

Inteligentne metody optymalizacji

Andrzej Jaskiewicz

Zakres przedmiotu

- Wprowadzenie
 - Elementy zagadnienia optymalizacji. Źródła trudności zagadnień optymalizacji. Przykłady zagadnień optymalizacji z naciskiem na zagadnienia dyskretne/kombinatoryczne. Klasyfikacja metod optymalizacji.
- Metody pełnego przeglądu. Idea metody podziału i ograniczeń. Przeszukiwanie losowe.
- Heurystyki zachłanne
 - Idea. Przykłady. Randomizacja. Heurystyki zachłanne oparte na żalu (regret).
- Lokalne przeszukiwanie
 - Idea sąsiedztwa. Przeszukiwanie lokalne w wersji stromej i zachłannej. Poprawa efektywności lokalnego przeszukiwania.
- Rozszerzenie lokalnego przeszukiwania
 - Lokalne przeszukiwanie z wieloma punktami startowymi. Lokalne przeszukiwanie ze zmiennym sąsiedztwem. Iteracyjne przeszukiwanie lokalne. Adaptacyjne przeszukiwanie lokalne. wielkoskalowe przeszukiwanie sąsiedztwa. Symulowane wyżarzanie i pochodne algorytmy. Przeszukiwanie Tabu. Pamięć długoterminowa. Hiper-heurystyki

Zakres przedmiotu

- Algorytmy populacyjne i inspirowane biologicznie.
 - Przykłady metod. Algorytmy genetyczne. Algorytmy ewolucyjne. Krzyżowanie i rekombinacja. Pojęcie i rola schematów. Metody selekcji. Sposoby kodowania rozwiązań. Kodowanie pośrednie. Hybrydowe algorytmy ewolucyjne. Hiper-heurystyki genetyczne. Algorytmy kolonii mrówek.
- Sposoby uwzględniania ograniczeń.
- Podstawy teoretyczne
 - Twierdzenie "No free lunch,,. Miary trudności zagadnień optymalizacji. Analiza krajobrazu funkcji celu
- Projektowanie inteligentnych metod optymalizacji dla konkretnych problemów.
 - Schematy postępowania. Przykłady.
- Sposoby eksperymentalnej oceny inteligentnych metod optymalizacji.
- Aktualne trendy rozwojowe

Świeży projekt wykładowcy

<https://www.sintef.no/projectweb/top/vrptw/1000-customers/>

You are here: TOP / VRPTW / 1000 customers	
NEARP / MCGRP	1000 customers
PDPTW	
▼ VRPTW	
Documentation	Here you find instance definitions and the best known solutions (to our knowledge) for the 1000 customer instances of Gehring & Homberger's extended VRPTW benchmark. The version reported here has a hierarchical objective: 1) Minimize number of vehicles 2) Minimize total distance. Distance and time should be calculated with double precision, total distance results are rounded to two decimals. Exact methods typically use a total distance objective and use integral or low precision distance and time calculations. Hence, results are not directly comparable.
Solomon benchmark	
25 customers	
50 customers	
100 customers	Instance definitions (text)
Gehring & Homberger benchmark	Here you find a zip file with the 1000 customer instances .
200 customers	Best known results for Gehring & Homberger's 1000 customer instances
400 customers	
600 customers	The instance names in blue are hyperlinks to files with corresponding detailed solutions. They have all been checked by our solution checker. Note that many best known solutions do not have a reference to a peer reviewed publication. For these, important details on the solution algorithm, the computing time, and the experimental platform are probably not available. Further, there is no guarantee that the solutions have been produced without using external information, such as detailed solutions published earlier. We may later introduce two categories: 'properly published' and 'freestyle', the latter with no restrictions.
800 customers	
1000 customers	
Contact information	

Instance	Vehicles	Distance	Reference	Date
c1_10_1q	100	42478.95	GH	2001
c1_10_2	90	42222.96	CAINIAO	Oct-18
c1_10_3	90	40101.36	Q	17-sep-15
c1_10_4	90	39468.60	Q	17-apr-13
c1_10_5i	100	42469.18	RP	25-feb-05
c1_10_6	99	43830.21	Q	05-sep-14
c1_10_7	97	43368.79	SCR	19-dec-21
c1_10_8	92	42629.91	CAINIAO	Mar-20
c1_10_9	90	40322.37	SCR	19-dec-21
c1_10_10	90	39852.44	SCR	10-sep-18

c2_10_1s	30	16879.24	LL	2001
c2_10_2b	29	17126.39	NBD	2009
c2_10_3	28	16829.47	CAINIAO SCR	Tie, Nov-18
c2_10_4	28	15607.48	SCR	Nov-18
c2_10_5	30	16561.29	VCGP	31-jul-12
c2_10_6	29	16863.71	SCR	Oct-18
c2_10_7	29	17602.84	SCR	Jun-19
c2_10_8	28	16512.43	SCR	Oct-18
c2_10_9	28	17809.34	SCR	19-dec-21
c2_10_10	28	15937.45	SCR	Oct-18

Q - Quintiq. <http://www.quintiq.com/optimization/vrptw-world-records.html>.

RP - S. Ropke & D. Pisinger. "A general heuristic for vehicle routing problems", technical report, Department of Computer Science, University of Copenhagen.

SCR - Piotr Sielski (piotr.sielski@wmii.uni.lodz.pl), Piotr Cybula, Marek Rogalski (marek.rogalski@wmii.uni.lodz.pl), Mariusz Kok, Piotr Beling, Andrzej Jaskiewicz, Przemysław Pełka, Emapa S.A (<http://www.emapa.pl>), "New methods of VRP problem optimization", unpublished research funded by The National Centre for Research and Development. project number: POIR.01.01.01.-00-0222/16.

WA - Rüdiger Wagemaker. MSc thesis in progress. Tilburg University.

WW - Witosław Wierzbicki. "Design and implementation of parallel programs with shared memory". Master of Science Thesis, University of Silesia, Sosnowiec, 2012.

VCGP - T. Vidal, T. G. Crainic, M. Gendreau, C. Prins. "A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows", *Computers & Operations Research*, Vol. 40, No. 1. (January 2013), pp. 475-489.

Elementy zagadnienia optymalizacji

- Cel(e) optymalizacji – funkcja celu i kierunek optymalizacji – minimalizacja/maksymalizacja
 - Cele w przypadku optymalizacji wielokryterialnej (poza zakresem przedmiotu)
- Przestrzeń decyzyjna
- Ograniczenia na dopuszczalne decyzje
- Sposób oceny wpływu decyzji na cel optymalizacji – sposób obliczania funkcji celu

Formułowanie zagadnienia optymalizacji

- Podejście deklaratywne:
 1. Jaki cel chcemy osiągnąć?
 2. Jakie decyzje możemy podjąć?
 3. Jakie ograniczenia musimy uwzględnić?
 4. Jak obliczyć wpływ naszych decyzji na nasz cel?

Przykład – marszrutyzacja pojazdów (vehicle routing)

1. Cel: minimalizacja kosztów rozwiezienia towarów do klientów
2. Możliwe decyzje: przydział klientów do pojazdów, kolejność obsługi/odwiedzin klientów
3. Ograniczenia: pojemność pojazdów, czas pracy pojazdów/kierowców, okna czasowe klientów...
4. Obliczane funkcji celu: analityczne, funkcja liniowa

Rozwój metod optymalizacji w ostatnich latach

- Ogromny rozwój metodyki pozwalający na rozwiązywanie coraz bardziej złożonych i większych zagadnień
- Integracja z systemami informatycznymi
- Wielopoziomowe związki ze sztuczną inteligencją – podstawy teoretyczne, optymalizacja w AI, AI w optymalizacji

Sformułowanie zagadnienia optymalizacji

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

przy ograniczeniach (p.o)

$$\mathbf{x} \in S$$

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

jest równoważne

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

Rozwiązanie optymalne

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

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

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

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

Nazywane też rozwiązaniem **globalnie** optymalnym, choć ten przymiotnik formalnie jest zbędny

Nie musi być unikalne, jeżeli więcej rozwiązań spełnia powyższy warunek

Programowanie matematyczne

- Zagadnienie optymalizacji może być zdefiniowane jako problem programowania matematycznego, jeżeli:
rozwiązanie jest zdefiniowane jako wektor zmiennych:

$$\mathbf{x} \in R^n, \mathbf{x} = \{x_1, \dots, x_n\}$$

a ograniczenia jako zbiór równości/nierówności matematycznych:

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

Klasyfikacja problemów programowania matematycznego

- Funkcja celu i ograniczenia są liniowe \Rightarrow
 - problem liniowego programowania matematycznego (linear programming)
- Funkcja celu lub co najmniej jedno ograniczenie są nieliniowe \Rightarrow
 - problem nieliniowego programowania matematycznego (non-linear programming)
- Zmienne ciągłe \Rightarrow
 - problem ciągłego (liniowego/nieliniowego) programowania matematycznego (continuous programming)
- Zmienne dyskretne \Rightarrow
 - problem dyskretnego programowania matematycznego (discrete programming)
- Zmienne mieszane ciągłe i dyskretne \Rightarrow
 - problem mieszanego programowania matematycznego (mixed discrete programming)
- Zmienne całkowitoliczbowe \Rightarrow
 - problem całkowitoliczbowego programowania matematycznego (integer programming)
- Zmienne binarne \Rightarrow
 - problem binarnego programowania matematycznego (binary programming)

Możliwe różnorodne kombinacje, np. problem całkowitoliczbowego liniowego programowania matematycznego , problem binarnego nieliniowego programowania matematycznego ...

