Algorytmy ewolucyjne i metaheurystyczne

Andrzej Jaszkiewicz

Zakres przedmiotu

- Wprowadzenie
- Dokładne i heurystyczne algorytmy optymalizacji
- Przeszukiwanie lokalne
- Metaheurystyki oparte na lokalnym przeszukiwaniu
  - Symulowane wyżarzanie
  - Przeszukiwanie tabu
  - Iteracyjne przeszukiwanie lokalne
- Algorytmy populacyjne
  - Algorytmy kolonii mrówek
  - Algorytmy ewolucyjne
  - Hybrydowe algorytmy ewolucyjne
- Ograniczenia w algorytmach metaheurysycznych
- Podstawy teoretyczne i konstrukcja algorytmów metaheurystycznych
- Wielokryterialne algorytmy metaheurystyczne?

Sformułowanie problemu optymalizacji – myślenie w kategoriach celów (deklaratywne)

Myślenie regułowe (imperatywne)

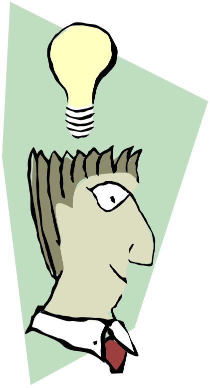
Optymalizacja

- Jakie są moje cele? Co chcę osiągnąć?
- Jakie decyzje mogę podjąć?
- Jakie ograniczenia muszę uwzględnić?
- Jak ocenić wpływ decyzji na cele?

Przykład – ustalanie lokalizacji towarów w zautomatyzowanym magazynie

Myślenie regułowe

Jeżeli towar jest często zamawiany, to należy go umieścić blisko obszaru kompletowania



Przykład – ustalanie lokalizacji towarów w zautomatyzowanym magazynie

- Jakie są moje cele? Co chcę osiągnąć?
- Średnia długość drogi podczas kompletowania zlecenia powinna być jak najkrótsza.

- Jakie decyzje mogę podjąć?
- Przydział towarów do lokalizacji w magazynie

- Jakie ograniczenia muszę uwzględnić?
- Przydział każdego produktu.
- Ograniczenia technologiczne

- Jak ocenić wpływ decyzji na cele?
- Funkcja nieliniowa służąca do obliczania średniej długości drogi

Elementy zagadnienia optymalizacji

- Jakie są moje cele?
  Co chcę osiągnąć?
- Jakie decyzje mogę podjąć?
- Jakie ograniczenia muszę uwzględnić?
- Jak ocenić wpływ decyzji na cele?

- Kryteria. Funkcje celu
- Zmienne decyzyjne
- Ograniczenia. Przestrzeń rozwiązań dopuszczalnych
- Sposób obliczania funkcji celu:
  - Analityczny
  - Symulacyjny

Przykład – ustalanie lokalizacji towarów w zautomatyzowanym magazynie

- Kryteria. Funkcje celu
- Zmienne decyzyjne
- Ograniczenia. Przestrzeń rozwiązań dopuszczalnych
- Sposób obliczania funkcji celu:
  - Analityczny
  - Symulacyjny

- Średnia długość drogi podczas kompletowania zlecenia.
- Przydział towarów do lokalizacji w magazynie
- Przydział każdego produktu.
- Ograniczenia technologiczne
- Funkcja nieliniowa

Współczesna optymalizacja a badania operacyjne

- Ogromny rozwój metod i narzędzi optymalizacji. Znacznie szerszy zakres zastosowania.
  - Badania operacyjne koncentrowały się na ciągłych problemach liniowych
- Integracja z rozwiązaniami informatycznymi, np. wykorzystanie danych
- Różnorodne związki ze sztuczną inteligencją w tym maszynowym uczeniem

Problem optymalizacji

minimalizuj/maksymalizuj $z = f(\mathbf{x})$

przy ograniczeniach (p.o)

$\mathbf{x} \in S$

Problem optymalizacji

minimalizuj $z = f(\mathbf{x})$

=

maksymalizuj $z' = -f(\mathbf{x})$

Problem programowania matematycznego

Jeżeli rozwiązanie jest zdefiniowane jako wektor zmiennych:

$$\mathbf{x} \in R^n, \text{ t.j. } \mathbf{x} = \{x_1, \ldots, x_\mathrm{n}\}$$

a ograniczenia jako zbiór równości/nierówności:

$$\mathbf{x} \in S \Leftrightarrow g_l(\mathbf{x}) \leq / \geq / = 0,\ j=1,\ldots,L$$

mamy do czynienia z problemem programowania matematycznego

Rodzaje problemów programowania matematycznego

- Jeżeli funkcja celu i ograniczenia są liniowe $\Rightarrow$ liniowy problem programowania matematycznego
- Jeżeli funkcja celu lub co najmniej jedno ograniczenie są nieliniowe $\Rightarrow$ nieliniowy problem programowania matematycznego

Rodzaje problemów programowania matematycznego

- Jeżeli zmienne są ciągłe $\Rightarrow$ ciągły problem programowania matematycznego
- Jeżeli zmienne są dyskretne $\Rightarrow$ dyskretny problem programowania matematycznego
- Jeżeli zmienne są mieszane ciągłe i dyskretne $\Rightarrow$ mieszany problem programowania matematycznego

Rodzaje problemów programowania matematycznego

- Jeżeli zmienne są całkowitoliczbowe $\Rightarrow$ całkowitoliczbowy problem programowania matematycznego
- Jeżeli zmienne są binarne $\Rightarrow$ binarny problem programowania matematycznego

Problem optymalizacji kombinatorycznej

- Dyskretny i skończony zbiór rozwiązań
- Struktura kombinatoryczna(?)

- Zawsze da się zdefiniować jako binarny/dyskretny problem programowania matematycznego…
- … ale nie zawsze warto

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

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

Rozwiązanie (globalnie) optymalne

W przypadku minimalizacji rozwiązanie $\mathbf{x}^{opt} \in S$ które spełnia

$$f(\mathbf{x}^{opt}) \leq f(\mathbf{x}) \text{ dla wszystkich } \mathbf{x} \in S$$

W przypadku maksymalizacji rozwiązanie $\mathbf{x}^{opt} \in S$ które spełnia

$$f(\mathbf{x}^{opt}) \geq f(\mathbf{x}) \text{ dla wszystkich } \mathbf{x} \in S$$

Klasyfikacja algorytmów optymalizacji

- algorytmy dokładne
  - przeszukiwanie wyczerpujące (exhaustive),
  - podziału i ograniczeń (B&B),
  - programowanie dynamiczne
- algorytmy heurystyczne
  - specjalizowane
  - metaheurystyczne:
    - losowego przeszukiwania (random search)
    - lokalnego przeszukiwania i odmiany
      - symulowane wyżarzanie
      - przeszukiwanie tabu
    - genetyczne/ewolucyjne
    - mrówkowe, świetlikowe…
    - hybrydy
    - …

Klasyfikacja algorytmów (2)

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

Dlaczego problemy mogą być trudne do rozwiązania

- Duża liczba możliwych rozwiązań *przeszukanie całej przestrzeni rozwiązań dopuszczalnych w celu znalezienia najlepszego jest nierealne*
- Problem jest złożony *(złożoność obliczeniowa),*
  użycie modeli uproszczonych i rezultaty są bezużyteczne
- Czasochłonne obliczanie f. celu *np. niezbędna symulacja*
- Funkcja oceny jest obarczona niepewnością
- Potencjalne rozwiązania mocno ograniczone
  znalezienie jednego dopuszczalnego jest problemem

Duża liczba rozwiązań

- Problem spełnienia wyrażenia logicznego (SAT)
  np. problem 100 zmiennych

$$F(x) = (x_{13} \vee \overline{x}_{23} \vee x_{34}) \wedge (\overline{x}_{13} \wedge x_{23} \vee \overline{x}_{34}) \wedge \ldots = TRUE$$

$$|S| = 2^{100} \approx 10^{30}$$

(dwie możliwości dla zmiennej, 100 zmiennych)

Duża liczba rozwiązań

- Problem komiwojażera (TSP – travelling salesperson problem)
- Liczba rozwiązań (n-1)! / 2

Liczba wierzchołków	Liczba rozwiązań
5	12
10	181440
20	6,08E+16
50	3,04E+62
100	4,67E+155

Modelowanie problemu

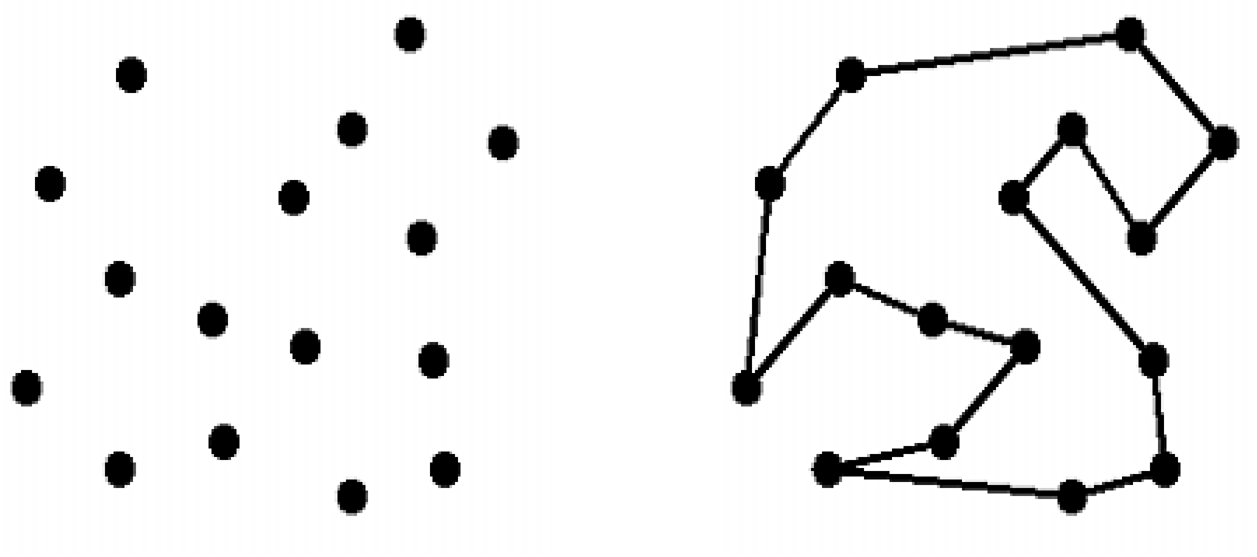
PROBLEM $\leftrightarrow$ MODEL $\leftrightarrow$ ROZWIĄZANIE

- Model - przybliżenie rzeczywistości
- Rozwiązanie $\rightarrow$ problem

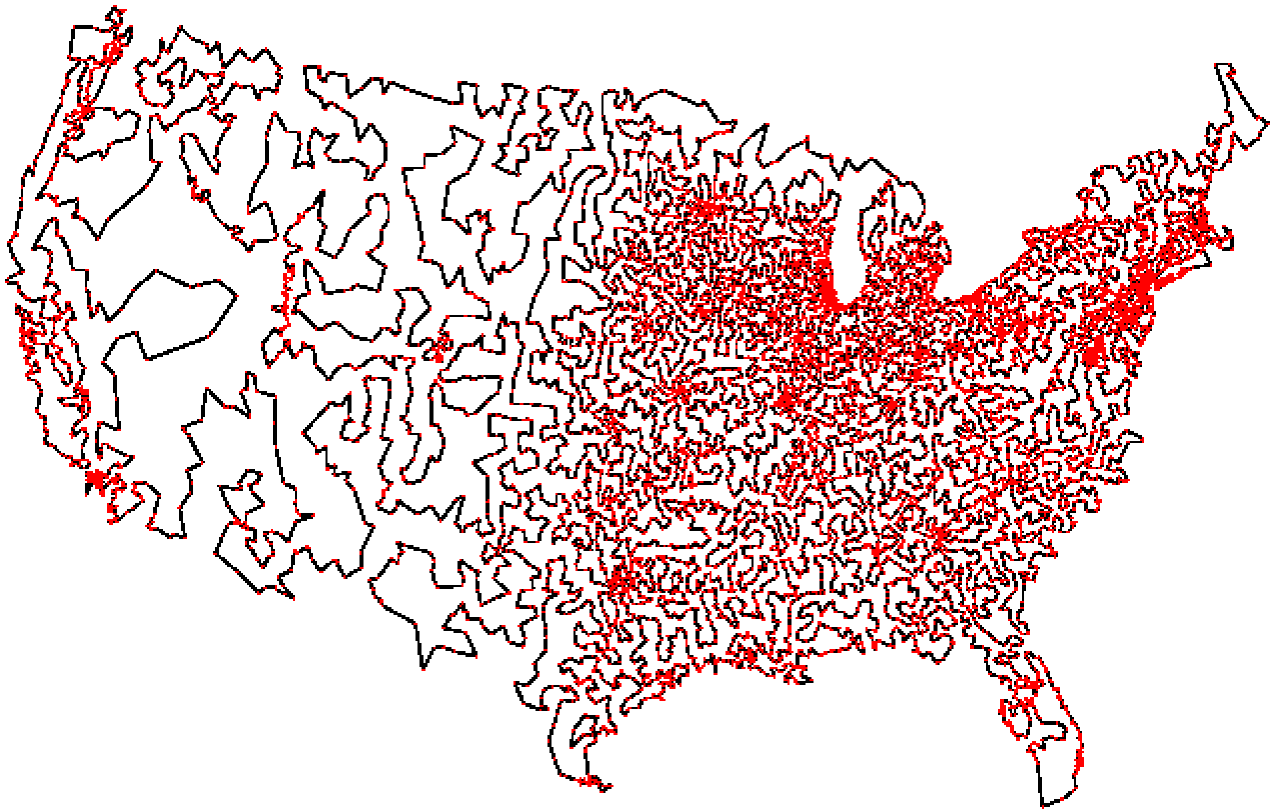
- Np. Problem transportowy z nieliniową i nieciągłą funkcją celu
- Dwa sposoby rozwiązania
  - uprościć model, żeby pasował do tradycyjnego modelu i dokładnej metody rozwiązania
  - wykorzystać podejście (meta)heurystyczne

Przykłady problemów kombinatorycznych

Problem komiwojażera (TSP)



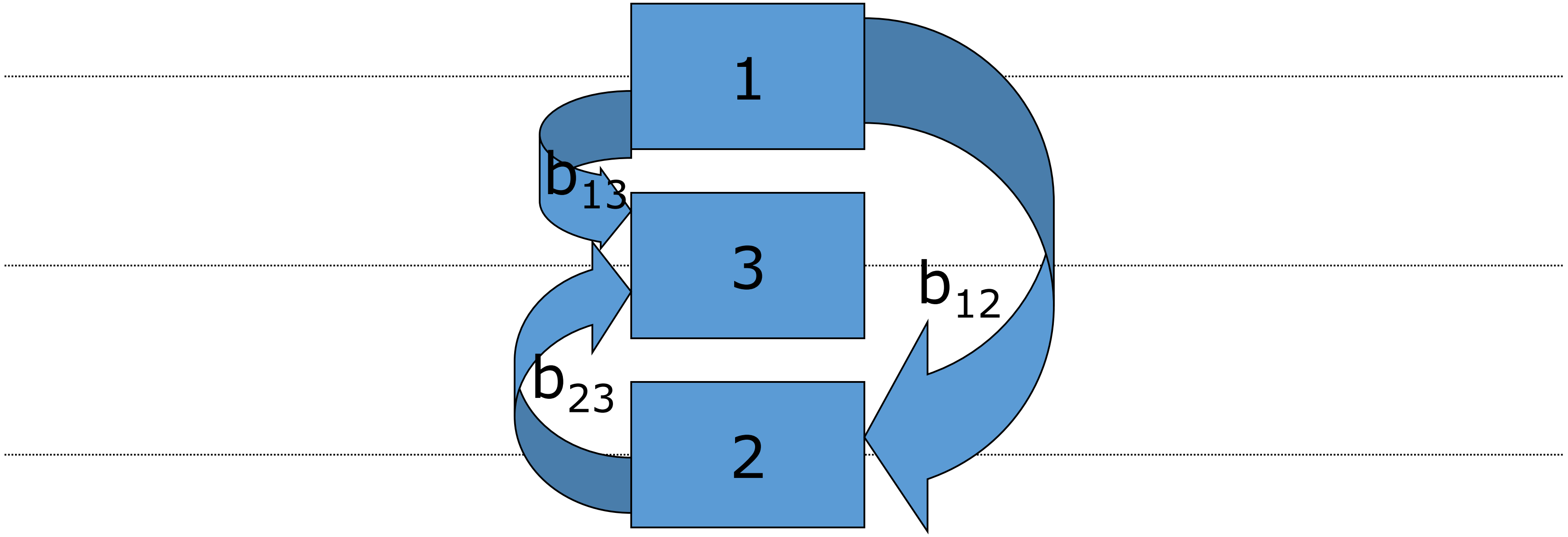
Problem komiwojażera (TSP)



Problem kwadratowego przydziału (QAP)

