomijać ruchy ocenione w poprzedniej iteracji gdyż wiadomo, że nie przynoszą one poprawy

Pomijanie złych ruchów na podstawie reguł heurystycznych

- Technika znana także pod nazwą *candidate moves* – ruchy kandydackie
- W sąsiedztwie ocenia się ruchy kandydackie
- Pozostałe ruchy pomija się lub ocenia się z niewielkim prawdopodobieństwem

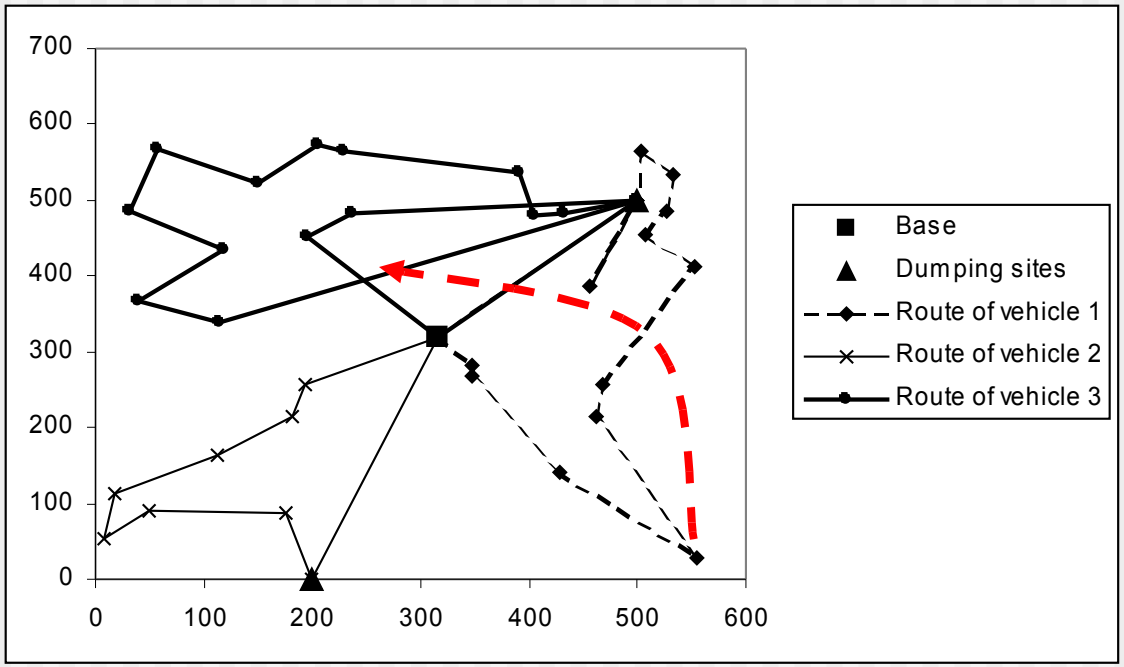
Pomijanie złych ruchów na podstawie reguł heurystycznych

- TSP – dla każdego wierzchołka tworzymy listę *N′* << *N* najbliższych wierzchołków
- Ruchy kandydackie wprowadzają tylko łuki prowadzące do najbliższych wierzchołków



Pomijanie złych ruchów na podstawie reguł heurystycznych

- W VRP można rozważać tylko ruchy pomiędzy blisko położonymi trasami



Pomijanie złych ruchów na podstawie reguł heurystycznych

- Ruchy złe i kandydackie mogą być identyfikowane na podstawie działania metody wyższego rzędu
- Pamięć długoterminowa w
  - Lista dobrych łuków w TSP
  - Lista łuków występujących w ruchach przynoszących poprawę
- Wykorzystanie informacji z operatorów rekombinacji
  - Np. do ruchów kandydackich zalicza się ruchy wprowadzające łuki występujące o dowolnego z rodziców (nawet jeżeli nie znalazły się one w potomku)

Techniki zaawansowane

- Np. przeglądanie sąsiedztwa o rozmiarze wykładniczym w czasie wielomianowym

```
Wygeneruj rozwiązanie x
powtarzaj
    znajdź najlepszy ruch m∈ M(x)
        jeżeli f(m(x)) > f(x) to
            x := m(x)
dopóki nie znaleziono lepszego
    rozwiązania
```

Problem optymalizacji

Wady

- kończą działanie w optimum lokalnym (poza przypadkiem dokładnej struktury sąsiedztwa)
  - dokładne sąsiedztwa są w większości przypadków nie praktyczne ponieważ sprowadzają się do kompletnego przeszukania przestrzeni rozwiązań dopuszczalnych,
- jakość rozwiązań w dużej mierze zależy od wyboru rozwiązania startowego
  - dla większości problemów nie ma żadnych wskazań do tego jak odpowiednio wybrać rozwiązanie startowe

Zalety

- są elastyczne
- można stosować dla każdego problemu optymalizacji kombinatorycznej