Solvery programowania matematycznego

- Problemy programowania matematycznego mogą być rozwiązywane za pomocą standardowy solverów
- Efektywne solvery istnieją głównie dla problemów:
 - liniowego programowania matematycznego (LP)
 - mieszanego liniowego programowania matematycznego (MIP)
 - np. Gurobi, Cplex, Frontline, Lindo, Matlab...

Solvery programowania matematycznego - postęp w ostatnich latach

- Metoda punktu wewnętrznego (interior point)
 - Inspirowana metodami programowania nieliniowego
 - Lepsza od metody sympleksowej dla dużych problemów
- Preprocessing – usuwanie zbędnych (zawsze nieaktywnych) ograniczeń i zmiennych
 - np.: $x_1 + x_2 \leq 5, x_2 + x_3 \leq 5 \Rightarrow \cancel{x_1 + x_3 \leq 5}$
- Branch and price – połączenie branch and bound z generowaniem kolumn
- Branch and cut – połączenie branch and bound z cutting planes (płaszczyzn odcinających)
- Matheuristics – połączenie solverów z metaheurystykami
- Optymalizacja obliczeń (np. równoległość)

Znaczenie modelowania problemu

- Np. funkcja (skalaryzującą Czebyszewa)
 - minimalizuj $\max \left(\lambda_j \left(z_j^0 - f_j(x) \right) \right), j = 1, \dots, p$
 - p.o.
 - $x \in S$
- Wersja liniowa:
 - minimalizuj ε
 - p.o.
 - $\varepsilon \geq \lambda_j \left(z_j^0 - f_j(x) \right), j = 1, \dots, p$
 - $x \in S$

Zagadnienia optymalizacji kombinatorycznej

- Dyskretny i skończony zbiór rozwiązań
- Struktura kombinatoryczna (zbiory, permutacje, drzewa, grafy...)
- Zagadnienie optymalizacji kombinatorycznej zawsze da się zdefiniować jako problem programowania matematycznego (dyskretnego, binarnego). Często wiąże się z szybkim wzrostem liczby zmiennych i ograniczeń (np. n^2 zmiennych dla TSP)
- Pozwala to na zastosowanie uniwersalnych solverów programowania matematycznego, ale, w ogólności, nie jest niezbędne przy stosowaniu inteligentnych metod optymalizacji

Pojęcia problemu i instancji problemu optymalizacji kombinatorycznej

- Problem optymalizacji kombinatorycznej to zbiór instancji definiowanych wg. pewnego schematu
- Instancja to konkretne zagadnienia optymalizacji - funkcja celu, przestrzeń decyzyjna, ograniczenia, kierunek optymalizacji
- Instancja powstaje poprzez ustalenie pewnych parametrów problemu, np. liczby wierzchołków i odległości pomiędzy wierzchołkami w problemie komiwojażera

Przykłady problemów optymalizacji kombinatorycznej

- Problem przydziału (assignment)
- Problem plecakowy (knapsack)
- Problem minimalnego drzewa rozpinającego (minimum spanning tree)
- Problem pokrycia zbioru (set covering problem)
- Problem komiwojażera (traveling salesperson problem - TSP)
- Problem marszrutyżacji pojazdów (vehicle routing problem - VRP)
- Problem kolorowania grafu
- Grupowanie (clustering)
- ...

Problem przydziału (assignment)

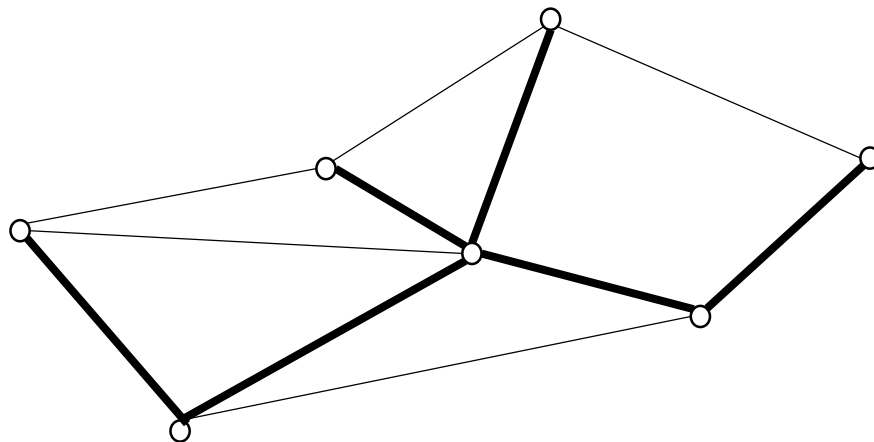
- Dany jest zbiór elementów i zadań. Każdy element może być przydzielony do każdego zadania. Przydział każdego elementu do zadania wiąże się z pewnym kosztem. Do każdego zadania można przydzielić jeden element. Należy znaleźć przydział elementów do zadań minimalizujący łączny koszt.
- Np. przydział pracowników do zadań

Problem plecakowy (knapsack)

- Dany jest zbiór elementów o danej wadze i wartości. Należy wybrać elementy do umieszczenia w plecaku, tak aby nie przekroczyć pojemności plecaka i maksymalizować łączną wartość wybranych elementów
- Np.
 - Złodziej wybierający fanty do zabrania
 - Inwestor wybierający składniki portfela inwestycyjnego

Problem minimalnego drzewa rozpinającego (minimum spanning tree)

- Należy znaleźć drzewo rozpinające (zbiór krawędzi łączących wszystkie wierzchołki) w grafie ważonym o minimalnej łącznej wadze
- Np. planowanie połączeń telekomunikacyjnych o minimalnym łącznym koszcie łączących wszystkie wierzchołki

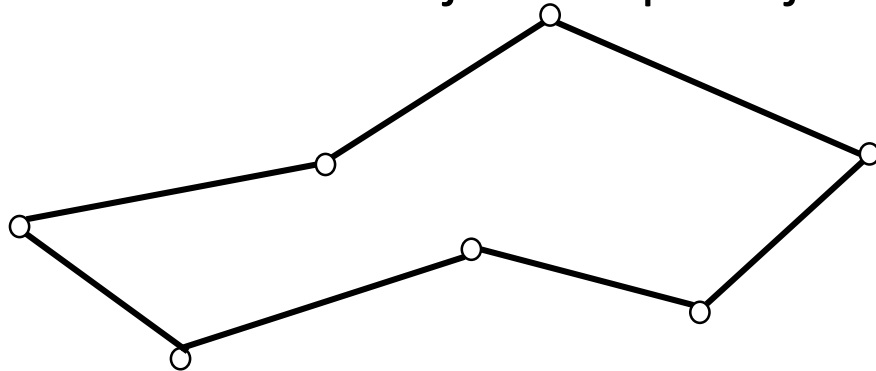


Problem pokrycia zbioru (set covering problem)

- Dany jest zbiór elementów i zestaw podzbiorów tych elementów. Każdy podzbiór ma pewien koszt. Należy pokryć wszystkie elementy wybierając podzbiory o minimalnym łącznym koszcie.