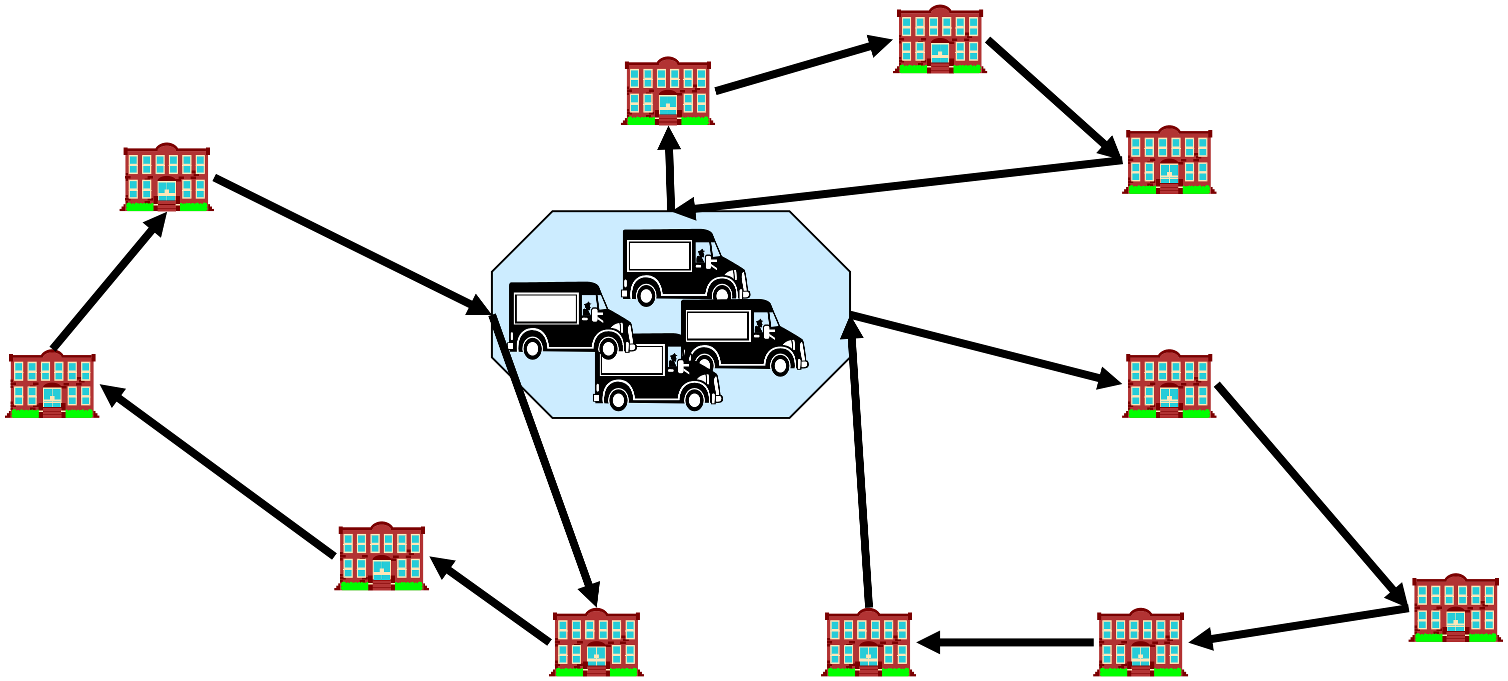
- Dane
  - Odległości pomiędzy możliwymi lokalizacjami
  - Przepływy pomiędzy czynnościami
- Przykład – lokalizacja personelu medycznego

Problem kwadratowego przydziału (QAP)



Problem marszrutyzacji pojazdów (VRP)

- Należy odwiedzić wszystkie wierzchołki korzystając ze zbioru pojazdów



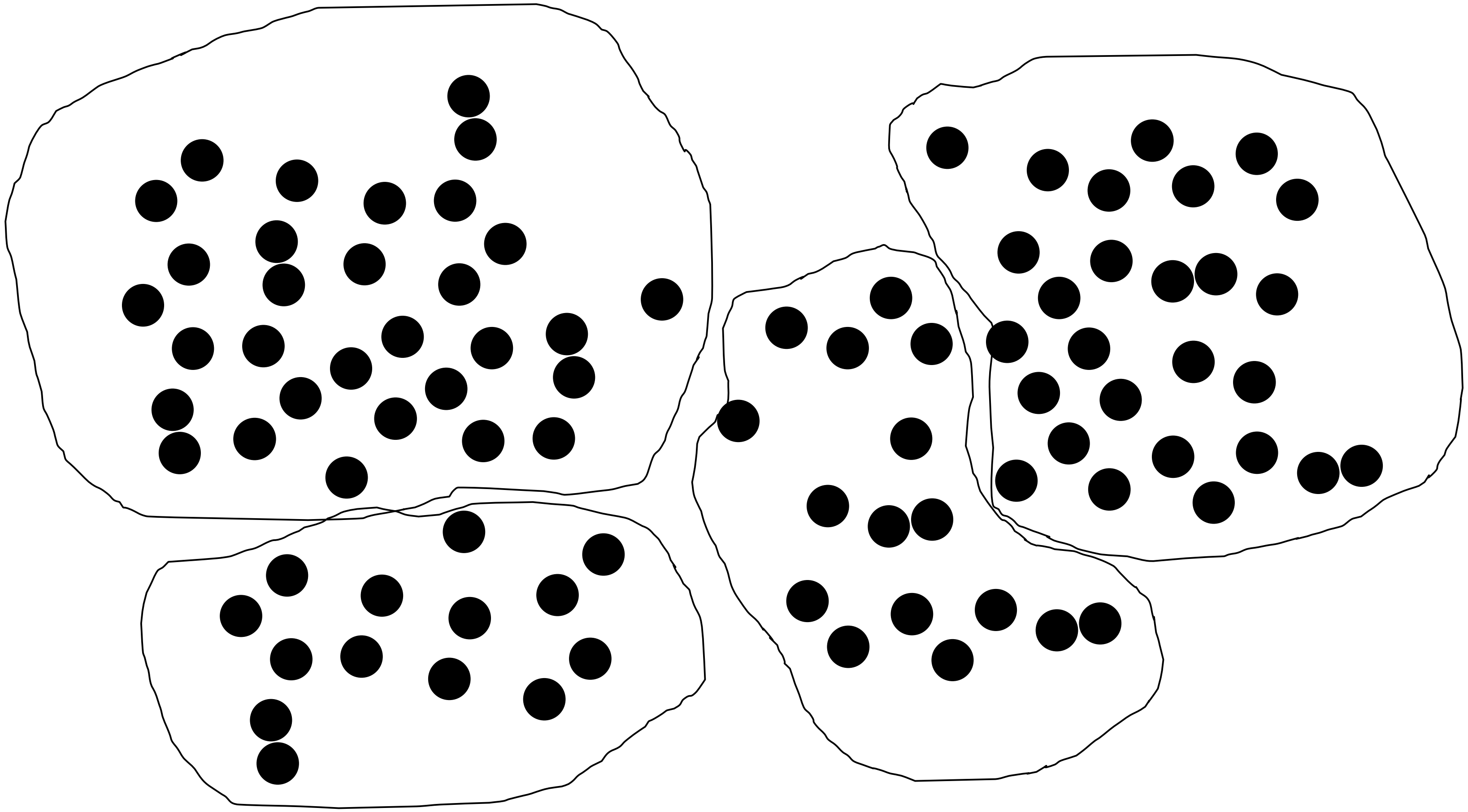
Problem plecakowy

- Dany jest zbiór elementów, z których każdy posiada wartość i wagę
- Do plecaka należy włożyć elementy o jak najwyższej łącznej wartości, których waga nie przekracza pojemności plecaka

Problem grupowania (clustering)

- Dany jest zbiór obiektów i macierz odległości/podobieństwa pomiędzy każdą parą elementów
- Obiekty należy podzielić na K grup średnią odległość wewnątrzgrupową

Grupowanie



Co jest potrzebne do zaprojektowania algorytmu

- Reprezentacja rozwiązania
- Funkcja oceny
- Cel (minimalizacja, maksymalizacja)

Reprezentacja rozwiązania

- Ciąg binarny
- Reprezentacja zmiennoprzecinkowa, np. programowanie nieliniowe
- Permutacja, np. TSP, QAP
- Zbiór (np. problem plecakowy)
- 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

0000 – 0

0001 – 1

0010 – 2

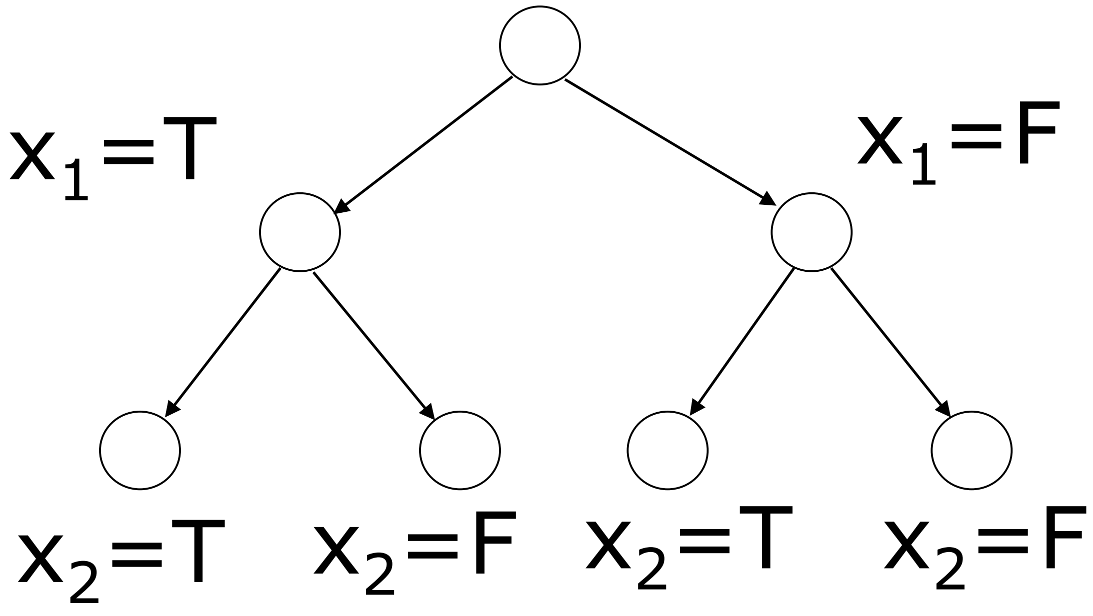
0011 – 3

...

Ale można inaczej dzieląc S. Jak?

Exhaustive search (2)

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



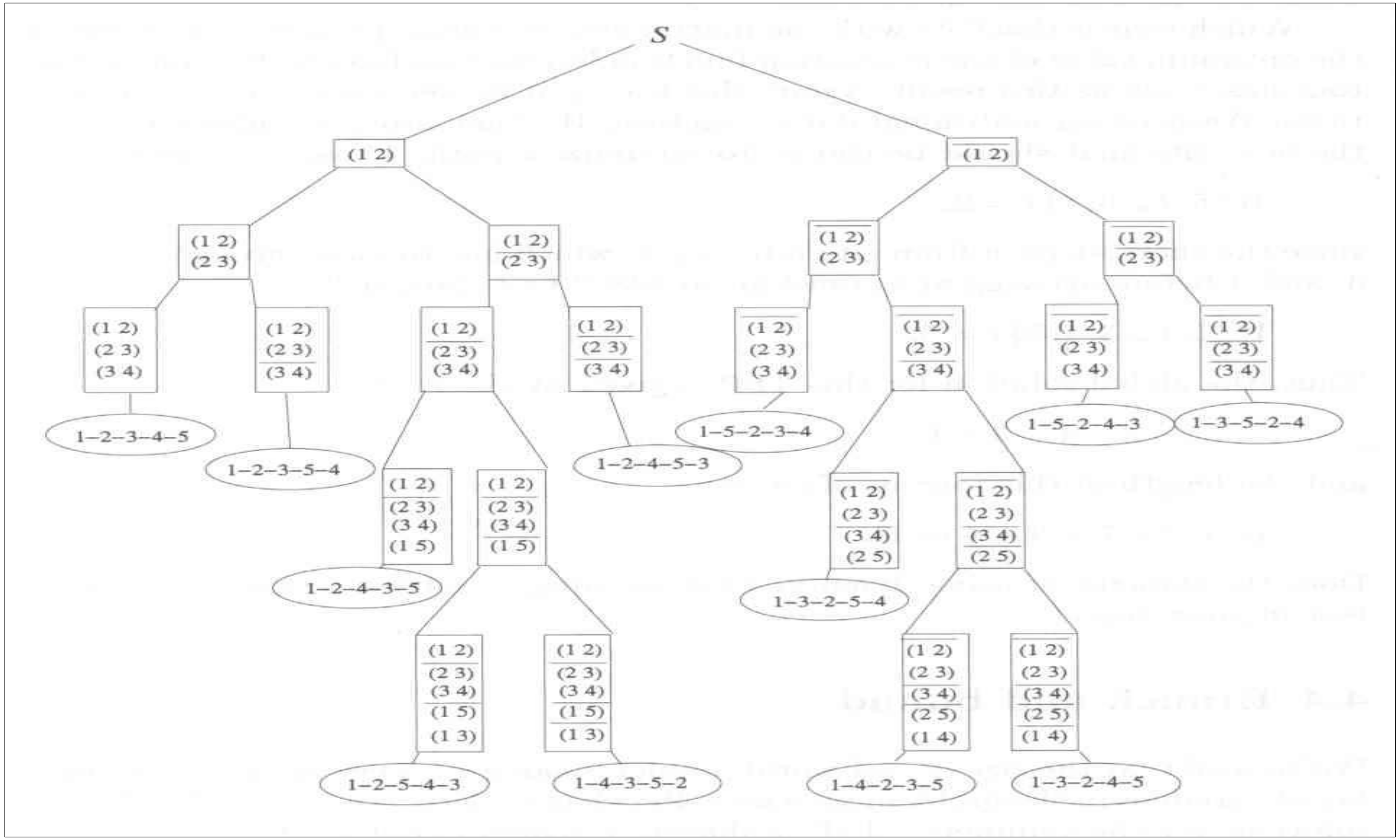
Exhaustive search - TSP

- 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



Złożoność black-box search

- Na komputerach klasycznych $O(|S|)$
- Na komputerach kwantowych przy zastosowaniu rozszerzeń algorytmu Grovera $\Theta(\sqrt{|S|})$
  - W czasie wielomianowym, gdyby mechanika kwantowa była choć nieznacznie nieliniowa (lub możliwy był bezpośredni pomiar stanu kwantowego)
  - Brak możliwości efektywnego rozwiązywania problemów NP-zupełnych jako prawo natury?
    - Daniel S. Abrams and Seth Lloyd. 1998. Nonlinear Quantum Mechanics Implies Polynomial-Time Solution for NP-Complete and # P Problems. Phys. Rev. Lett. 81 (Nov 1998), 3992–3995. Issue 18. https://doi.org/10.1103/PhysRevLett.81.3992
    - ScottAaronson. 2005. Guest Column: NP-Complete Problems and Physical Reality. SIGACT News 36, 1 (March 2005), 30–52. https://doi.org/10.1145/1052796.1052804

Przeszukiwanie losowe

Powtarzaj

Wygeneruj losowe rozwiązanie **x**

Do spełnienia warunków stopu

Zwróć najlepsze znalezione rozwiązanie

Heurystyki zachłanne (greedy)

- Konstrukcja rozwiązania poprzez dodawania kolejnych elementów np. wierzchołków (łuków/krawędzi)
- W każdym kroku dodawany jest jeden, najlepszy element

Heurystyki zachłanne (greedy)