Unikanie wad algorytmów lokalnego przeszukiwania

- wprowadzenie bardziej złożonej definicji sąsiedztwa w celu możliwości przeszukania większej części przestrzeni rozwiązań dopuszczalnych
- ograniczone akceptowanie pogorszeń funkcji celu
- wykonanie algorytmu lokalnego przeszukiwania dla dużej liczby rozwiązań startowych
- dobre rozwiązania startowe

Dobre rozwiązania startowe
Heurystyki

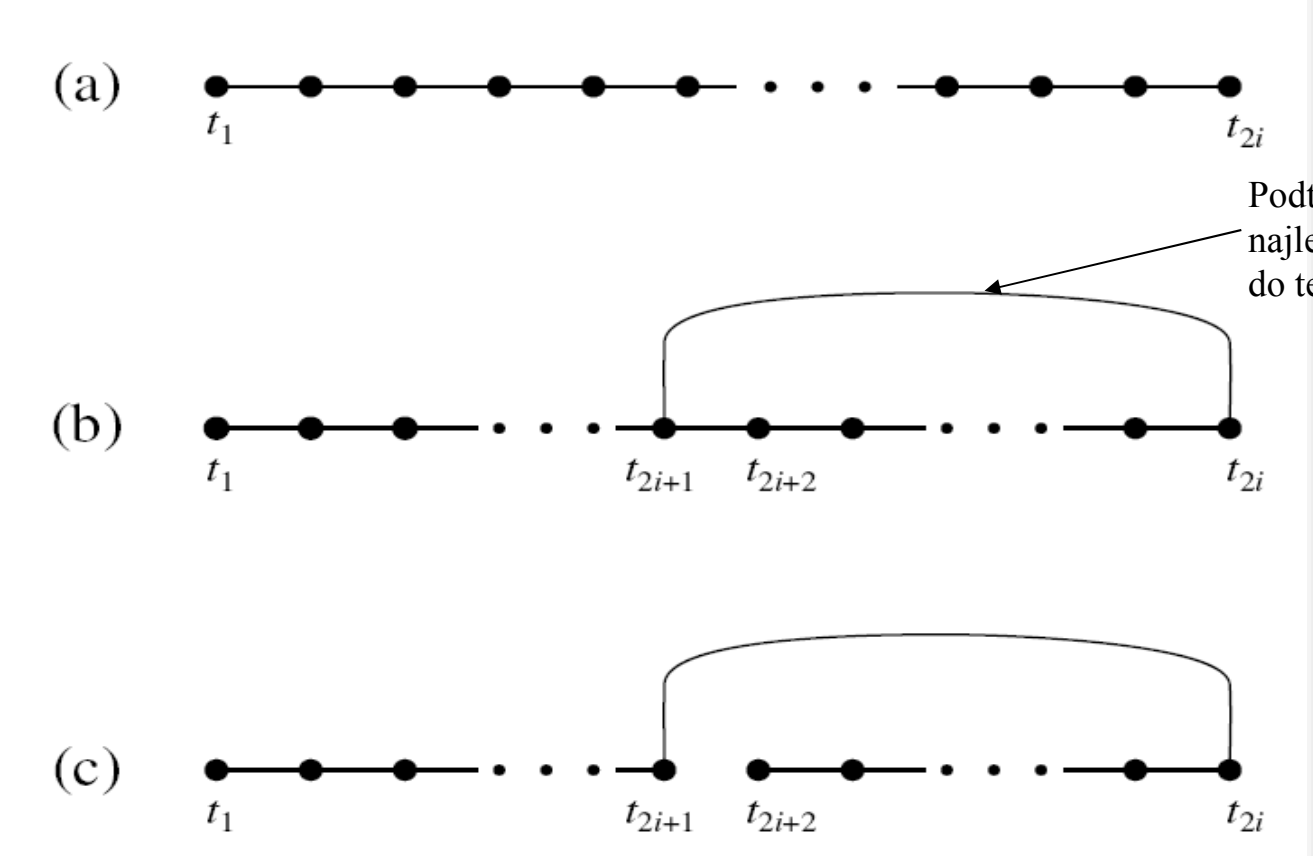
- NN (Nearest Neighbor)
  - Iteracyjne dodawanie do trasy najbliżej leżącego miasta.
- Greedy
  - Trasa jest budowana tak, że zawsze tworzy cykl Hamiltona. W każdej iteracji dodawany jest jeden najkrótszy łuk z pozostałych dostępnych.
- Clarke-Wright
  - Na początku każde miasto połączone z bazą, potem iteracyjna eliminacja takiego połączenia z bazą (przejazd bezpośrednio do miasta), która da największe oszczędności

Algorytm Lina-Kernighana
LK search

- Od 1973 do 1989 najlepszy algorytm, i ciągle w czołówce
- Nowa implementacja LK rozwiązuje w 50 min instancje z 1 000 000 miast
- Bazuje na ruchach 2-opt z restrykcjami

Algorytm Lina-Kernighana
LK search

- Ścieżka Hamiltona (nie cykl)



Lokalne przeszukiwanie

Porównanie wersji zachłannej i stromej

- Podobieństwa
  - Przeglądają N(x)
  - Dochodzą do lokalnego optimum
- Różnice
  - Stromy – szuka najlepszego
  - Zachłanny – pierwszy lepszy