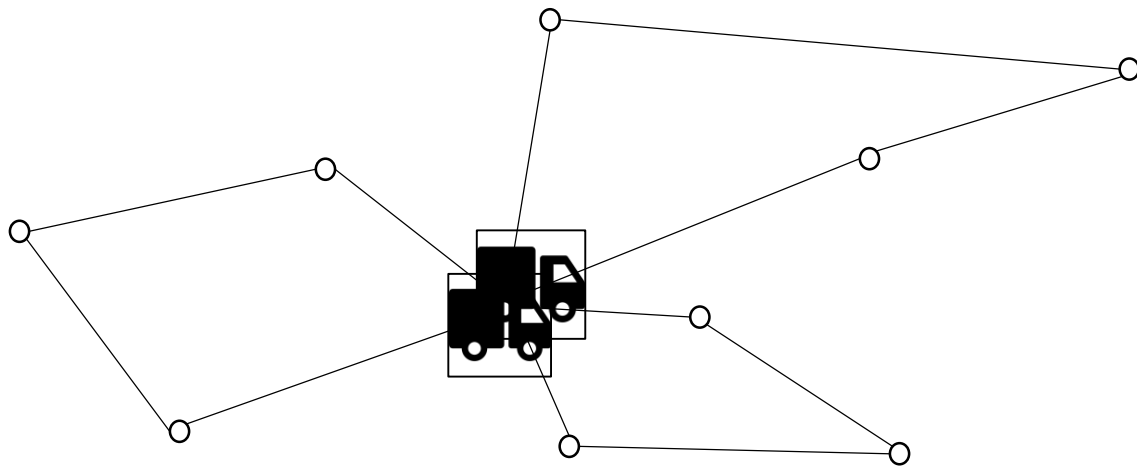
Problem komiwojażera (traveling salesperson problem - TSP)

- Dany jest zbiór wierzchołków (miast) i odległości pomiędzy każdą parą wierzchołków (pełen graf ważony). Należy znaleźć cykl Hamiltona (ścieżkę zamkniętą przechodzącą przez wszystkie wierzchołki) o minimalnej łącznej długości
- Np.:
 - Częsty element praktycznych problemów transportowych
 - Planowanie kolejności operacji robotów



Problem marszrutyzacji pojazdów (vehicle routing problem)

- Dany jest zbiór wierzchołków wraz z odległościami i parametry floty pojazdów (np. pojemność, okna czasowe), w tym wierzchołek bazowy. Należy znaleźć zbiór tras odwiedzających wszystkie wierzchołki, zaczynających i kończących się w bazie, spełniających wszystkie ograniczenia o minimalnej łącznej długości



Problem kolorowania grafu

- Należy pokolorować (przydzielić etykiety) wierzchołkom niepełnego grafu, tak aby wierzchołki o tym samym kolorze nie były połączone krawędzią, używając jak najmniejszej liczby kolorów
- Np. przydział częstotliwości do anten w sieciach komórkowych (krawędź oznacza ryzyko interferencji)

Grupowanie

- Należy podzielić zbiór elementów dla których znana jest pewna miar podobieństwa lub odległości minimalizując pewną miarę jakości grupowania, np. stosunek średniej odległości rozwiązań umieszczonych w jednej grupie do średniej odległości rozwiązań umieszczonych w różnych grupach

Wersje problemów

- Praktycznie każdy problem może mieć różne wersje
- Np. dla problemów komiwojażera i marszrutyzacji pojazdów można rozważać:
 - Okna czasowe
 - Ograniczenia na czas pracy pojazdów
 - Ograniczenia kolejnościowe
 - Heterogeniczną lub homogeniczną flotę pojazdów
 - Grafy skierowane lub nie – problemy symetryczne lub asymetryczne
 - ...

Obszary zastosowań optymalizacji kombinatorycznej

- Transport, logistyka, planowanie produkcji, harmonogramowanie projektów i czasu pracy (scheduling), robotyka, telekomunikacja, sztuczna inteligencja i maszynowe uczenie, projektowanie inżynierskie, medycyna, optymalizacja wykorzystania zasobów (w tym informatycznych), inżynieria oprogramowania, marketing, finanse/inwestycje...

Źródła trudności zagadnień optymalizacji

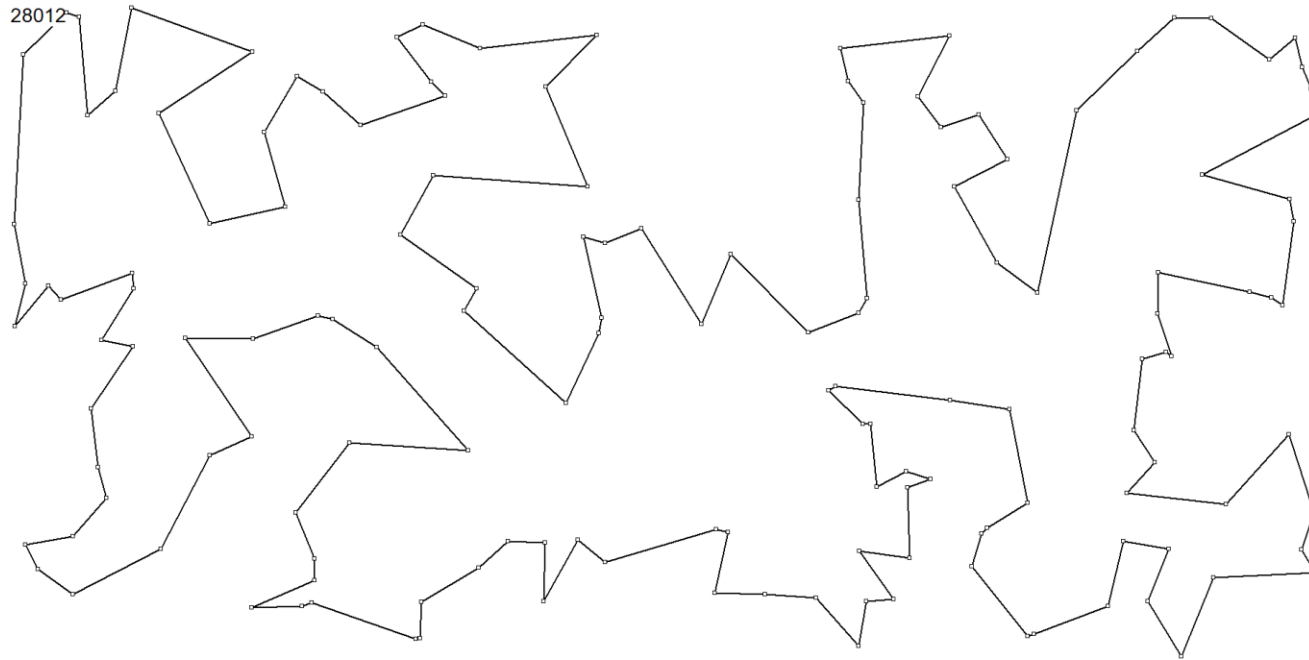
- Duża liczba rozwiązań
- Ograniczenia
- Złożoność obliczeniowa
- Niepewność (nie rozważana w ramach przedmiotu)

Duża liczba rozwiązań

- Problem plecakowy z n elementami:
 - Liczba podzbiorów 2^n , chociaż mniej dopuszczalnych i mniej spełniających warunki optymalności (tj. brak możliwości dodania elementu do plecaka)
 - Np. $n = 100$ to $1,27 * 10^{30}$ rozwiązań
- Problem przydziału z n elementami i m zadaniami
 - Może być traktowany jak macierz $n \times m$ z wyborem poszczególnych przydziałów, więc 2^{nm} , chociaż mniej dopuszczalnych
- Symetryczny problem komiwojażera z n wierzchołkami
 - Liczba permutacji $n - 1$ elementów (wybór pierwszego nie ma znaczenia) $(n - 1)!$
 - Np. $n = 20$ to $2,4 * 10^{18}$ rozwiązań, $n = 100$ to $9,3 * 10^{157}$ rozwiązań
- Z drugiej strony nieskończona liczba rozwiązań w problemach ciągłych może ułatwiać rozwiązanie

Problem komiwojażera ze 150 węzłami

- Problem komiwojażera ze 150 węzłami



Czy możemy rozwiązać taki problem?