Utwórz rozwiązanie początkowe, np. **x** := Ø

powtarzaj

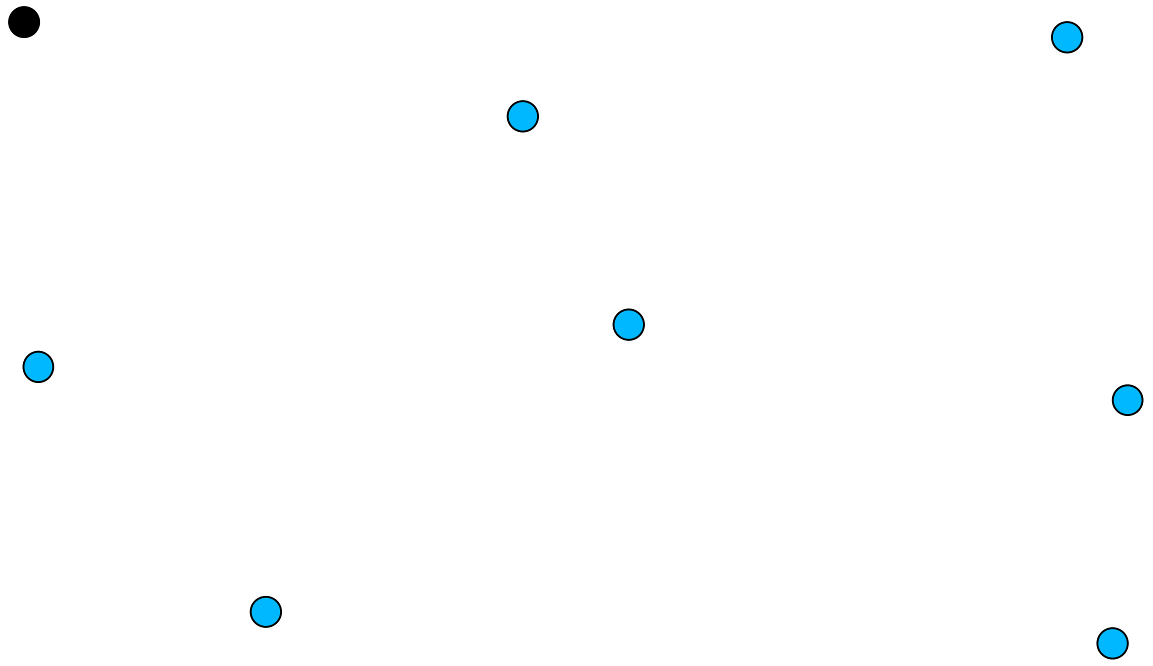
  dodaj do **x** lokalnie najlepszy element nie wchodzący jeszcze w skład rozwiązania