- Liczba rozwiązań $(n-1)!$
- $(150-1)! = 3,8 \times 10^{260}$
- Wiek wszechświata w jednostkach czasu Plancka $\approx 10^{61}$
- Liczba cząstek w obserwowalnym Universe $\approx 10^{80}$
- Gdyby więc każda cząstka była komputerem i wygenerowała jedno rozwiązanie w czasie Plancka, moglibyśmy wygenerować do tej pory $\approx 10^{141}$ rozwiązań
- Zdecydowanie nie możemy rozwiązać tego problemu poprzez wyczerpujące przeszukiwanie

Ograniczenia

- Teoretycznie ułatwiają rozwiązanie, gdyż zmniejszają liczbę rozwiązań dopuszczalnych
- W praktyce często utrudniają
- W szczególności samo znalezienie rozwiązania dopuszczalnego może być bardzo trudne
- Trudniejsze staje się także przechodzenie pomiędzy rozwiązaniami, gdyż rozwiązania pośrednie mogą być niedopuszczalne

Złożoność obliczeniowa

- Problemy z klasy P – znane algorytmy efektywne o złożoności wielomianowej
 - Np. problem przydziału, minimalnego drzewa rozpinającego
- Problemy z klasy NP – znane są algorytmy o złożoności wykładniczej
- Otwarty problem czy $P \neq NP$
- Problemy NP-trudne można do nich przetransformować w czasie wielomianowych dowolny problem z klasy NP. NP-zupełne, jeżeli są jednocześnie w klasie NP.
 - Np. problem plecakowy, komiwojażera, kolorowania grafu
- Uwagi praktyczne:
 - Typowa złożoność obliczeniowa dotyczy najgorszego przypadku (instancji) (analiza, a nawet definicja przypadku średniego jest dużo trudniejsza). W praktyce problemy, nawet NP-trudne mogą się bardzo różnić zachowaniem w przypadku średnim. Dla pewnych problemów większość instancji może być dość łatwo rozwiązywana (np. problem plecakowy, komiwojażera), dla innych większość instancji wymaga wykładniczego czasu (np. kolorowanie grafu)
 - Złożoność wielomianowa nadal może być problematyczna dla wielomianów wyższego stopnia

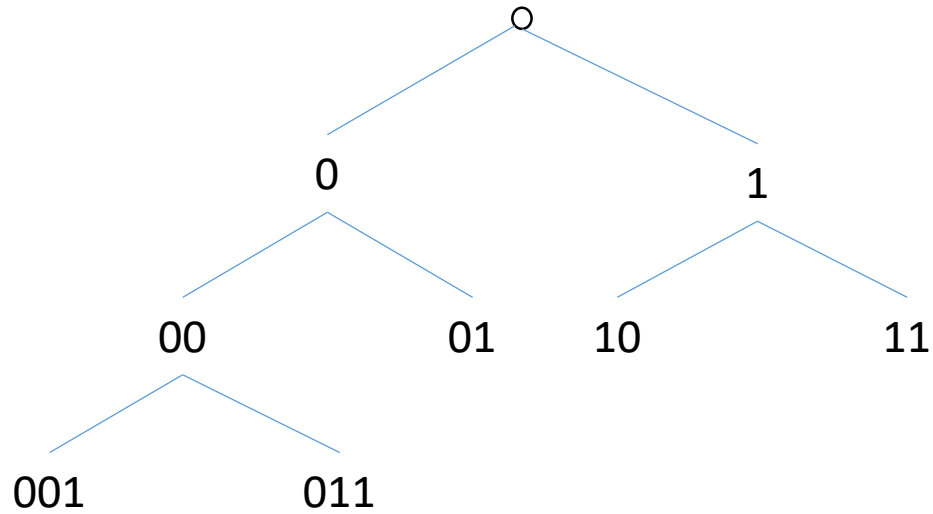
Klasyfikacja metod optymalizacji

- Metody dokładne (gwarantujące znalezienie optimum)
 - Pełen przegląd, metody podziału i ograniczeń (branch and bound), metody (i solvery) programowania matematycznego, metody dedykowane – z reguły jakieś wersje B&B, programowanie dynamiczne
- Metody przybliżone, heurystyki
 - Losowe błędzenie, heurystyki dedykowane, zachłanne heurystyki konstrukcyjne, metaheurystyki (inteligentne heurystyki), metody dokładne z ograniczonym czasem
 - Mogą, lub nie, dawać jakieś gwarancje przybliżenia
- Można też dzielić na metody ogólne (uniwersalne, stosowalne do różnych problemów) i dedykowane dla konkretnych problemów, choć podział nie jest ostry. Dedykowane to z reguły jakieś adaptacje ogólnych schematów

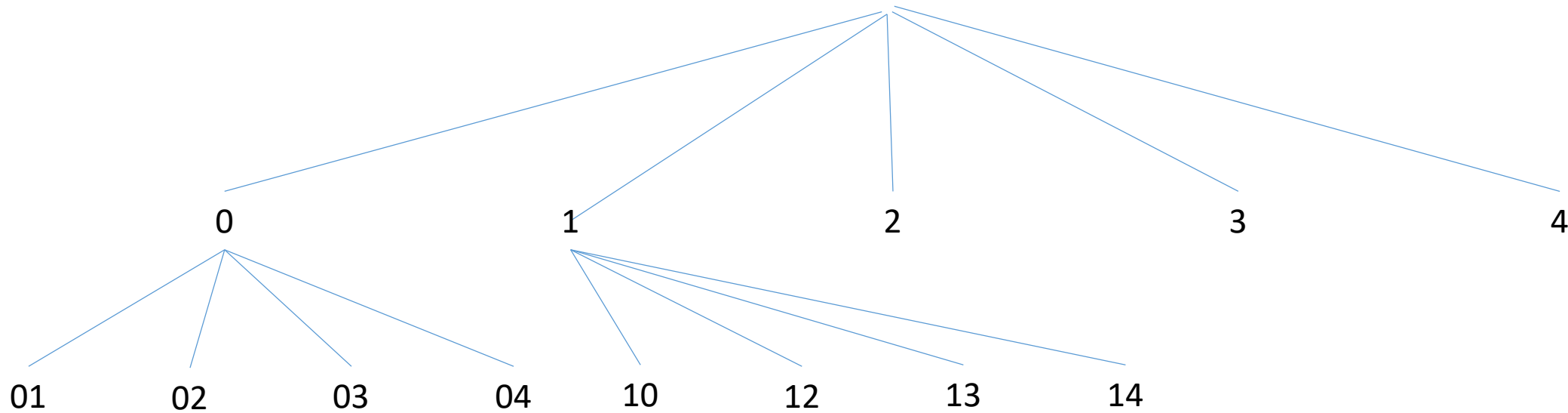
Pełen przegląd (exhaustive search, brute force)

- Generujemy i obliczamy wartość funkcji celu dla wszystkich rozwiązań
- Trudne, lub praktycznie niewykonalne przy dużej liczbie rozwiązań
- Potrzebny systematyczny sposób generowania wszystkich rozwiązań (najlepiej bez powtórzeń)
- Np. dla ciągów binarnych (problem przydziału, plecakowy) dekodowanie kolejnych liczb całkowitych
 - 0 000
 - 1 001
 - 2 010
 - 3 011
 - 4 100 (czy da się uniknąć zmiany więcej niż jednej cyfry?)
- Często organizowane jako przeglądanie drzewa

Drzewo pełnego przeglądu dla ciągów binarnych (pr. przydziału, plecakowy)



Drzewo pełnego przeglądu dla permutacji (np. problem komiwojażera)



Metoda podziału i ograniczeń

- Modyfikacja przeglądu drzewa
- Przed rozwinięciem gałęzi obliczana jest wartość dolnego lub górnego ograniczenia optymalnej wartości funkcji celu w tej gałęzi (lower/upper bound) – zakładamy, że można to zrobić znacznie szybciej niż rozwinąć całą gałąź
- Jeżeli dolne/górne ograniczenie jest gorsze od najlepszej już znalezionej wartości, to gałąź nie jest rozwijana (na pewno nie ma w niej lepszego rozwiązania)
- Jak znaleźć dolne/górne ograniczenie
 - Np. dla problemu plecakowego można szybko znaleźć optimum dla problemu zrelaksowanego (możemy wybierać ułamkowe części elementów) – górne ograniczenie dla oryginalnego problemu binarnego
 - Np. dla problemu komiwojażera dolne przybliżenie daje minimalne drzewo rozpinające
- Duże znaczenie dla efektywności może mieć kolejność rozwijania gałęzi, najlepiej szybko poprawiać najlepsze rozwiązanie
- Jeżeli przerywamy przed rozwinięciem wszystkich nieodciętych gałęzi, metoda staje się heurystyką. Wtedy kolejność rozwijania gałęzi wpływa także na jakość rozwiązań.

Złożoność black-box search na komputerach kwantowych

- Na komputerach klasycznych $O(|S|)$ – jedynym wyjściem jest pełen przegląd
- Na komputerach kwantowych przy zastosowaniu rozszerzeń algorytmu Grovera $\Theta(\sqrt{|S|})$
 - Nie istnieje lepszy algorytm kwantowy
 - Złożoność pozostaje wykładnicza dla wykładniczego rozmiaru przestrzeni rozwiązań (nie oznacza więc, że komputery kwantowe pozwalają efektywnie rozwiązywać problemy NP-zupełne)
 - 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>
 - Scott Aaronson. 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 (random search)

powtarzaj

wygeneruj i oceń (oblicz wartość funkcji celu) losowe rozwiązanie

Dopóki nie są spełnione warunki stopu

Zwróć najlepsze znalezione rozwiązanie

Losowe błędzenie (random walk)

wygeneruj i oceń losowe rozwiązanie

powtarzaj

 Zmodyfikuj i oceń w sposób losowy bieżące rozwiązanie

Dopóki nie są spełnione warunki stopu

Zwróć najlepsze znalezione rozwiązanie

- Losowa modyfikacja może być szybsza niż generowanie losowego rozwiązania od podstaw
- Próbkowanie przestrzeni rozwiązań jest mniej równomierne

Heurystyki/metody konstrukcyjne

- Rozwiązanie problemu optymalizacji jest konstruowane krokowo poprzez dokładanie kolejnych elementów (lub, ogólnie, inne operacje jednokierunkowe, np. łączenie tras w heurystyce Clarke'a-Wrighta dla VRP)
- Np.
 - Dodawanie kolejnych elementów do niepełnego rozwiązania problemu plecakowego
 - Dodawanie kolejnych wierzchołków/krawędzi do niepełnego rozwiązania problemu komiwojażera
- Jeżeli kolejne elementy są wybierane losowo, to jest to sposób na tworzenie losowego rozwiązania

Zachłanne heurystyki konstrukcyjne (greedy heuristics)

- Powstają, jeżeli kolejne elementy wybierane są tak, aby optymalizować bieżącą zmianę wartości funkcji celu
- Np.
 - Dodawanie do plecaka elementu o najlepszym stosunku wartości do wagi
 - Dodawanie najbliższego wierzchołka do rozwiązania problemu komiwojażera

Zachłanne heurystyki konstrukcyjne (greedy heuristics)

Utwórz niepełne rozwiązanie początkowe, np. $\mathbf{x} := \emptyset$

powtarzaj

dodaj do \mathbf{x} lokalnie najlepszy element nie wchodzący jeszcze w skład rozwiązania

dopóki nie utworzono pełnego rozwiązania

Heurystyka najbliższego sąsiada (nearest neighbor) dla problemu komiwojażera

wybierz (np. losowo) wierzchołek startowy

powtarzaj

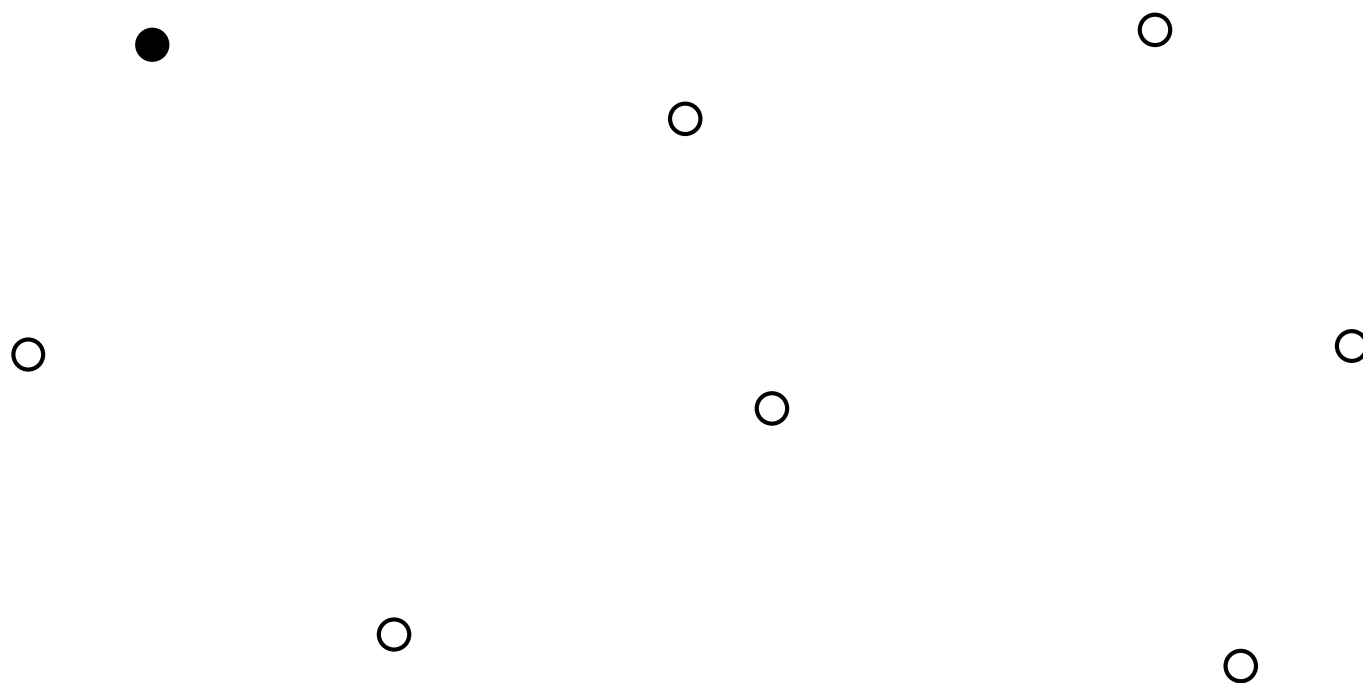
dodaj do rozwiązania wierzchołek (i prowadzącą do niego krawędź) najbliższy ostatnio dodanemu

dopóki nie zostały dodane wszystkie wierzchołki

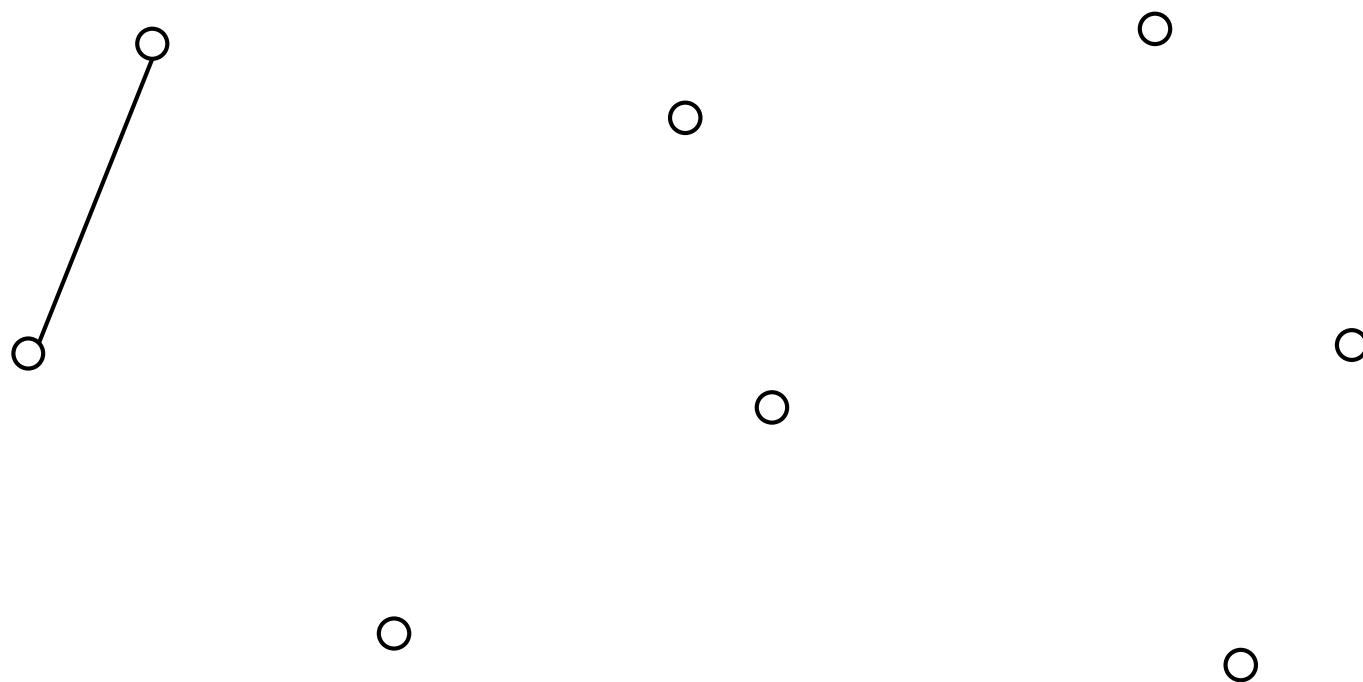
Dodaj krawędź z ostatniego do pierwszego wierzchołka