dopóki nie utworzono pełnego rozwiązania

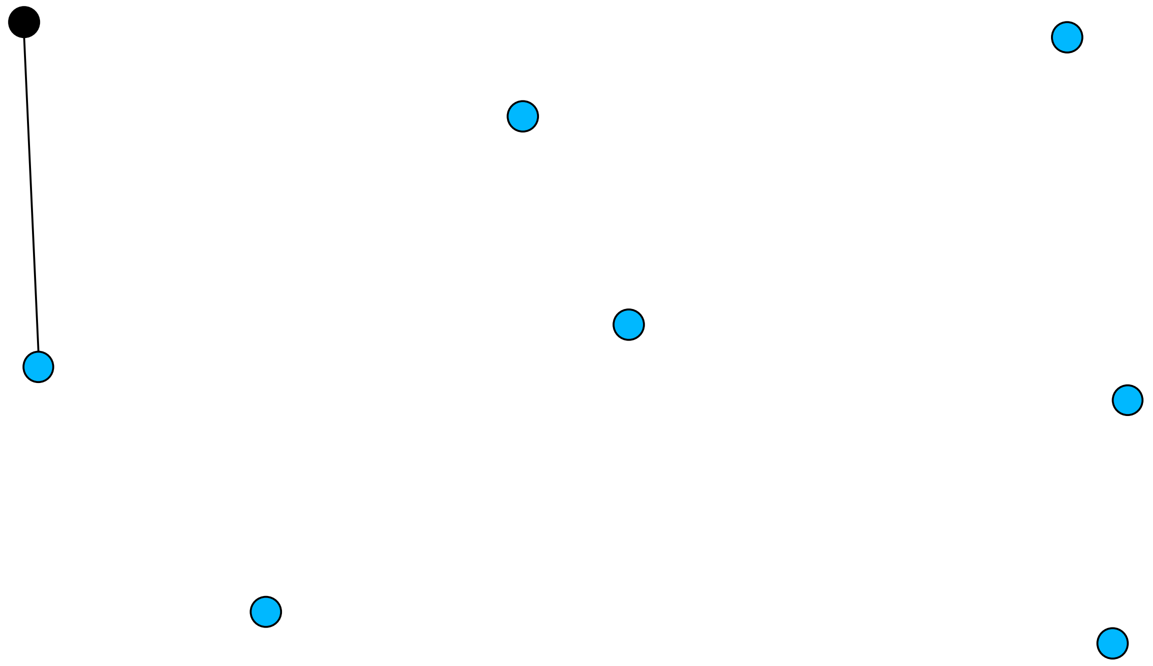
Przykłady heurystyk zachłannych

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

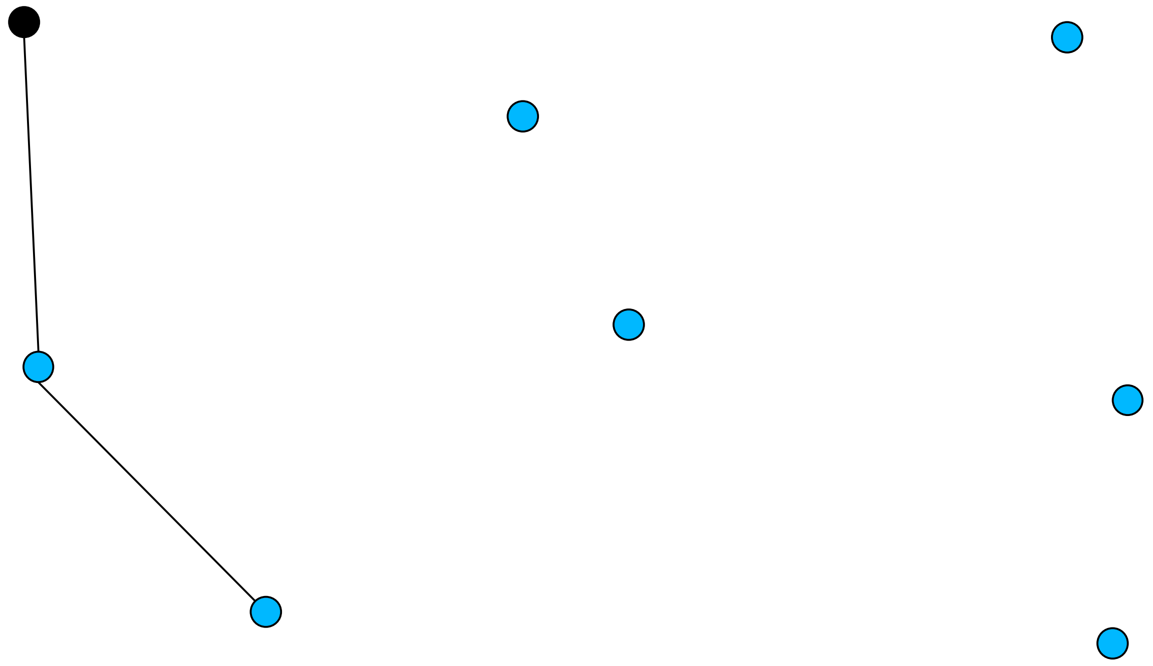
NN przykład



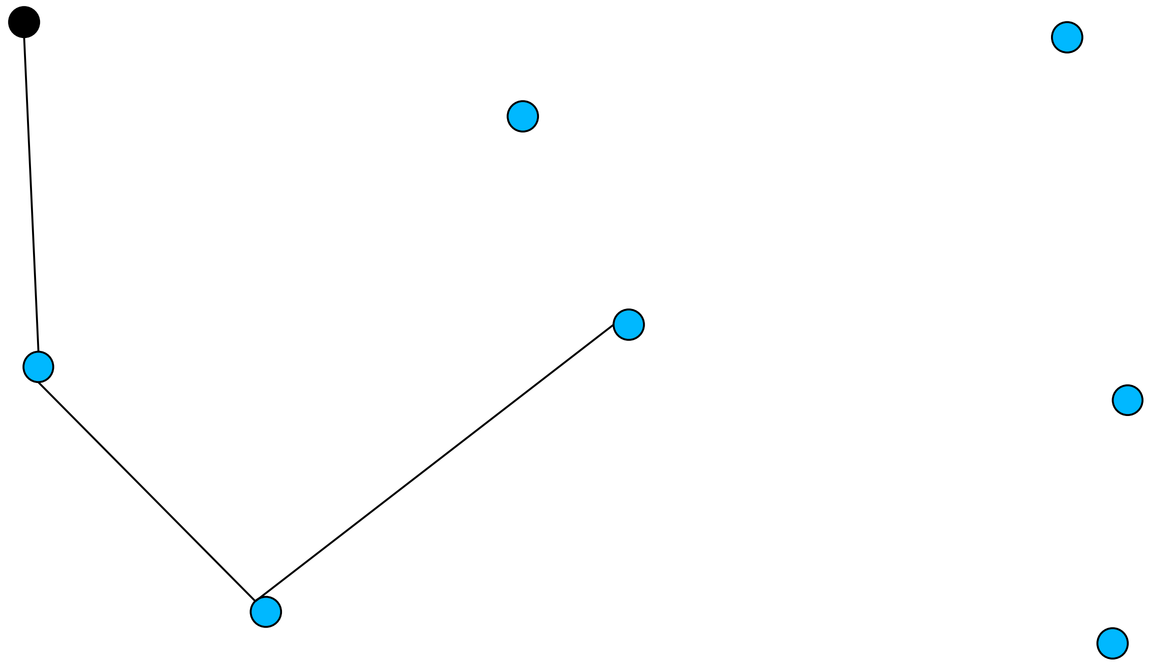
NN przykład



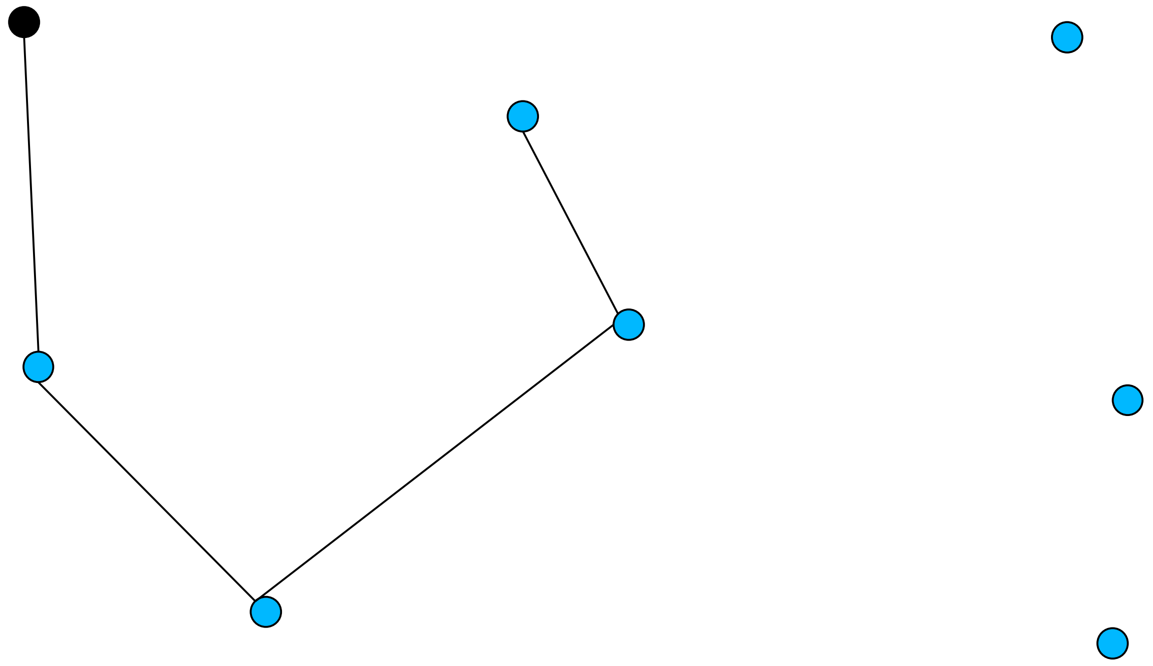
NN przykład



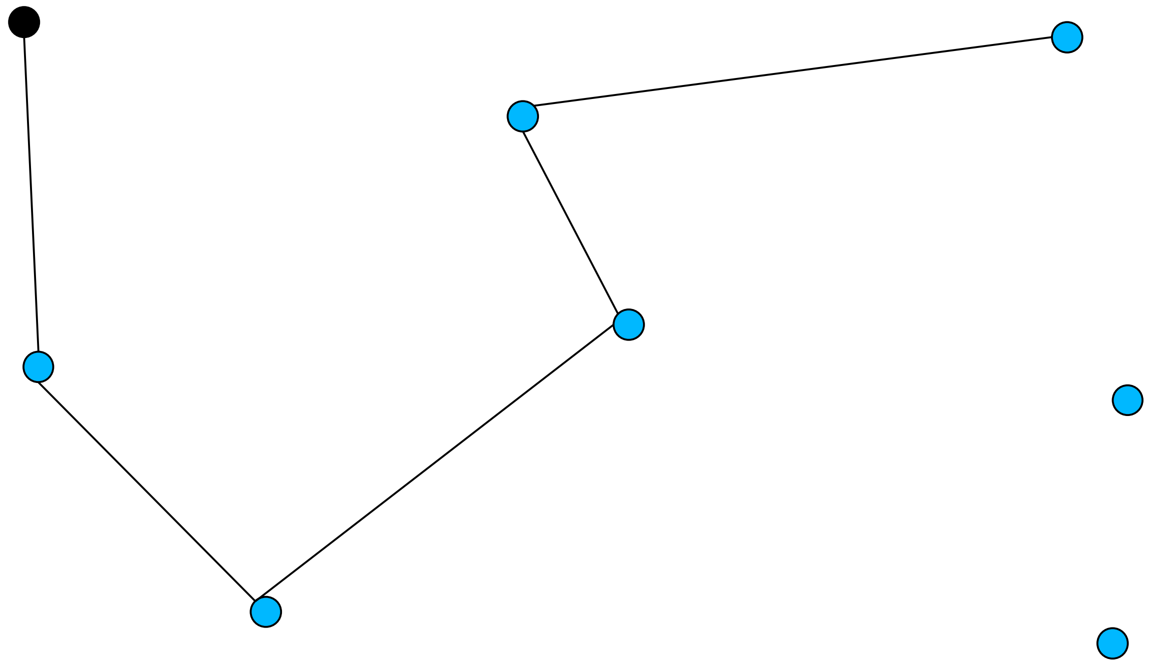
NN przykład



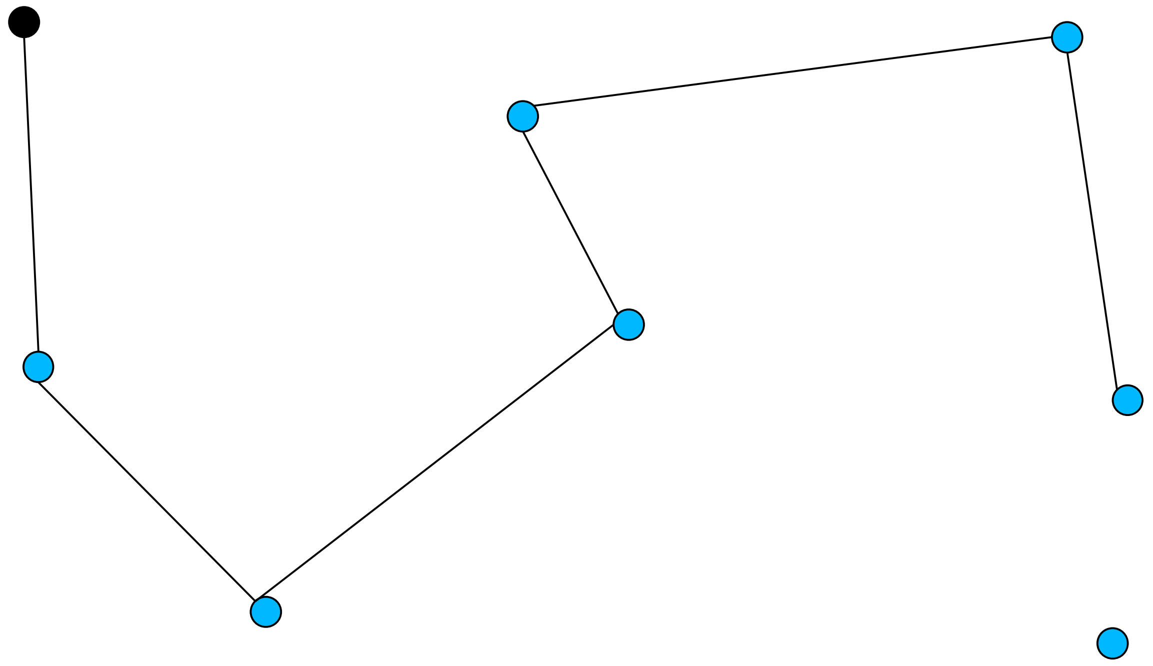
NN przykład



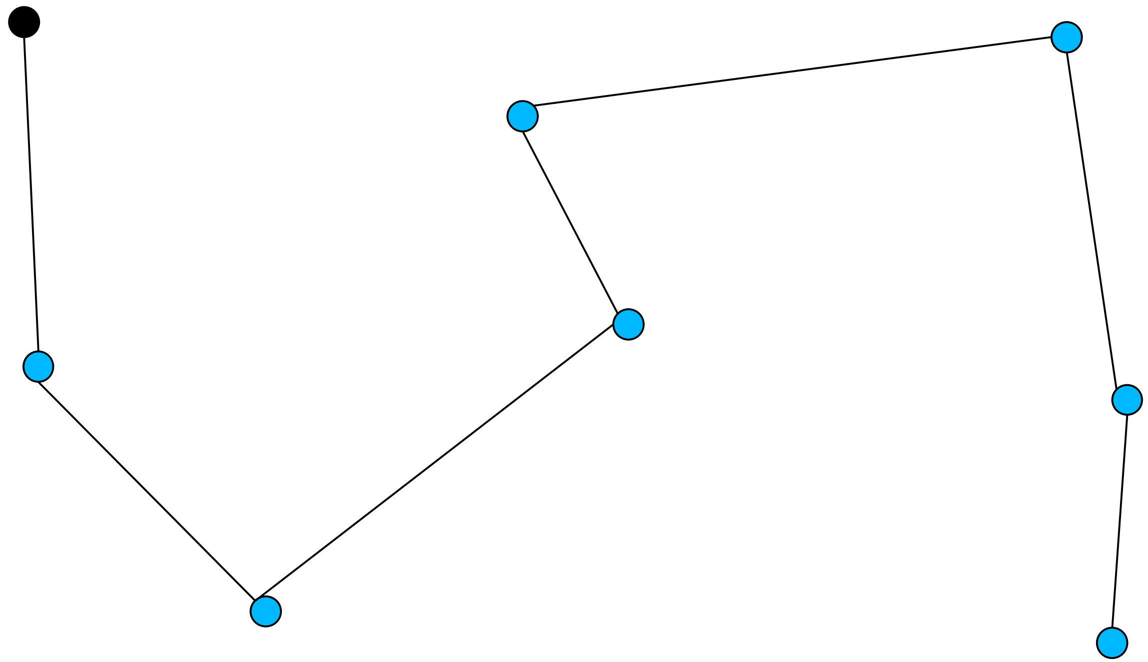
NN przykład



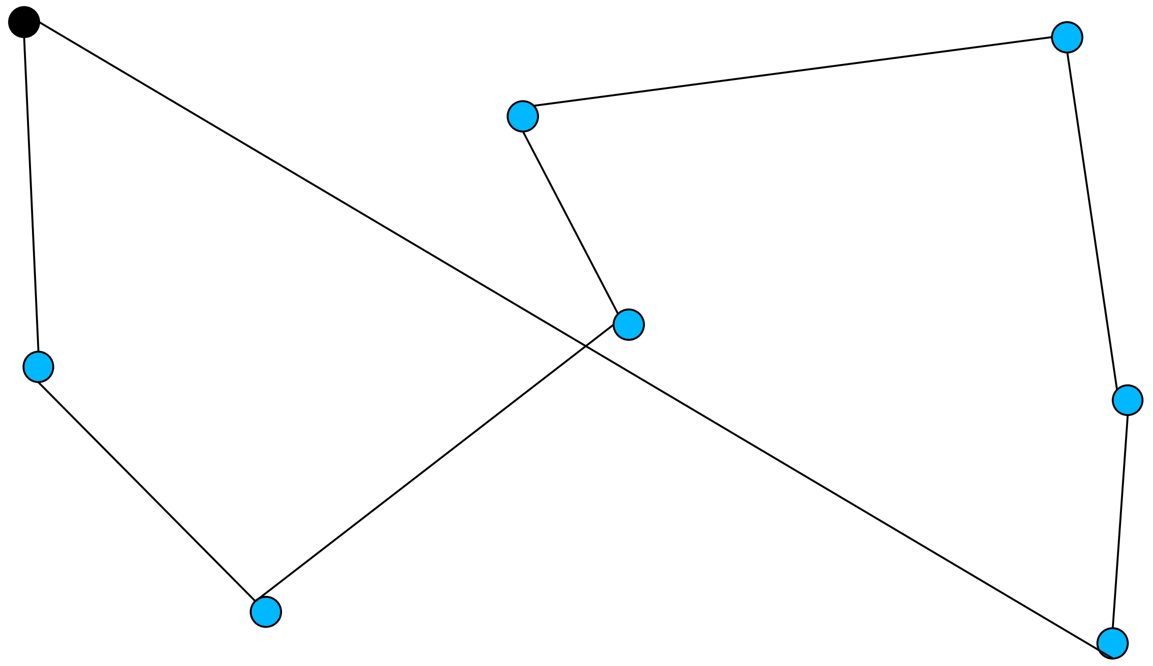
NN przykład



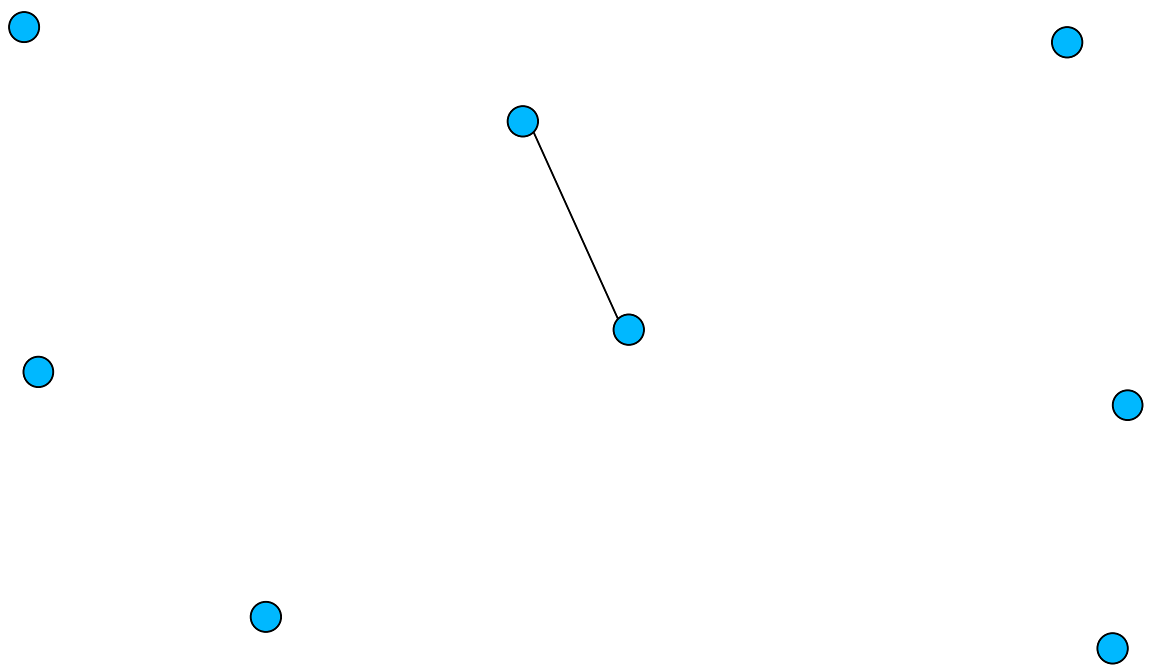
NN przykład



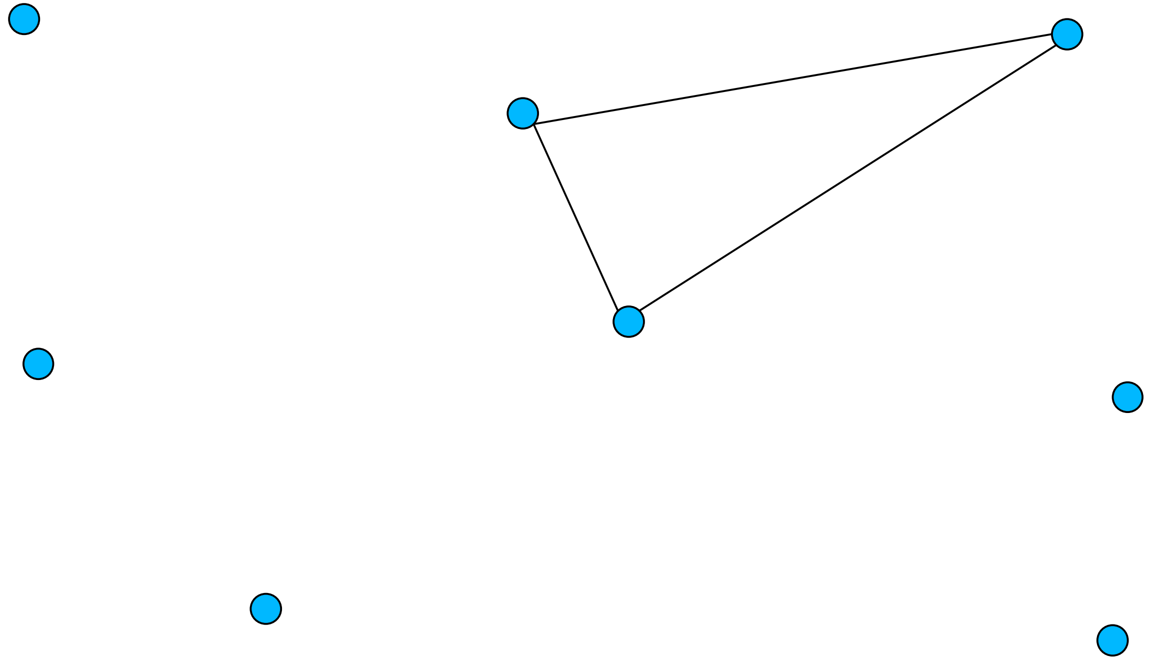
NN przykład



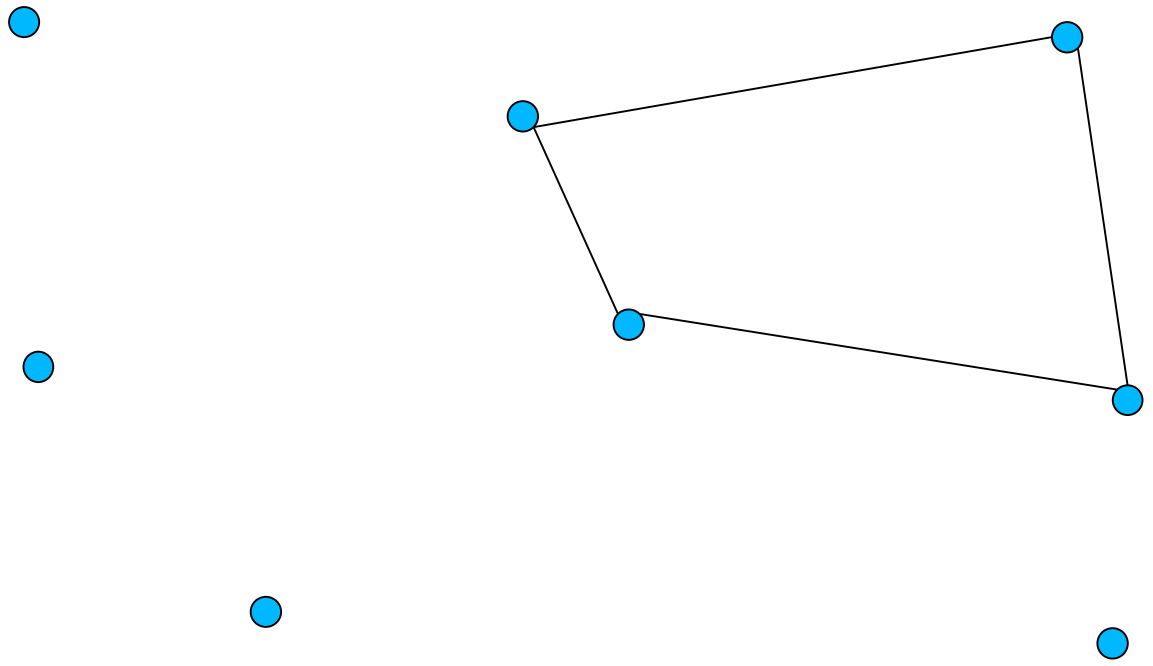
Greedy cycle przykład



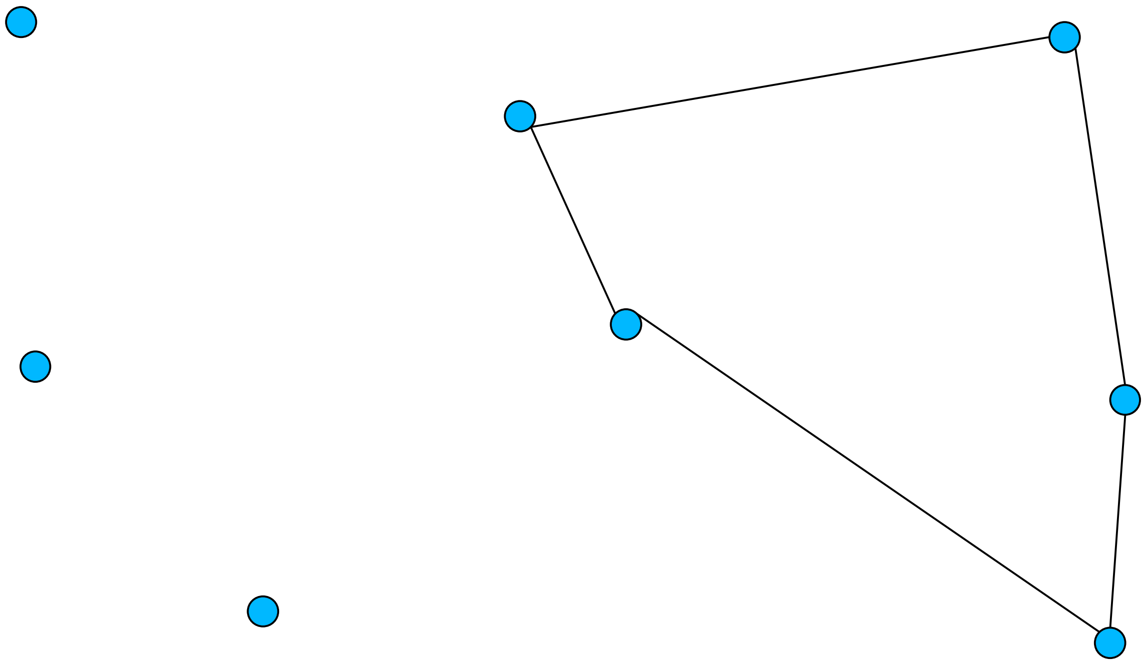
Greedy cycle przykład



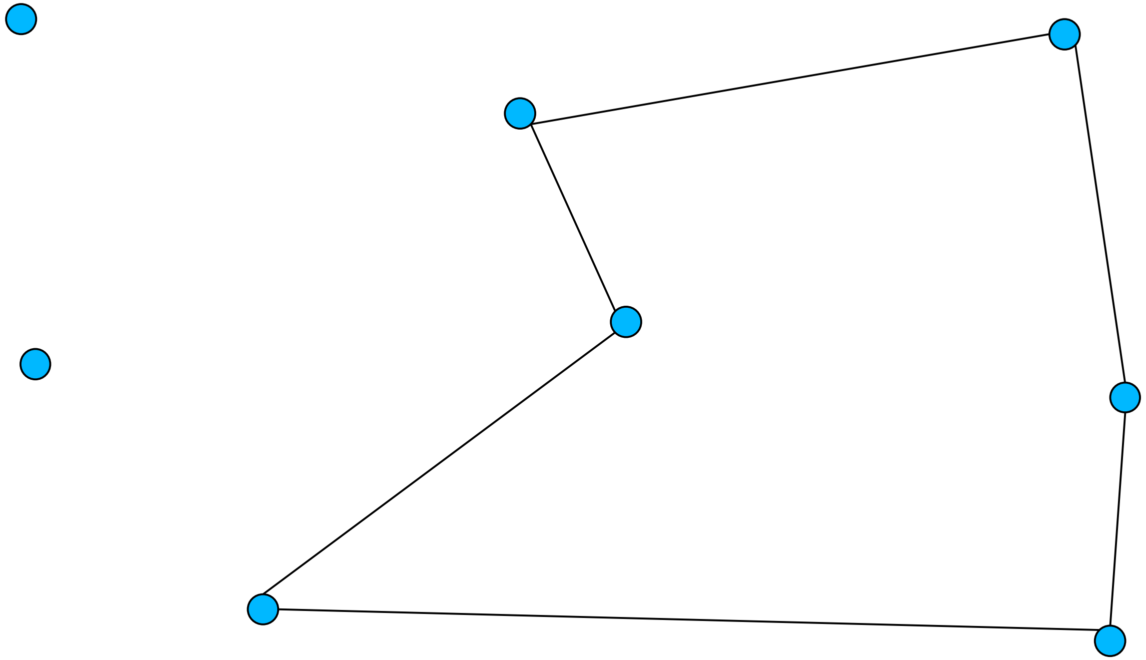
Greedy cycle przykład



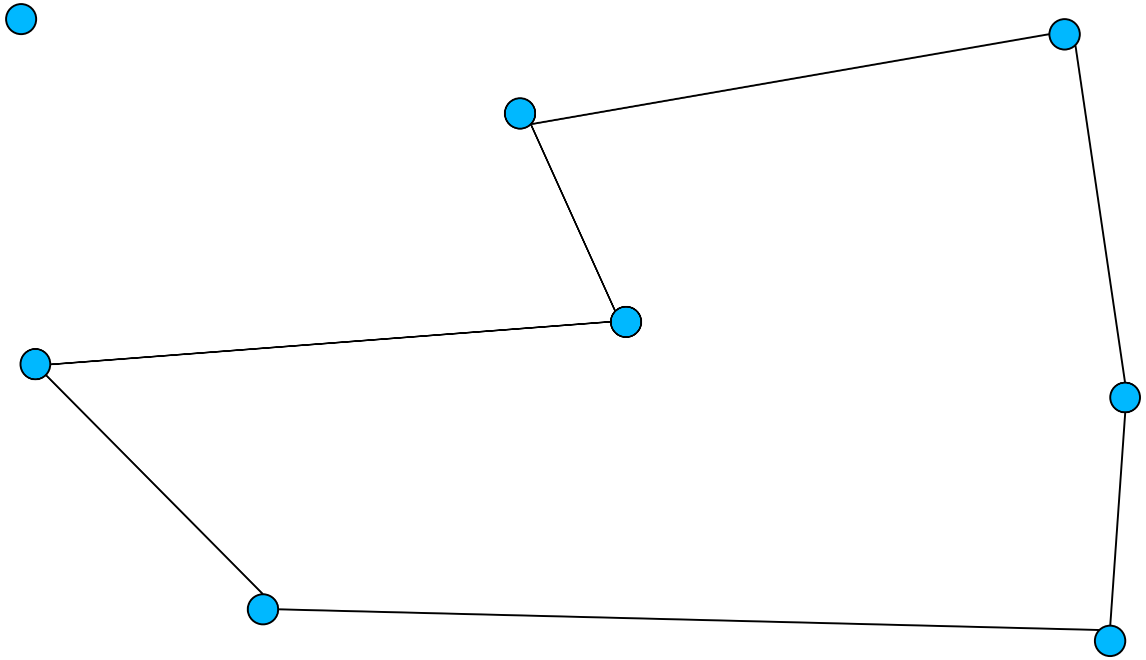
Greedy cycle przykład



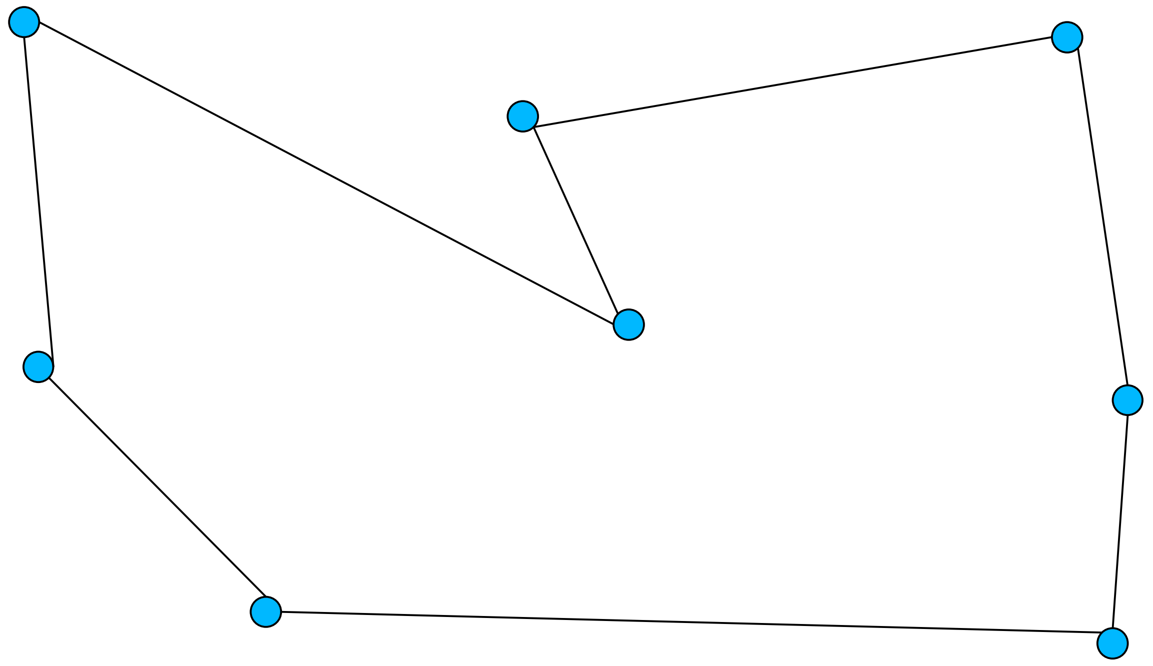
Greedy cycle przykład



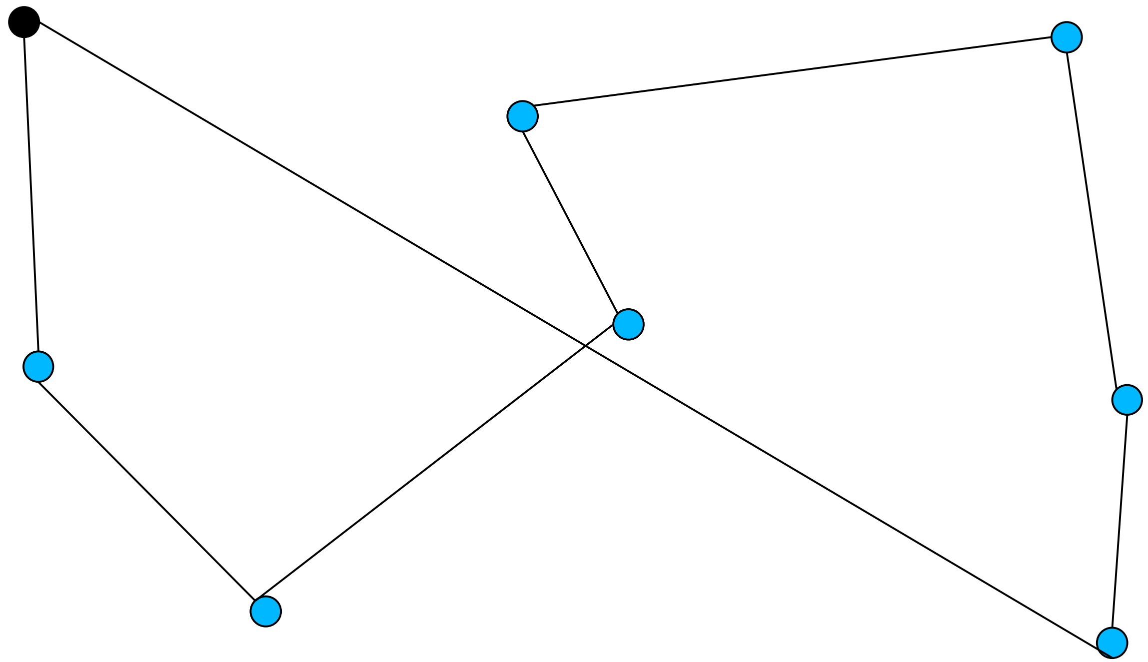
Greedy cycle przykład



Greedy cycle przykład



NN przykład



Heurystyka zachłanna dla problemu plecakowego

- Wstawiaj po kolei elementy o najwyższym stosunku wartości do wagi, które mieszczą się w plecaku, aż do zapełnienia plecaka.

GRASP

Greedy Randomized Adaptive Search Procedure

- T.A. Feo, M.G.C. Resende, Greedy Randomized Adaptive Search Procedures, Journal of Global Optimization 6, 109-133, 1995. http://www.research.att.com/~mgcr/papers.html

- Wadą heurystyk zachłannych jest ich determinizm, wystartowane z tego samego punktu (np. NN z tego samego wierzchołka), tworzą to samo rozwiązanie

- GRASP łączy zachłanność z losowością

GRASP

Konstrukcja dobrego rozwiązania

- Dokładanie pojedynczych elementów do rozwiązania
- Uporządkowana ograniczona lista kandydatów według tzw. funkcji zachłannej, która bierze pod uwagę elementy znajdujące się już w rozwiązaniu
- Wybór **losowy** jednego z najlepszych kandydatów z listy (zwykle nie jest to ten najlepszy)

GRASP Zachłanna procedura z elementem losowym

Utwórz rozwiązanie początkowe, np.

x := Ø

powtarzaj