Istnieje też inna wersja, w rozważa się wszystkie możliwe miejsca wstawienia kolejnych wierzchołków (nie tylko na końcu)

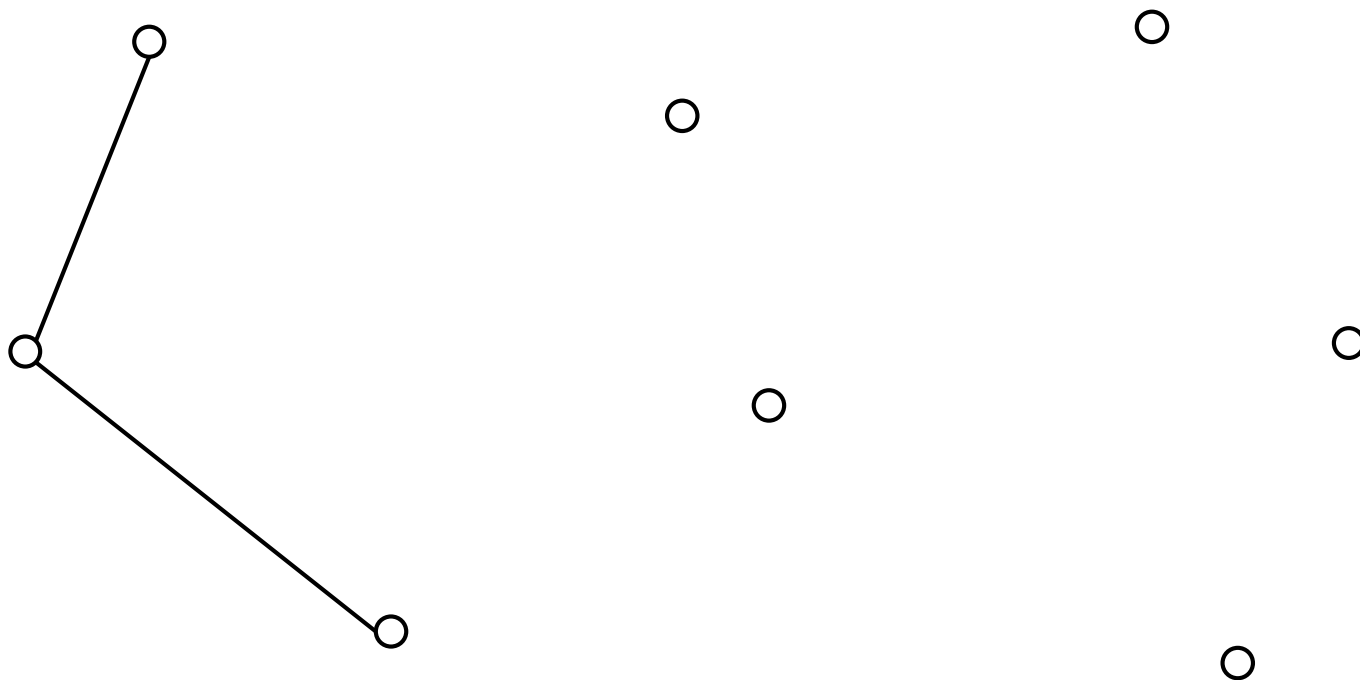
Heurystyka najbliższego sąsiada dla problemu komiwojażera



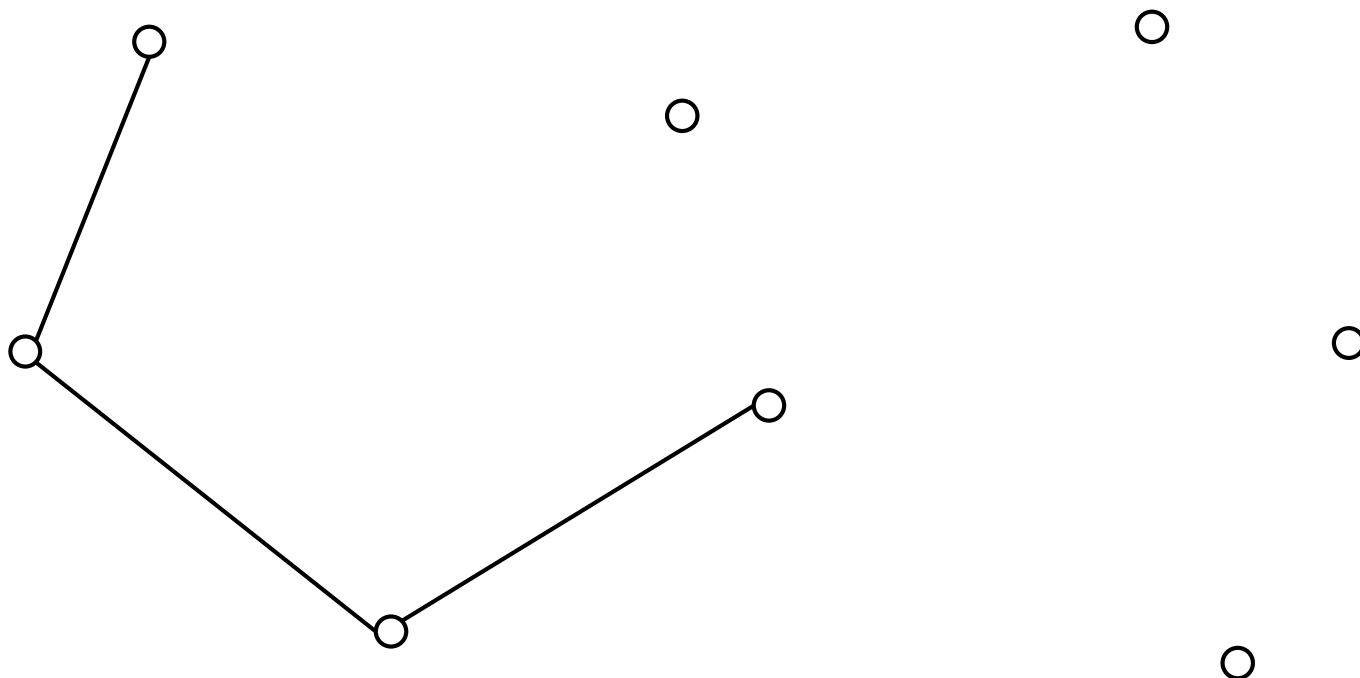
Heurystyka najbliższego sąsiada dla problemu komiwojażera



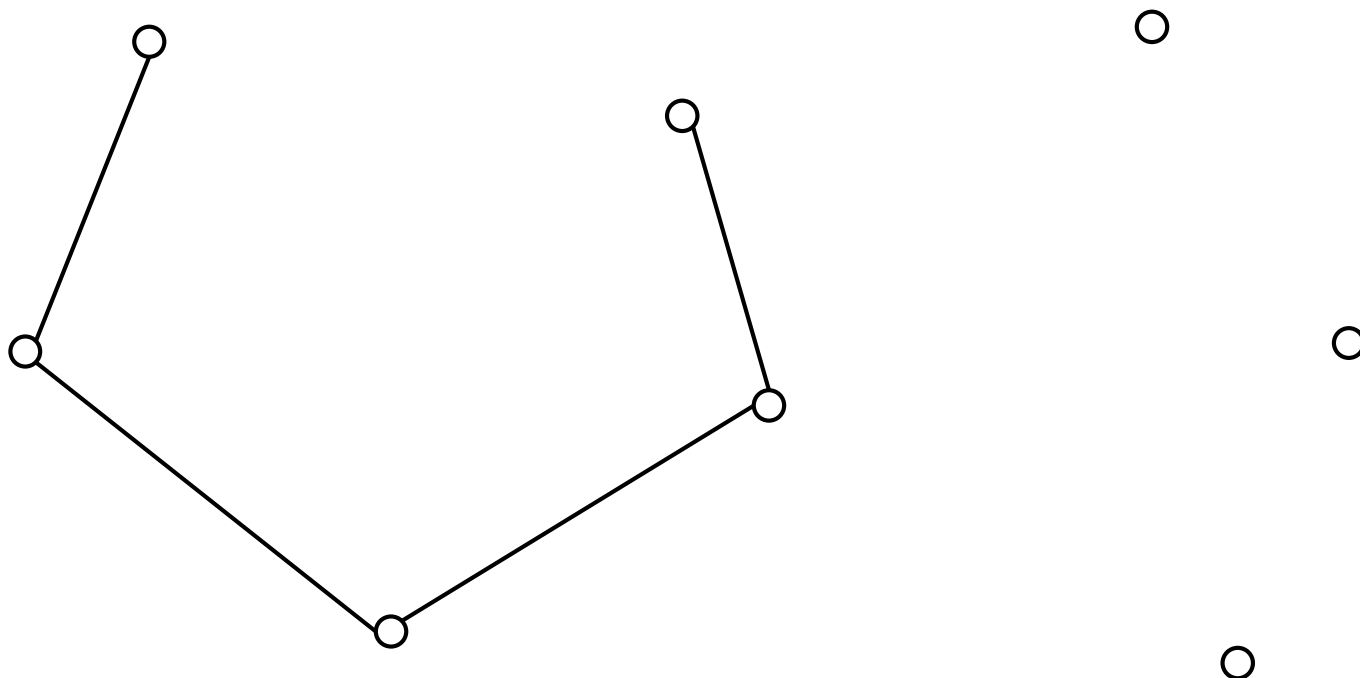
Heurystyka najbliższego sąsiada dla problemu komiwojażera