zbuduj ograniczoną listę kandydatów RCL (restricted candidate list) najlepszych elementów nie wchodzących jeszcze w skład rozwiązania

dodaj do **x** losowo wybrany element z RCL

dopóki nie utworzono pełnego rozwiązania

Konstrukcja ograniczonej listy kandydatów RCL

- Pierwsza faza
  - *c(e)* – koszt przyrostu f. celu związany z włączeniem elementu $e \in E$ do rozwiązania
  - c^{min}, c^{max} – najmniejszy i największy koszt przyrostu
- RCL jest budowana z elementów o najmniejszym przyroście
- Ograniczenie
  - ze względu na ilość (*p* elementów, % elementów)
  - ze względu na jakość (α)
    - $c(e) \in [c^{min}, c^{min} + \alpha\, (c^{max} - c^{min})]$
    - $\alpha = 0$?
    - $\alpha = 1$?

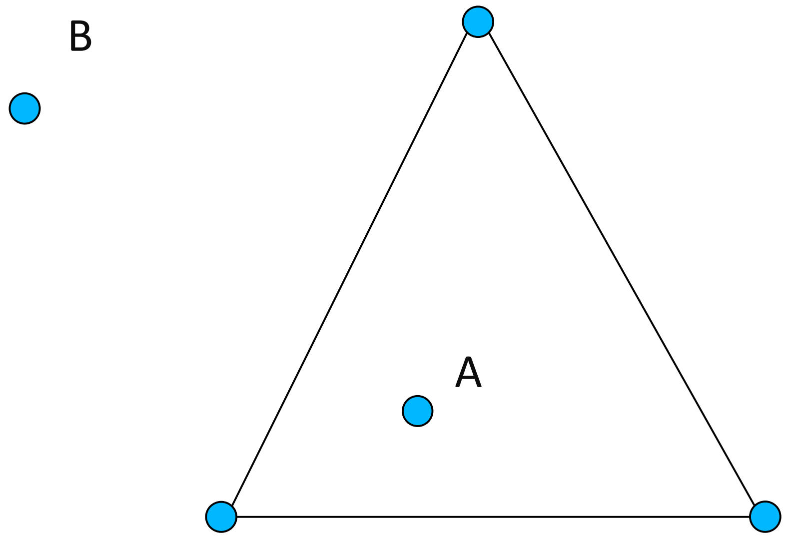
Regret heuristics

- Rozważmy wszystkie możliwe uporządkowane rosnąco koszty wstawienia dwóch elementów:
  - A: 10, 11, 12, 13, 13, 14
  - B: 20, 40, 45, 50, 50, 50
- Który element należałoby wybrać?
  - Zgodnie z zasadą zachłanną A…
  - … ale wtedy może się okazać, że stracimy opcję wstawienia B z kosztem 20 i zostanie nam tylko koszt 40 lub więcej

Regret heuristics

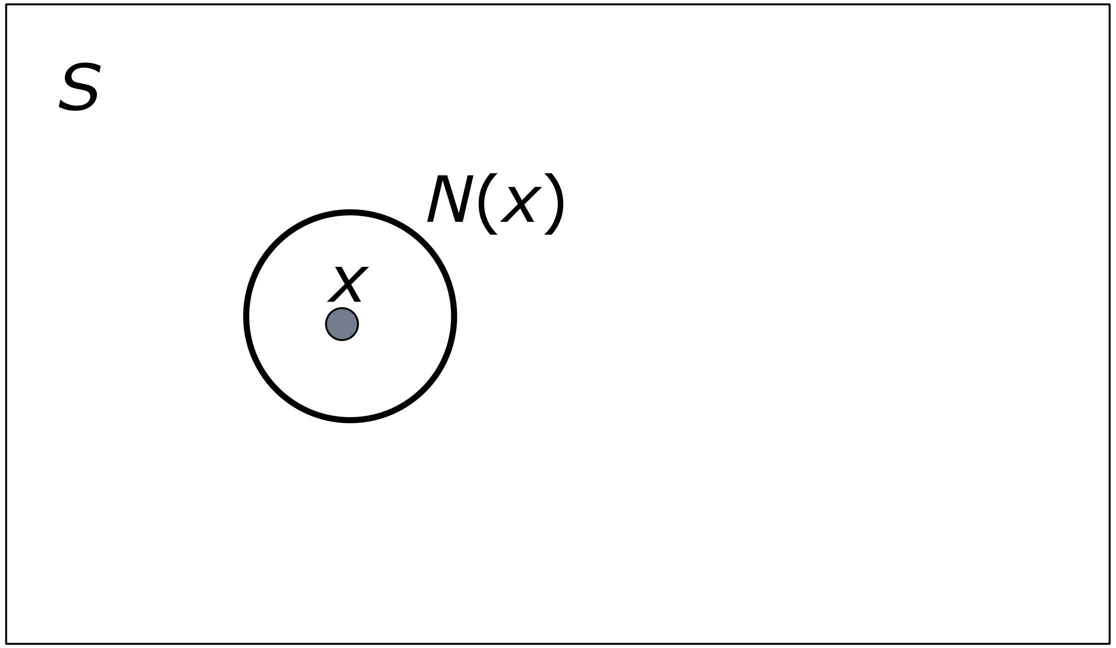
- k-żal (k-regret) to suma różnic pomiędzy najlepszym, a k kolejnymi opcjami wstawienia
- Wybieramy element o największym żalu
- Wstawiamy go w najlepsze miejsce

Regret heuristics - przykład



Przeszukiwanie lokalne

Idea sąsiedztwa



Definicja sąsiedztwa

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

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

- sąsiedztwo

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

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

Definicja sąsiedztwa (2)

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

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

Definicja sąsiedztwa (2) c.d.

- Inaczej mówiąc zakładamy, że miarą odległości rozwiązań *dist*(**x**, **y**) jest liczba ruchów jakie należy wykonać, aby przejść z jednego rozwiązania do drugiego, a ε w definicji:

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

  wynosi 1

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) – wymiana miast
    - *N*(*x*) – zbiór rozwiązań powstałych przez usunięcie *k* miast i wstawienie ich w innej kolejności
- 2-zamiana

$$1 - 2 - \boxed{3} - 4 - 5 - 6 - 7 - \boxed{8} - 9$$

$$1 - 2 - 8 - 4 - 5 - 6 - 7 - 3 - 9$$

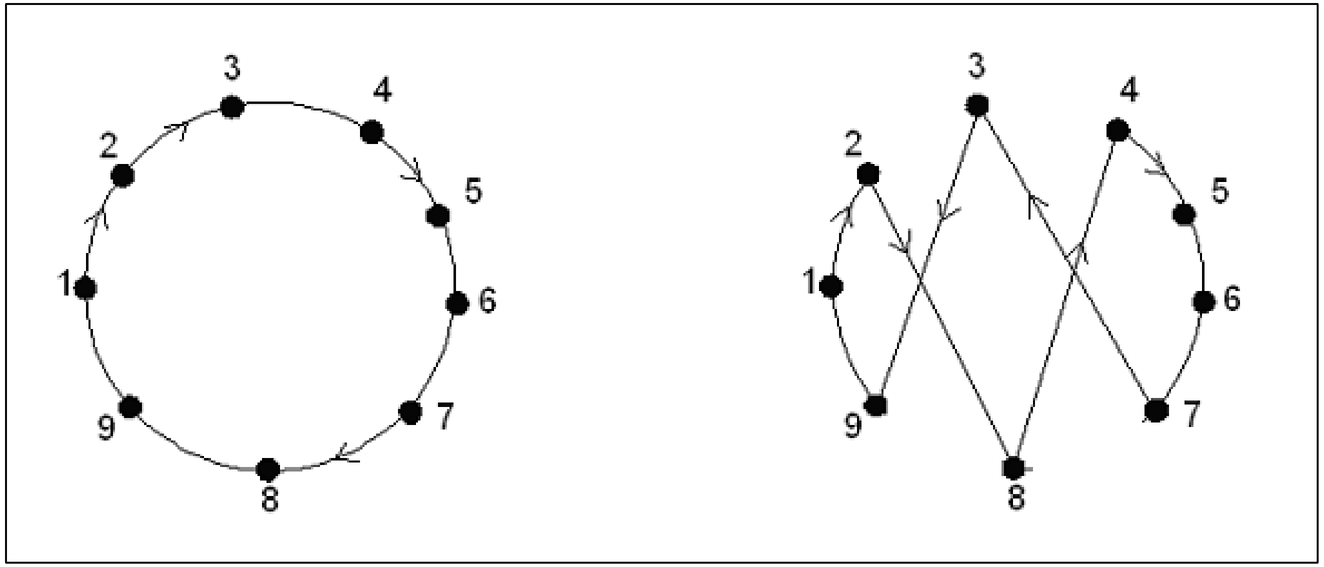
$$|N(x)| = n(n\text{-}1)/2$$

2-zamiana TSP

$$1 - 2 - \boxed{3} - 4 - 5 - 6 - 7 - \boxed{8} - 9$$

$N_2(3, 8)$

$$1 - 2 - 8 - 4 - 5 - 6 - 7 - 3 - 9$$



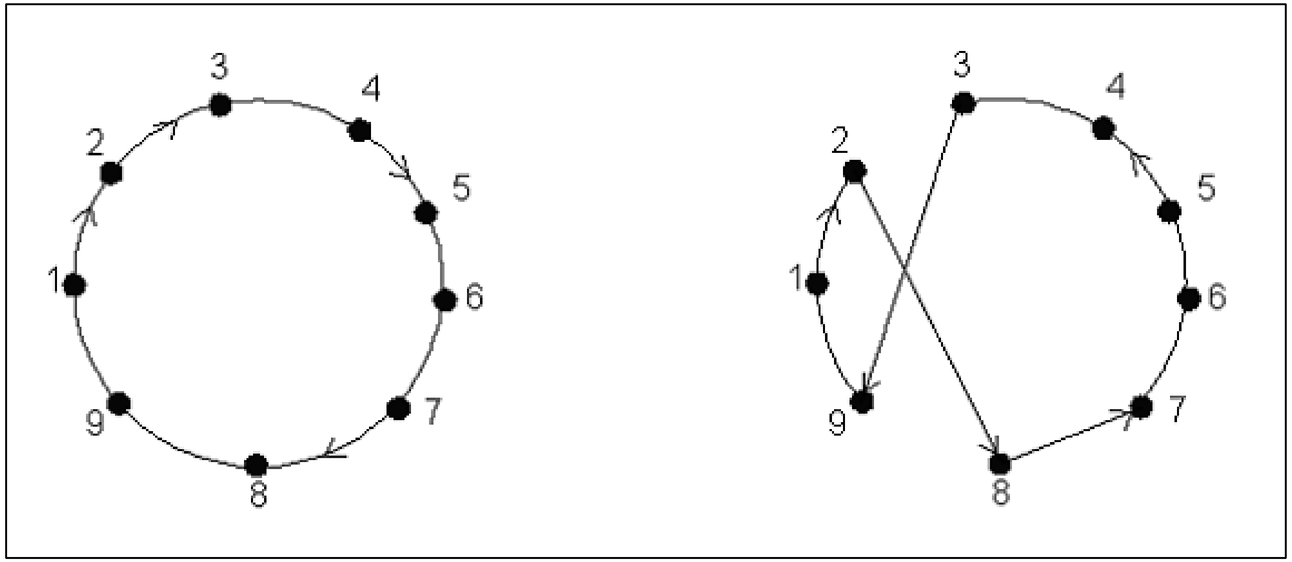
Sąsiedztwo TSP

wymiana łuków/krawędzi

1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9

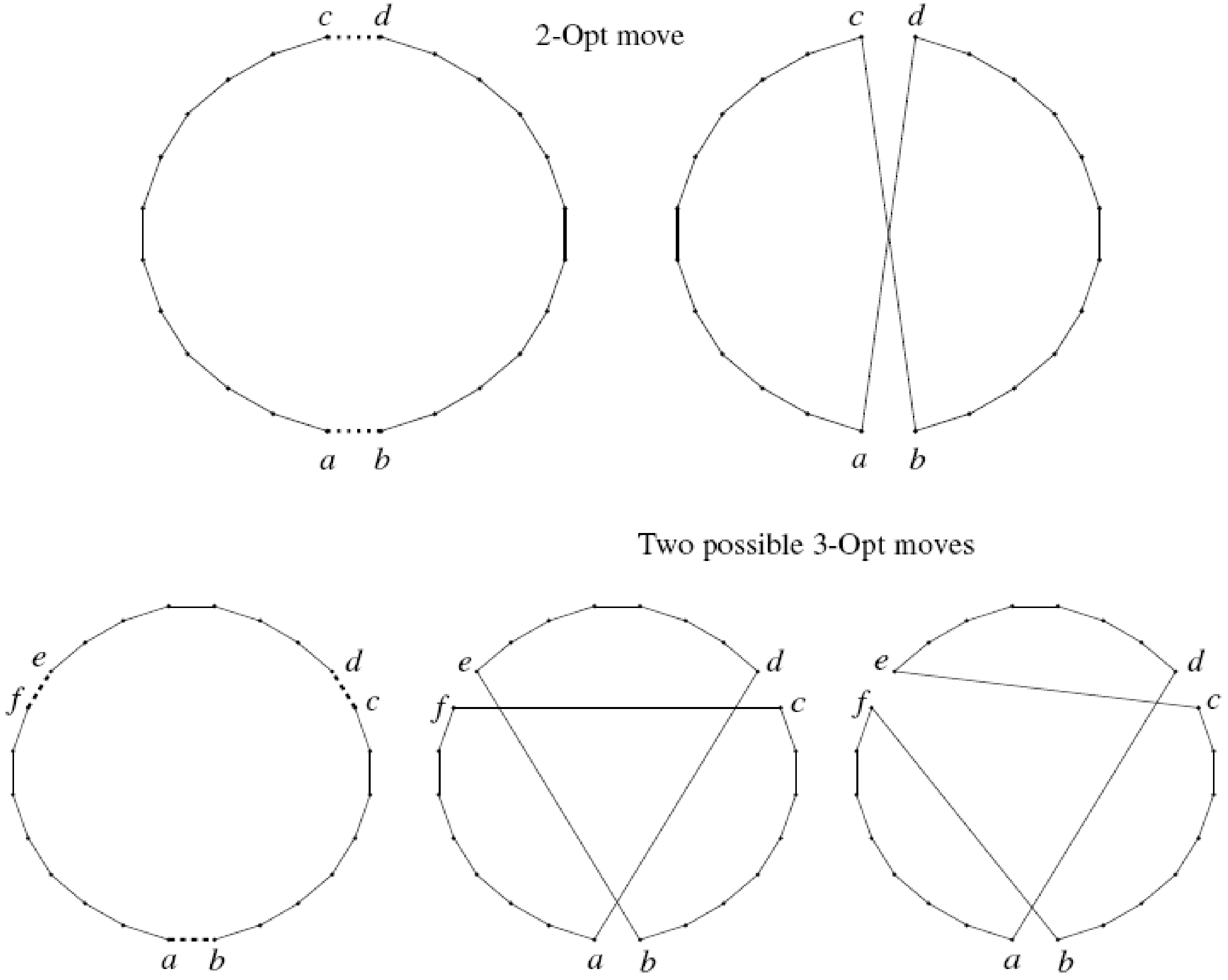
$N_2(3, 8)$

1 – 2 – 8 – 7 – 6 – 5 – 4 – 3 – 9



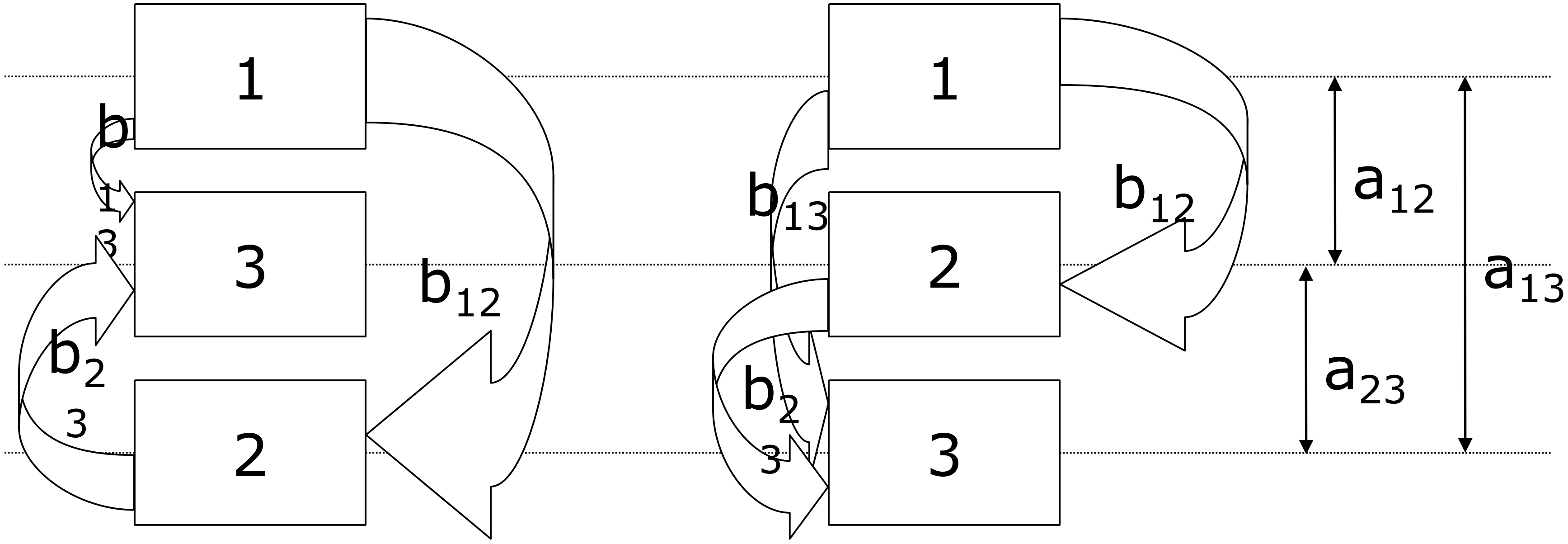
$$|N(x)| = n(n-3)/2$$

Wymiana łuków/krawędzi 2-opt i 3-opt



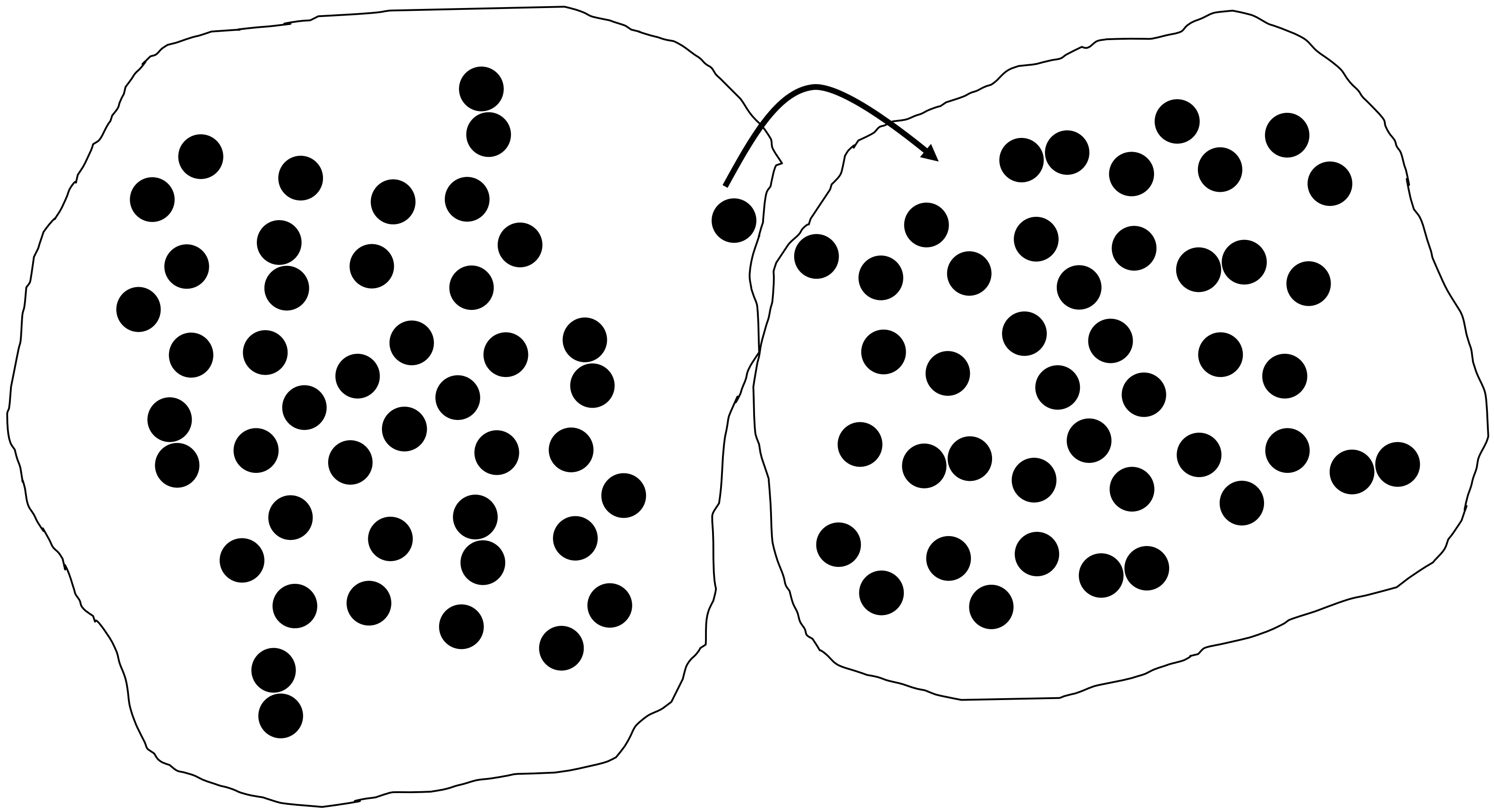
Sąsiedztwo QAP

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



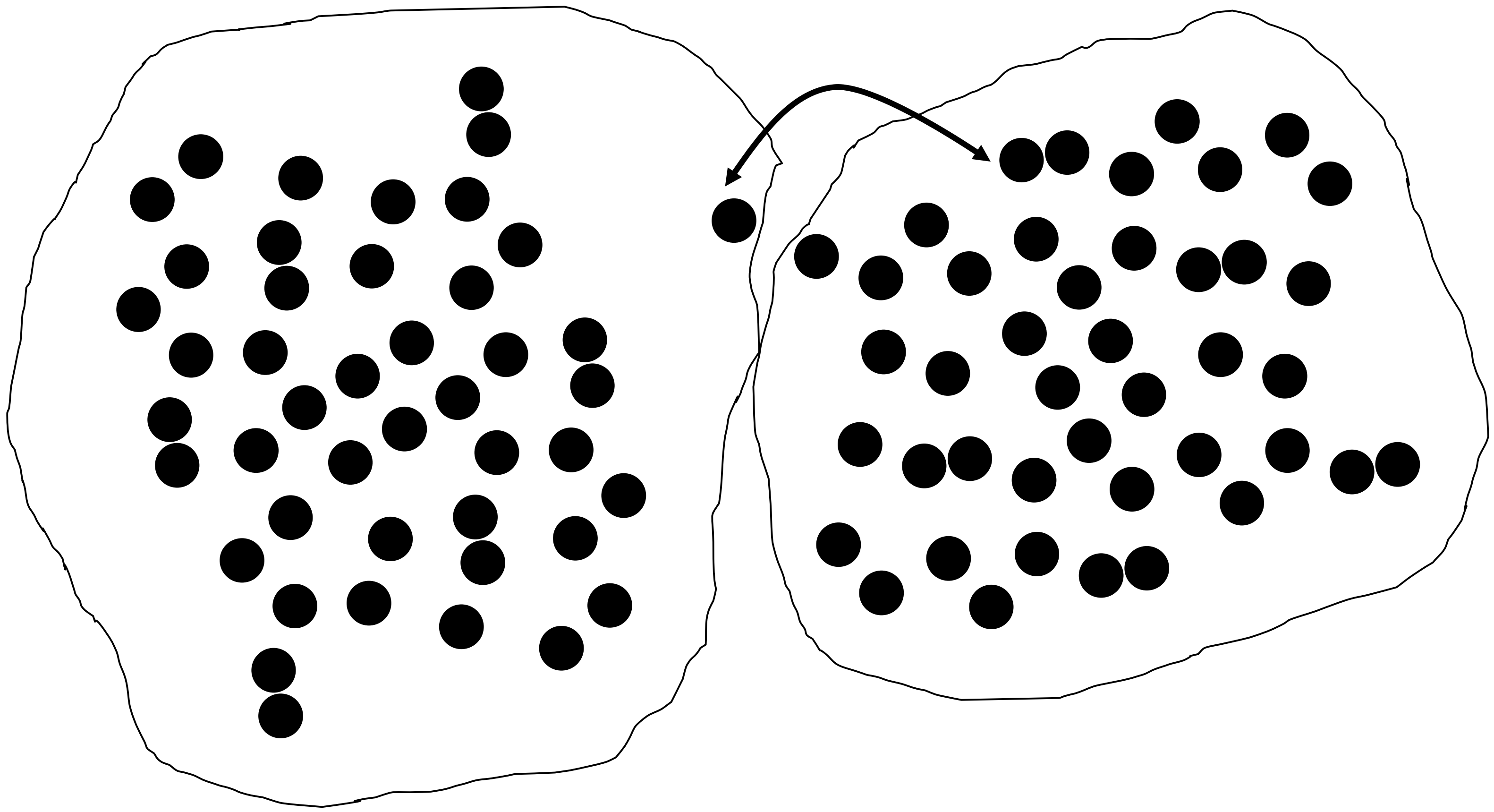
Sąsiedztwo grupowanie

- Przesunięcie wierzchołka



Sąsiedztwo grupowanie

- Wymiana wierzchołków



Sąsiedztwo grupowanie

- Przesunięcie obiektu
  - $|N(\mathbf{x})| = n\,(K - 1)$
- Wymiana obiektów
  - $|N(\mathbf{x})| \approx n\,((n - n / K)) / 2$

Przeszukiwanie lokalne

Eksploracja sąsiedztwa x

1. Wybierz rozwiązanie w *S* i oceń je, zdefiniuj jako rozwiązanie *bieżące*
2. Wygeneruj nowe rozwiązanie z rozwiązania bieżącego i oceń je
3. Jeśli nowe rozwiązanie jest lepsze zdefiniuj jako rozwiązanie bieżące, w przeciwnym wypadku odrzuć
4. Powtarzaj kroki 2 i 3 dopóki można uzyskać poprawę

Lokalne optimum

$\mathbf{x}_{min}$ jest *lokalnym minimum* jeśli

$$f(\mathbf{x}_{min}) \leq f(\mathbf{y}), \text{ dla wszystkich } \mathbf{y} \in N(\mathbf{x}_{min})$$

$\mathbf{x}_{max}$ jest *lokalnym maksimum* jeśli

$$f(\mathbf{x}_{max}) \geq f(\mathbf{y}), \text{ dla wszystkich } \mathbf{y} \in N(\mathbf{x}_{max})$$

Lokalne przeszukiwanie (local search LS)

Wersja zachłanna (greedy)

Wygeneruj rozwiązanie **x**

powtarzaj

  dla każdego y$\in$ *N*(**x**) w losowej kolejności

    jeżeli *f*(**y**) > *f*(**x**) **to**

      x := **y**

dopóki nie znaleziono lepszego rozwiązania po przejrzeniu całego *N*(**x**)

Wersja stroma (steepest)

Wygeneruj rozwiązanie **x**

powtarzaj

  znajdź najlepsze rozwiązanie **y** $\in$ *N*(**x**)

  jeżeli *f*(**y**) > *f*(**x**) **to**

      x := **y**

dopóki nie znaleziono lepszego rozwiązania po przejrzeniu całego *N*(**x**)

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 w takiej bezpośredniej implementacji 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 (delta)