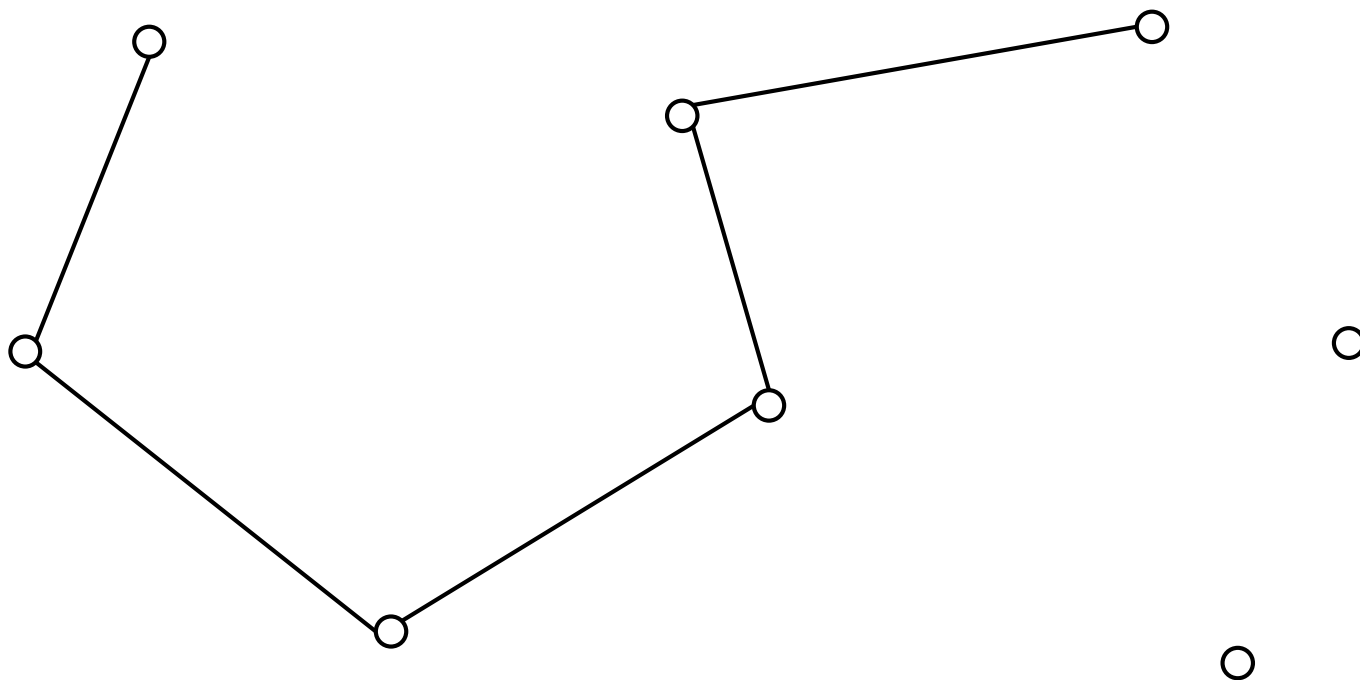
Heurystyka najbliższego sąsiada dla problemu komiwojażera



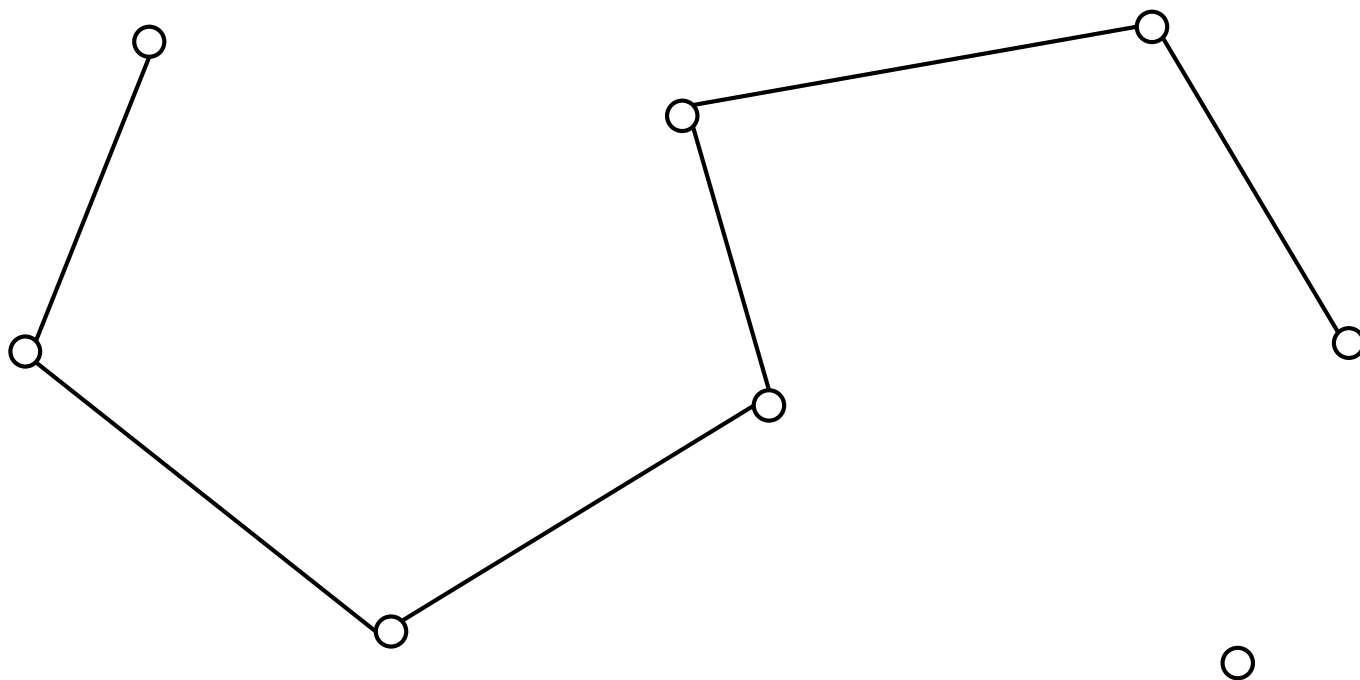
Heurystyka najbliższego sąsiada dla problemu komiwojażera