Wersja stroma

Wygeneruj rozwiązanie **x**

powtarzaj

  znajdź najlepszy ruch $m \in M(\mathbf{x})$

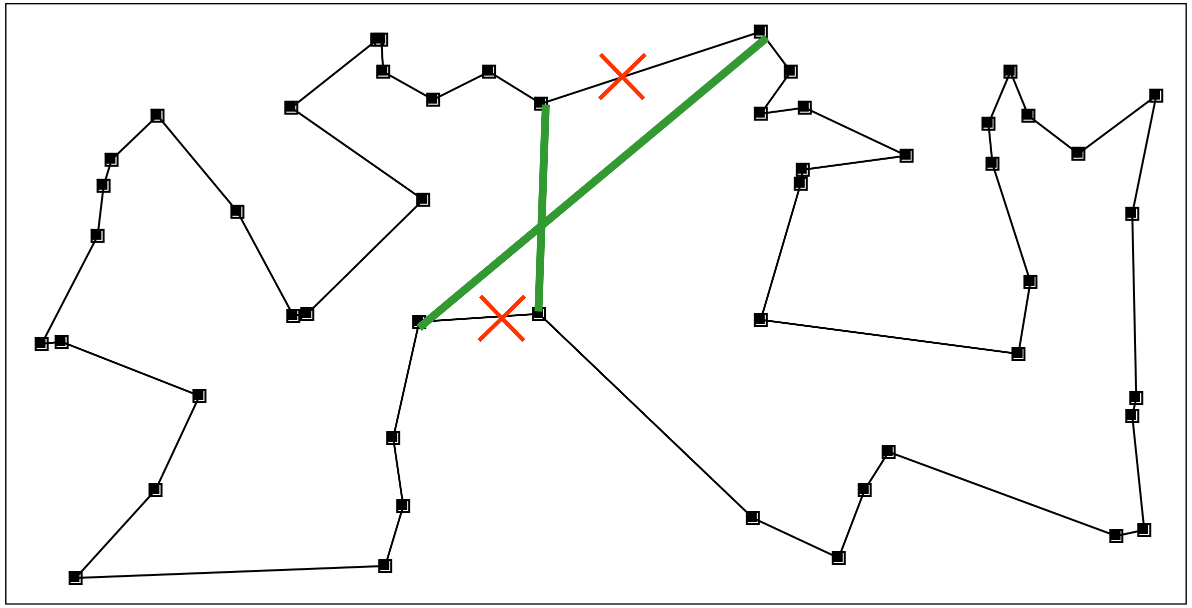
  jeżeli $f(\mathrm{m}(\mathbf{x})) > f(\mathbf{x})$ **to**

    $\mathbf{x} := m(\mathbf{x})$

dopóki nie znaleziono lepszego rozwiązania po przejrzeniu całego $N(\mathbf{x})$

- **Ocenia się ruchy (zmianę f. celu), nie rozwiązania! Rozwiązania sąsiednie nie muszą być jawnie konstruowane!**
- **Dopiero tu modyfikowane jest rozwiązanie**

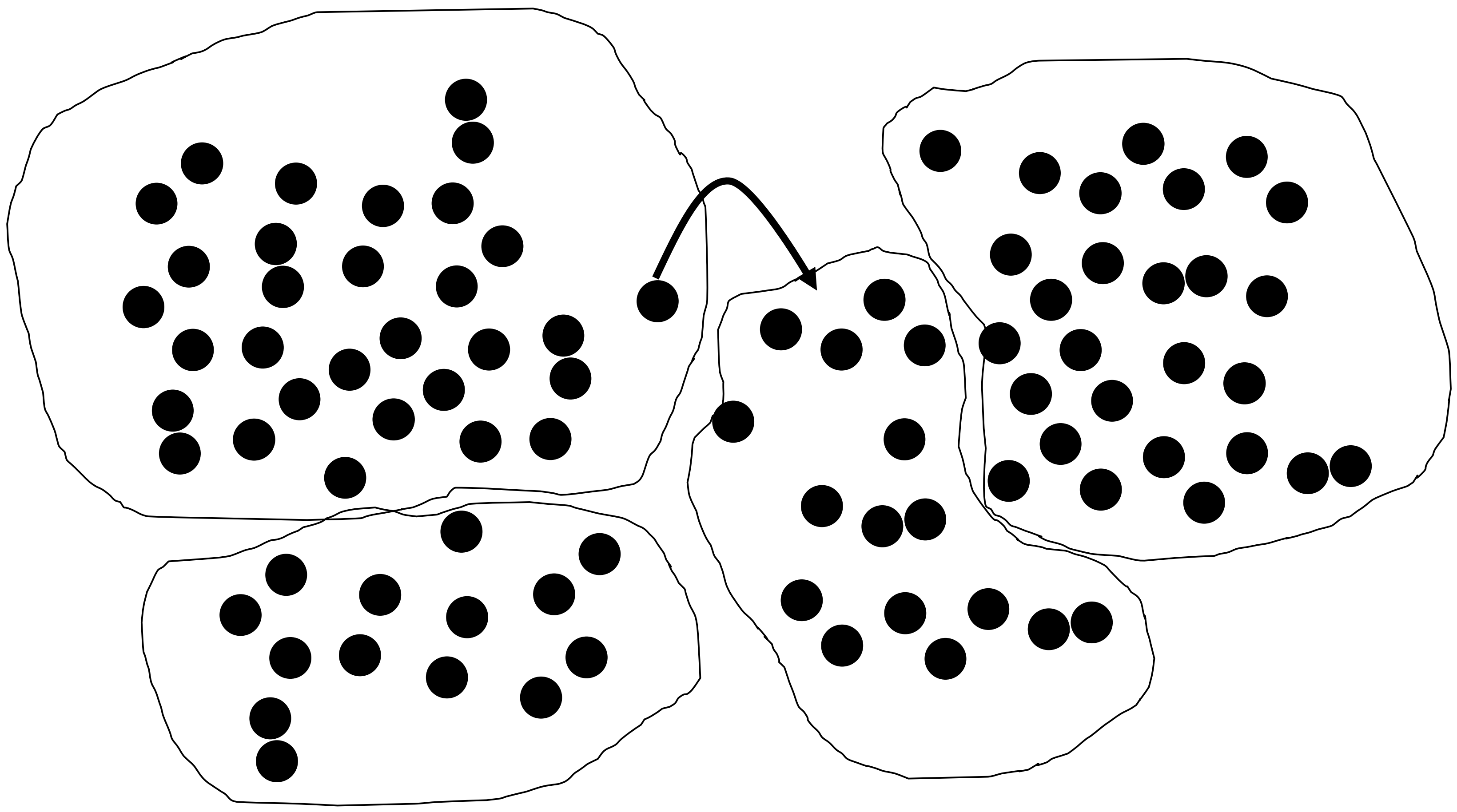
Ocena ruchów –problem komiwojażera (TSP)



Ocena ruchu:

Różnica kosztów dwóch dodawanych i dwóch usuwanych łuków

Grupowanie



Grupowanie

- Dany jest zbiór obiektów i macierz odległości pomiędzy każdą parą obiektów
- Obiekty należy podzielić na K grup minimalizując średnią odległość wewnątrzgrupową
- Ruch – przesunięcie obiektu z grupy do grupy
- Wielkość sąsiedztwa
  - $|N(\mathbf{x})| = n\ (K - 1)$

Obliczanie funkcji celu „od zera” (bez delty)

- Operacja jednostkowa – uwzględnianie (dodanie/odjęcie od średnie) odległości jednej pary obiektów
- „Od zera” trzeba przeliczyć n (n-1) / 2 par
- Złożoność jednej iteracji LS
  - $\underbrace{n\,(K-1)}_{\text{Liczba ruchów}} \times \underbrace{n\,(n-1)\,/\,2}_{\text{Operacji na ruch}}$

Złożoność przy wykorzystaniu delty

- Zmieniają się tylko dwie grupy, średnia wielkość grupy to n / K
- Należy uwzględnić 2 n / K odległości wewnątrzgrupowych
- Złożoność jednej iteracji
  - $n (K – 1) \times 2 n / K$
- Przyspieszenie
  - $(n (K – 1) \times n (n-1) / 2) / (n (K – 1) \times 2 n / K) = (n – 1) K / 4$

Koncepcja sąsiedztwa

- **Sąsiedztwo** $N(\mathbf{x})$ – zbiór rozwiązań, które można uzyskać lokalnie modyfikując **x**
- **Ruch** m – lokalna modyfikacja **x** do **y, y** = $m(\mathbf{x})$
- $M(\mathbf{x})$ – zbiór ruchów, które można zastosować do **x**
- $N(\mathbf{x}) = \{\mathbf{y} \in S : \exists m \in M(\mathbf{x}) | \mathbf{y} = m(\mathbf{x})\}$

- Ruchy powinny modyfikować rozwiązania lokalnie w przestrzeni rozwiązań i funkcji celu
- Rozmiar $M(\mathbf{x})$ ($N(\mathbf{x})$) powinien być dużo mniejszy od całej przestrzeni rozwiązań

Wykorzystanie ocen ruchów z poprzednich iteracji

- Dla $\mathbf{y} \in N(\mathbf{x})$ z reguły $N(\mathbf{x}) \cap N(\mathbf{y}) = \emptyset$ lub $|N(\mathbf{x}) \cap N(\mathbf{y})| \ll |N(\mathbf{x})|$
  - ale
- $M(\mathbf{x}) \cap M(\mathbf{y})$ może być liczny tzn. wiele ruchów pozostaje taka sama po wykonaniu ruchu
  - Czyli choć rozwiązania sąsiednie są inne, to wiele ruchów jakie można wykonać i ich ocen pozostaje takich samych

Lokalne przeszukiwanie w wersji stromej

Wygeneruj rozwiązanie **x**

powtarzaj

  znajdź najlepszy ruch $m \in M(\mathbf{x})$

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

    $\mathbf{x} := m(\mathbf{x})$ (zaakceptuj $m(\mathbf{x})$)

dopóki nie znaleziono lepszego rozwiązania po przejrzeniu całego $N(\mathbf{x})$

Wykorzystanie ocen ruchów z poprzednich iteracji

- W poprzedniej iteracji zmieniły się tylko dwie grupy
- Należy przeliczyć (z wykorzystaniem delty) tylko ruchy dotyczące zmienionych grup
- Liczba przesunięć obiektów z tych grup
  - $\approx \underbrace{2\,n / K}_{\text{Liczba obiektów w zmienionych grupach}} \times \underbrace{(K - 1)}_{\text{Liczba pozostałych grup}} \approx 2n$
- Liczba przesunięć obiektów do tych grup
  - $\approx \underbrace{(n - 2\,n / K)}_{\text{Liczba obiektów w pozostałych grupach}}\, 2 \approx 2n$
- Łącznie liczba ruchów dotyczących zmodyfikowanych grup
  - $\approx 4n$

Wykorzystanie ocen ruchów z poprzednich iteracji

- Razem liczba ruchów
  - $\approx 4n$
- Złożoność jednej iteracji (z deltą)
  - $\approx 4n \times 2\ n\ /\ K = 8\ n^2\ /\ K$
- Przyspieszenie (względem wersji z deltą)
  - $\approx\ (n\ (K-1)\ 2\ n\ /\ K)\ /\ (8\ n^2\ /\ K) = (K-1)\ /\ 4$
- Przyspieszenie względem wersji bez delty
  - $\approx (n\ (K-1)\ n\ (n-1)\ /\ 2)\ /\ (8\ n^2\ /\ K) = (n-1)(K-1)K\ /\ 16$

Czy można zrobić lepiej?

- Rozdzielmy delty na delty związane ze wstawianiem i usuwaniem obiektu z każdej grupy
- Przeliczenie takiej delty to $\approx$ n / K operacji
- Dla każdego obiektu spoza zmodyfikowanych ostatnio grup rozważamy jego usunięcie raz i wstawienie dwa ray
- Liczba delt (wstawienia/usunęcia) dla przesunięć obiektów z tych grup
  - $\approx 4\ n / K \times (K - 1) \approx 4n$
- Liczba delt (wstawienia/usunęcia) dla przesunięć obiektów do tych grup
  - $\approx (n - 2\ n / K)\ 3 \approx 3n$

- Złożoność jednej iteracji LS
  - $\approx 7n \times n / K = 7\ n^2\ K$
- Przyspieszenie (względem wersji z deltą)
  - $\approx\ (n\ (K - 1)\ 2\ n / K) / (7\ n^2 / K) = 2\ ((K - 1)) / 7$
- Przyspieszenie względem wersji bez delty
  - $\approx (n\ (K - 1)\ n\ (n-1) / 2) / (7\ n^2 / K) = (n - 1)(K - 1)K / 14$

Alternatywne spojrzenie

- Rozdzielamy ruch na operacje wstawienia i usunięcia obiektu
- Dla każdej operacji (wstawienia lub usunięcia) w pierwszej iteracji LS obliczamy delty
- Następnie przeliczamy tylko delty, które się zmieniły

Wykorzystanie ocen ruchów z poprzednich iteracji

- Technicznie
  - Cache ruchów z poprzedniej iteracji
    - lub
  - Uporządkowana wg. delty lista ruchów przynoszących poprawę
  - Związany z tym narzut może redukować efekty

Wykorzystanie ocen ruchów z poprzednich iteracji – uporządkowana lista ruchów – wersja stroma

Zainicjuj *LM* – listę ruchów przynoszących poprawę uporządkowaną od najlepszego do najgorszego

Wygeneruj rozwiązanie **x**

powtarzaj

  przejrzyj wszystkie **nowe** ruchy i dodaj do *LM* ruchy przynoszące poprawę

  Przeglądaj ruchy *m* z *LM* od najlepszego do znalezienia aplikowalnego ruchu

    Sprawdź czy *m* jest aplikowalny i jeżeli nie, usuń go z *LM*

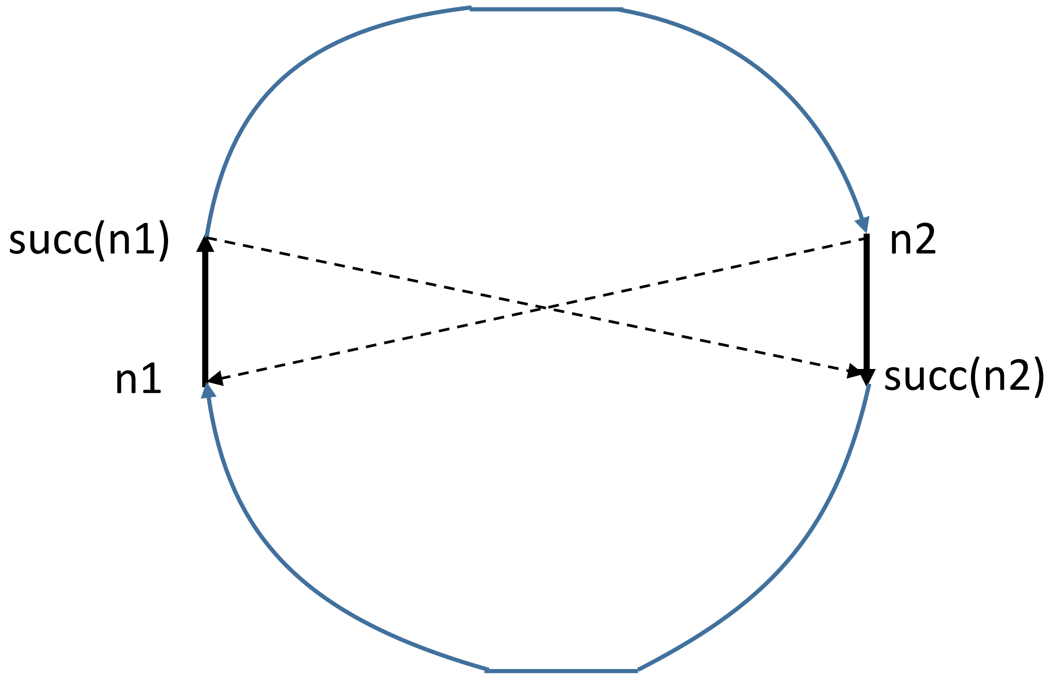
  jeżeli znaleziono ruch *m* **to**

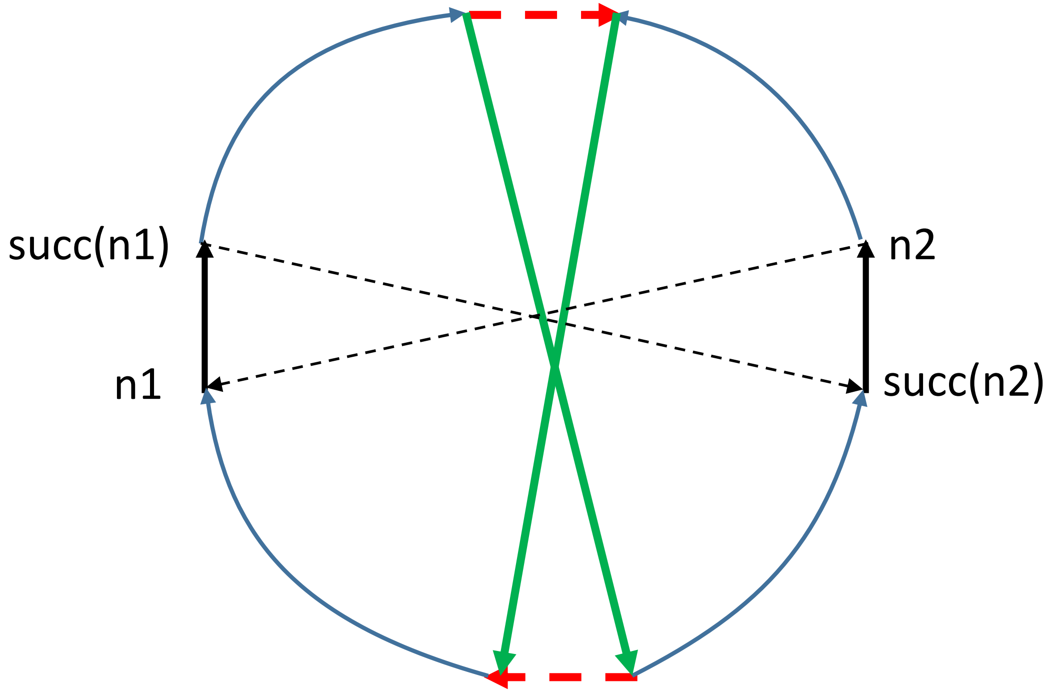
    x := *m*(**x**) (zaakceptuj *m*(**x**))

dopóki nie znaleziono ruchu *m* po przejrzeniu całej listy *LM*

Przykład dla TSP

Oceniamy ruch związany z usunięciem krawędzi (n1, succ(n1)), (n2,succ(n2))

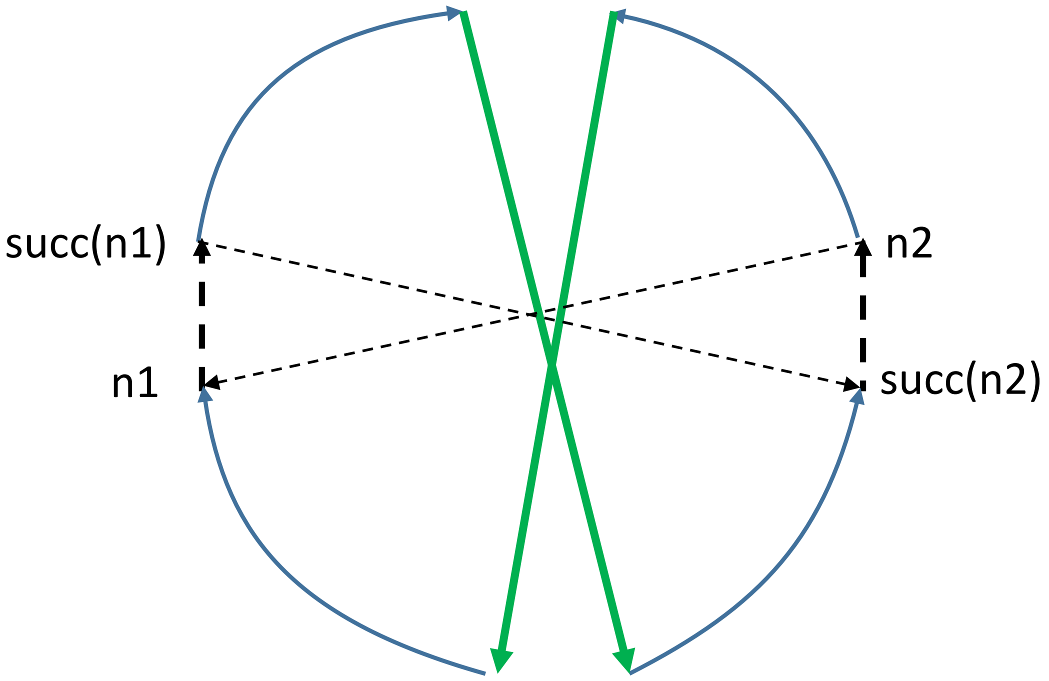




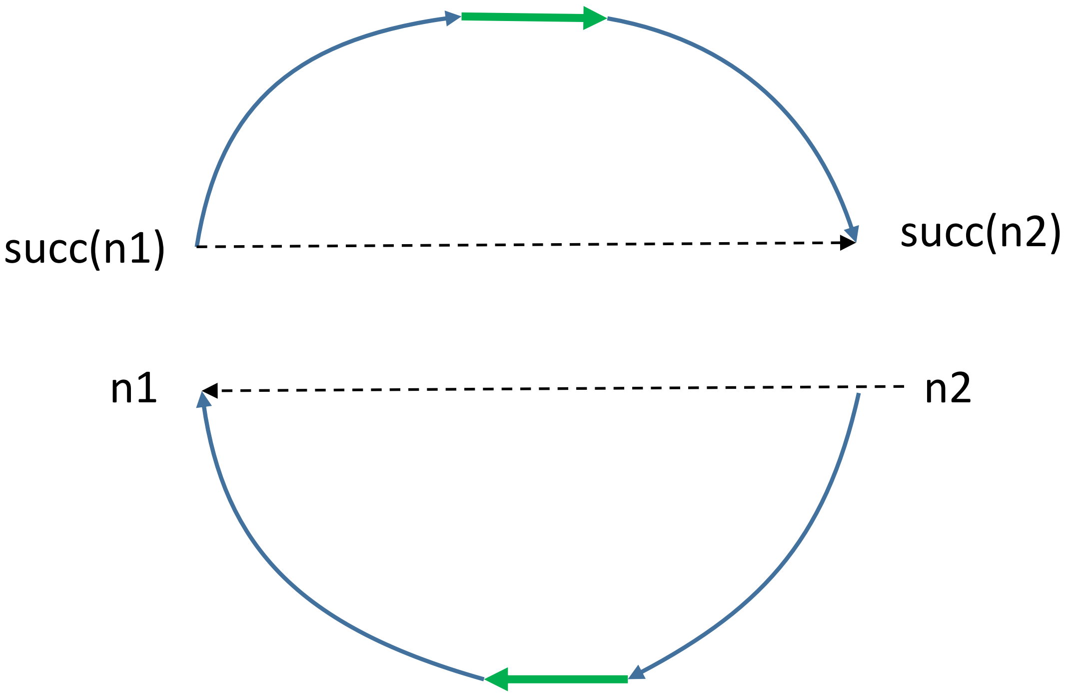
Zapamiętujemy ruch
Dodajemy krawędzie (n1,n2), (succ(n1),succ(n2))

Wykonujemy inny ruch

Przykład dla TSP



Po wykonaniu innego ruchu



Zapamiętany ruch nie jest poprawny,
ale może stać się poprawny po ponownym odwróceniu

Przykład c.d.

Wnioski:
Musimy też brać pod uwagę kierunek przechodzenia krawędzi w bieżącym rozw.
Każdy ruch może tę kolejność zmienić
3 sytuacje:

- Usuwane krawędzie nie występują już w bież. rozw. -> usuwamy ruch z *LM*
- Usuwane krawędzie występują w bież. rozw. w różnym kierunku -> zostawiamy ruch w *LM*, ale nie aplikujemy
- Usuwane krawędzie występują w bież. rozw. w tym samym kierunku – aplikujemy i usuwamy z *LM*

Globalna pamięć delt

- Do tej pory zakładaliśmy zapamiętywanie bieżących delt
- Delty mogą się jednak powtarzać w trakcie obliczeń w dalszych iteracjach – np. wstawianie/usuwanie takiego samego obiekty do/z takiej samej grupy
  - W szczególności w metodach wyższego rzędu obejmujących wiele (równoległych) uruchomień LS
- Można więc zapamiętywać wszystkie już obliczone delty w pamięci globalnej
  - Hashowanie w celu poprawy efektywności
  - Techniki zarządzania pamięcią, np. usuwanie najrzadziej używanych delt

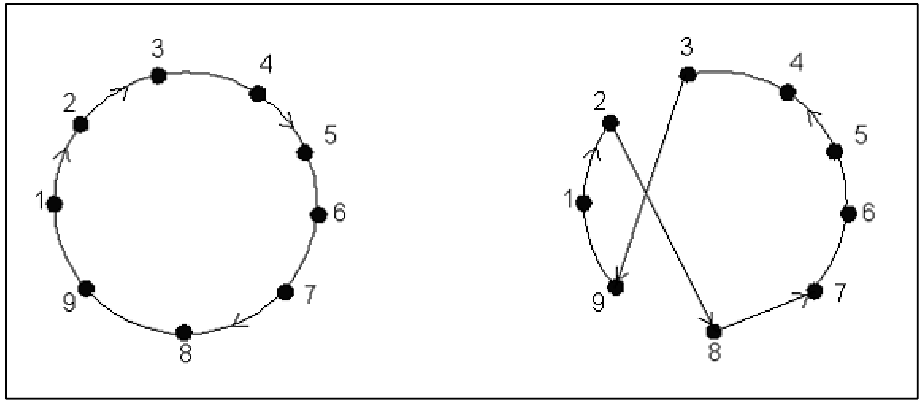
Túlio A.M. Toffolo, Thibaut Vidal, Tony Wauters, Heuristics for vehicle routing problems: Sequence or set optimization?, Computers & Operations Research, Volume 105, 2019, Pages 118-131,https://doi.org/10.1016/j.cor.2018.12.023.

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



Standardowe przeglądanie sąsiedztwa TSP – wymiana dwóch łuków

Wersja steepest

Dla każdego wierzchołka n1 od 0 do N-1

- **Dla każdego** wierzchołka n2 od n1+1 do N-1
  - **jeżeli** łuki n1-następnik(n1) i n2-następnik(n2) nie są sąsiednie
    - Sprawdź czy ruch (n1, n2) przynosi poprawę i jest najlepszym dotąd znalezionym

Przeglądanie sąsiedztwa TSP z ruchami kandydackimi– wymiana dwóch łuków

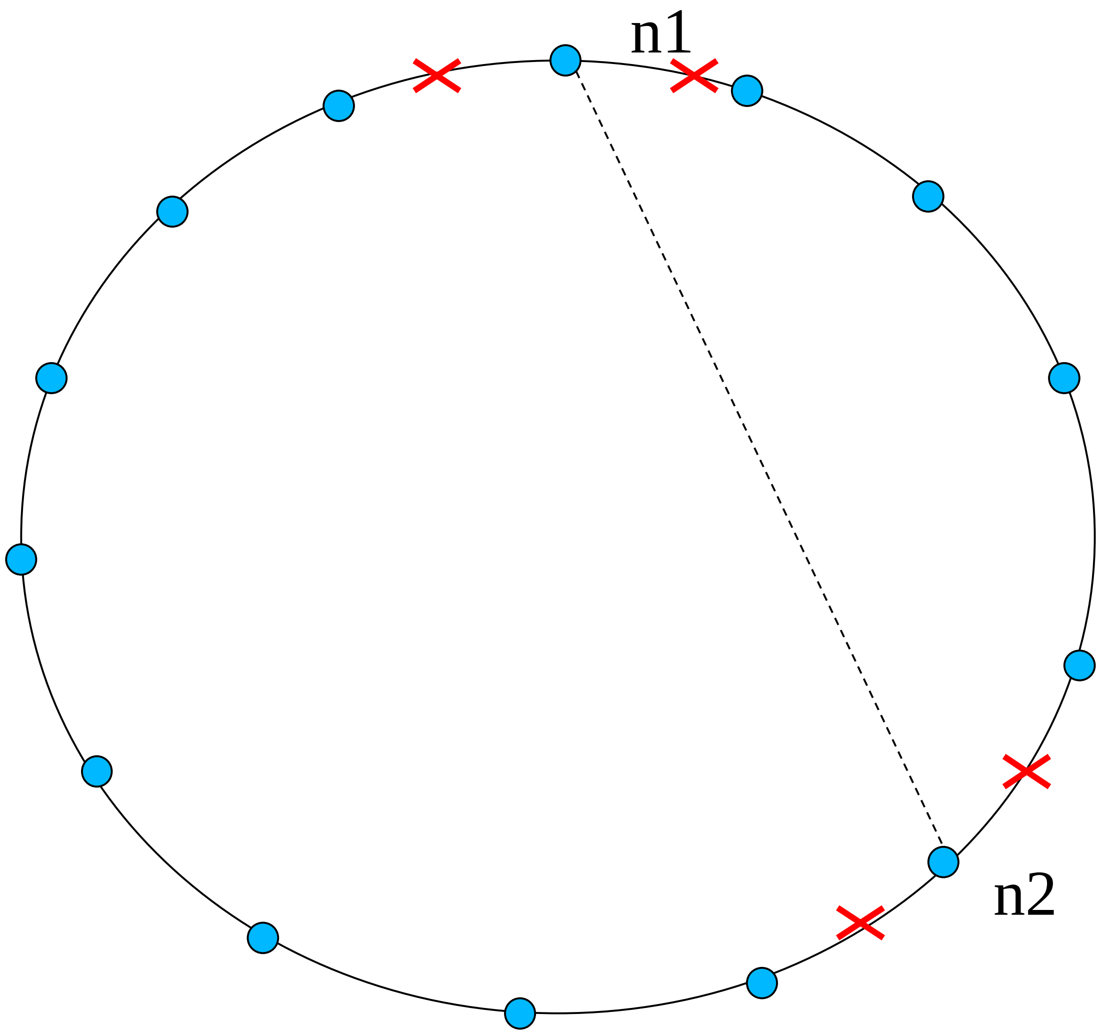
Wersja steepest

Dla każdego wierzchołka n1 od 0 do N-1

Dla każdego <u>łuku kandydackiego n1-n2</u>

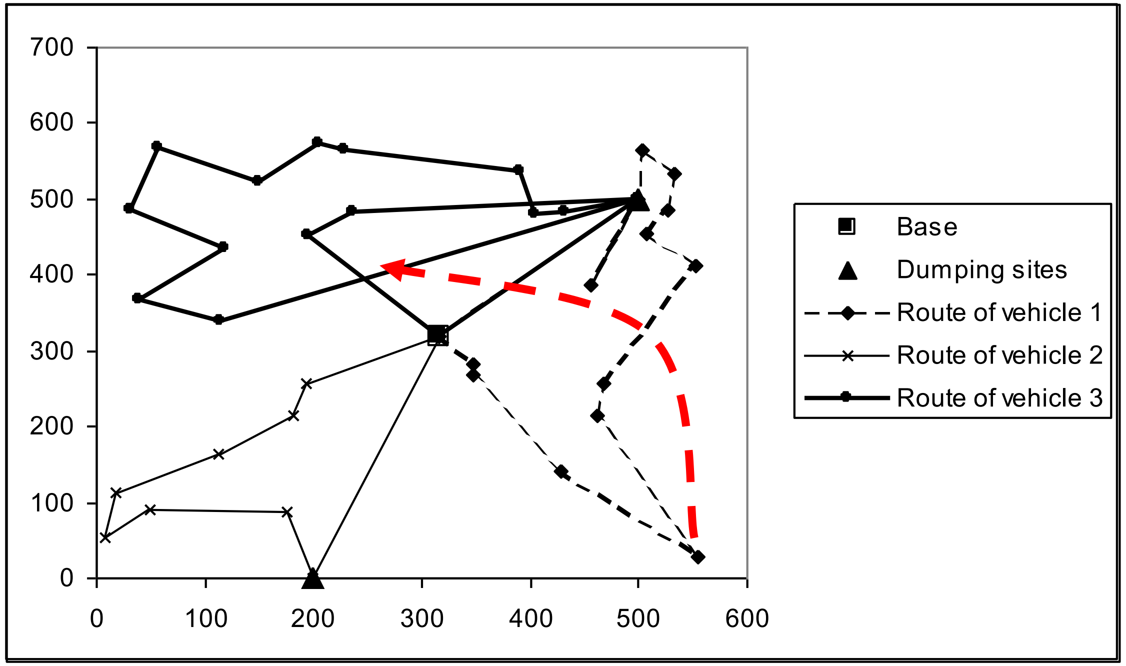
sprawdź wszystkie ruchy polegające do dodaniu łuku n1-n2 i usunięciu jednego z obecnych łuków łączących n1

Ruchy kandydackie w TSP



Pomijanie złych ruchów na podstawie reguł heurystycznych

- W VRP można rozważać tylko ruchy pomiędzy blisko położonymi trasami



Ruchy kandydackie w TSP na podstawie znanych rozwiązań

- Generujemy pewną liczbę rozwiązań stosując heurystykę lub lokalne przeszukiwanie bez ruchów kandydackich
- Każdy łuk, który wystąpi choć raz w jednym z tych rozwiązań staje się łukiem kandydackim

Pomijanie złych ruchów na podstawie reguł heurystycznych

- Ruchy złe i kandydackie mogą być identyfikowane na podstawie działania metody wyższego rzędu, np.
- 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 u dowolnego z rodziców (nawet jeżeli nie znalazły się one w potomku)

Ruchy kandydackie grupowanie

- Rozważaj przesunięcie obiektu tylko do bliskich grup
- Jak zdefiniować bliskie grupy?
  - Środek ciężkości
  - Lista najbliższych obiektów, warunek występowanie co najmniej jednego z listy
  - Lista obiektów grupowanych razem w poprzednich rozwiązaniach

Techniki zaawansowane

- Np. przeglądanie sąsiedztwa o rozmiarze wykładniczym w czasie wielomianowym

Wygeneruj rozwiązanie **x**

powtarzaj

znajdź najlepszy ruch $\mathbf{m} \in M(\mathbf{x})$ Problem optymalizacji

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

$\mathbf{x} := \mathrm{m}(\mathbf{x})$

dopóki nie znaleziono lepszego rozwiązania

- Zastosowanie rozszerzeń kwantowego algorytmu Grovera $\Theta(\sqrt{|M(\mathbf{x})|})$