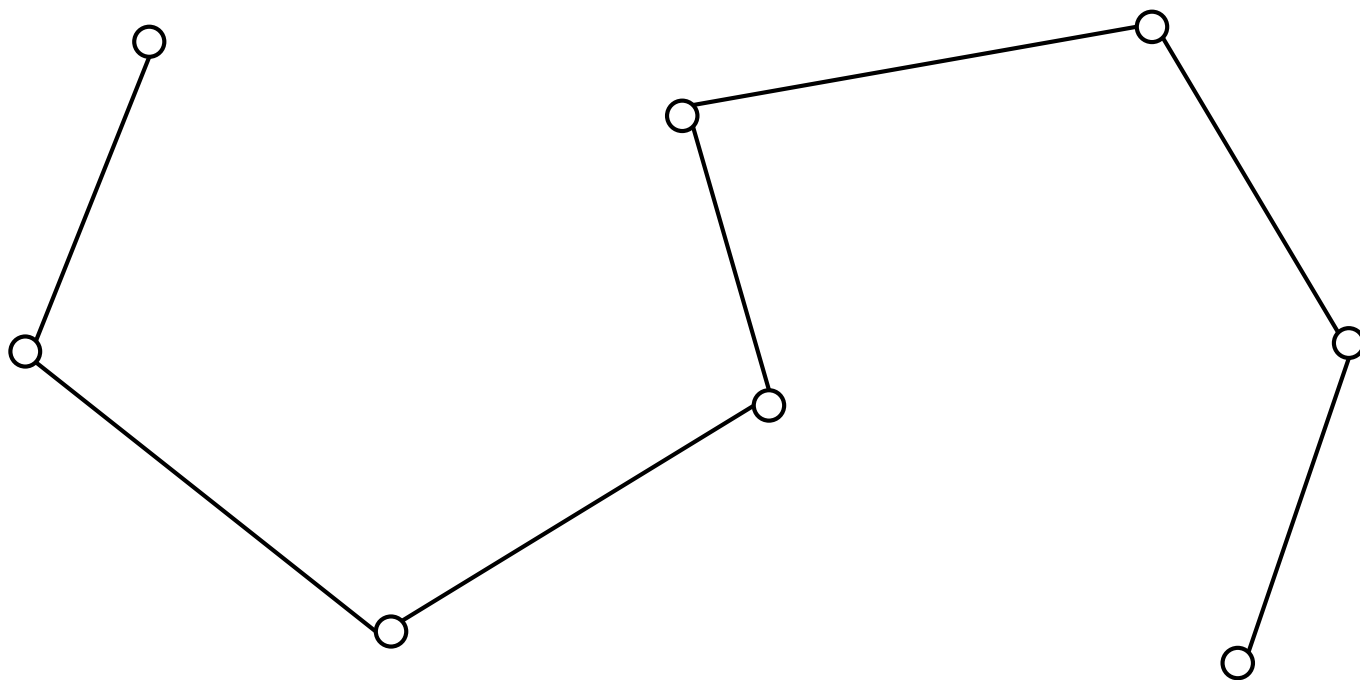
Heurystyka najbliższego sąsiada dla problemu komiwojażera



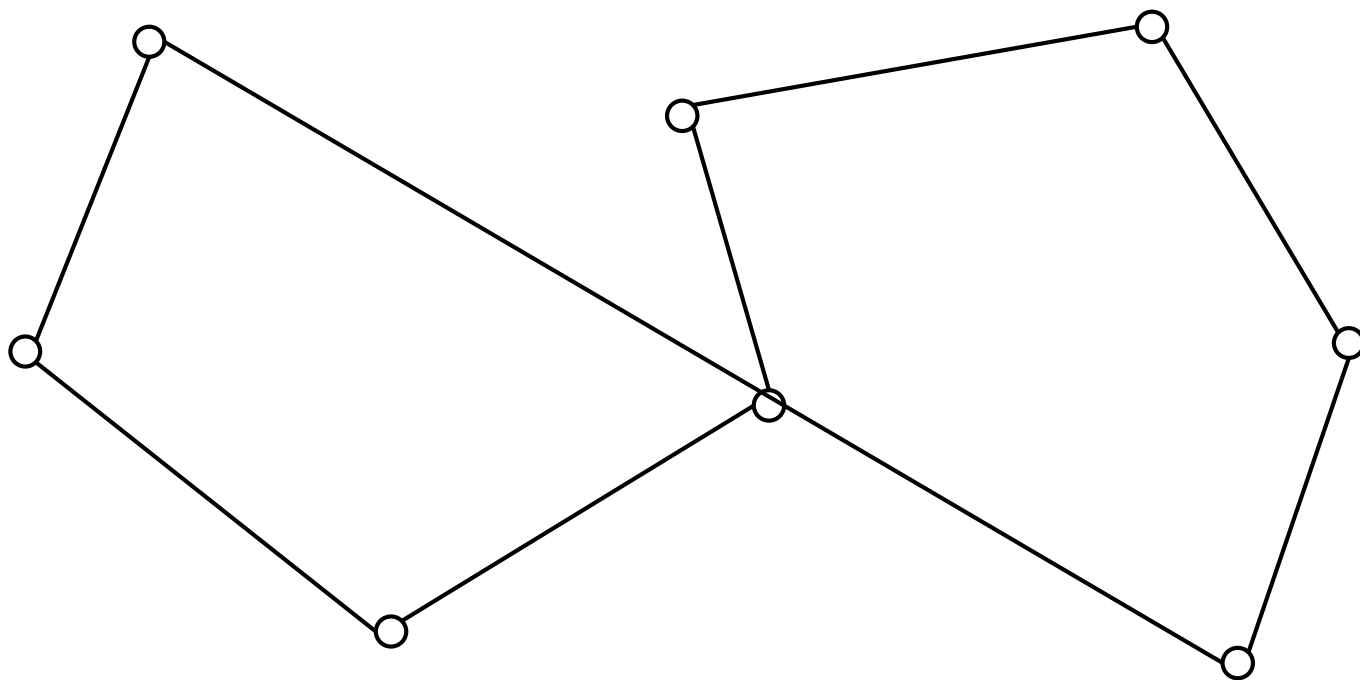
Heurystyka najbliższego sąsiada dla problemu komiwojażera



Heurystyka najbliższego sąsiada dla problemu komiwojażera



Heurystyka najbliższego sąsiada dla problemu komiwojażera



Metoda rozbudowy cyklu (greedy cycle)

wybierz (np. losowo) wierzchołek startowy

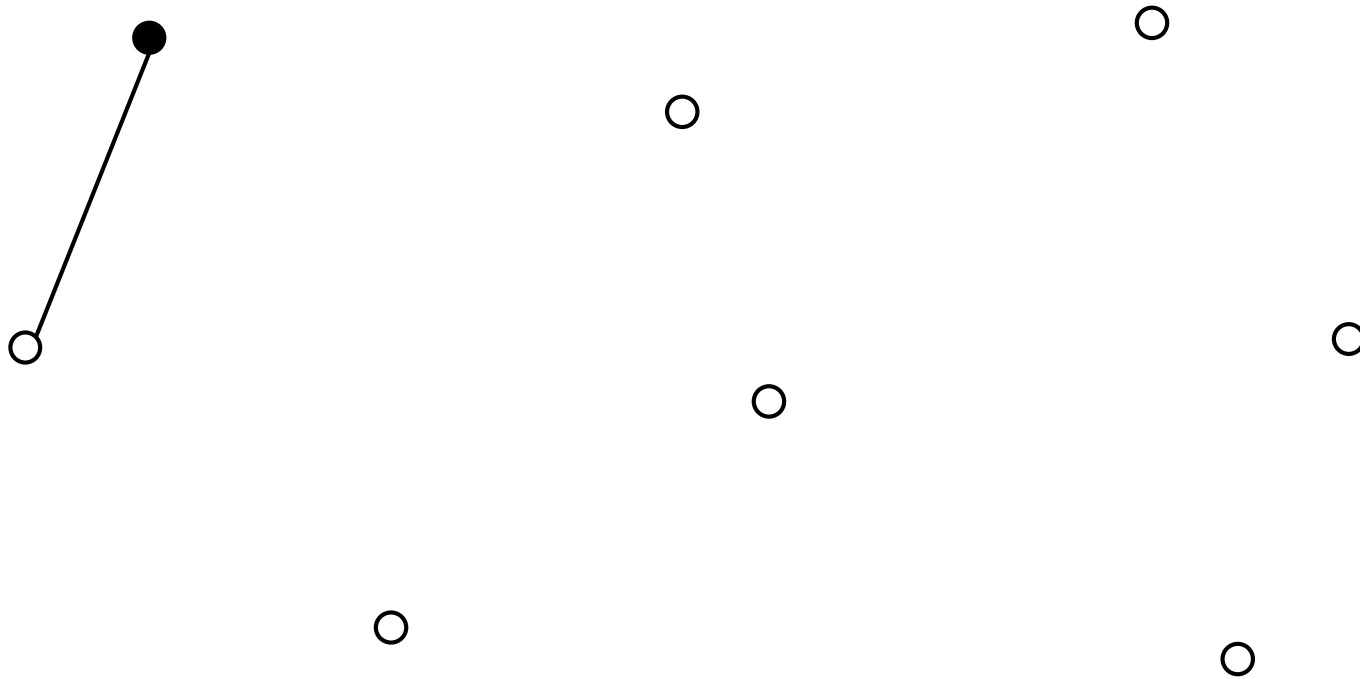
wybierz najbliższy wierzchołek i stwórz z tych dwóch wierzchołków niepełny cykl

powtarzaj

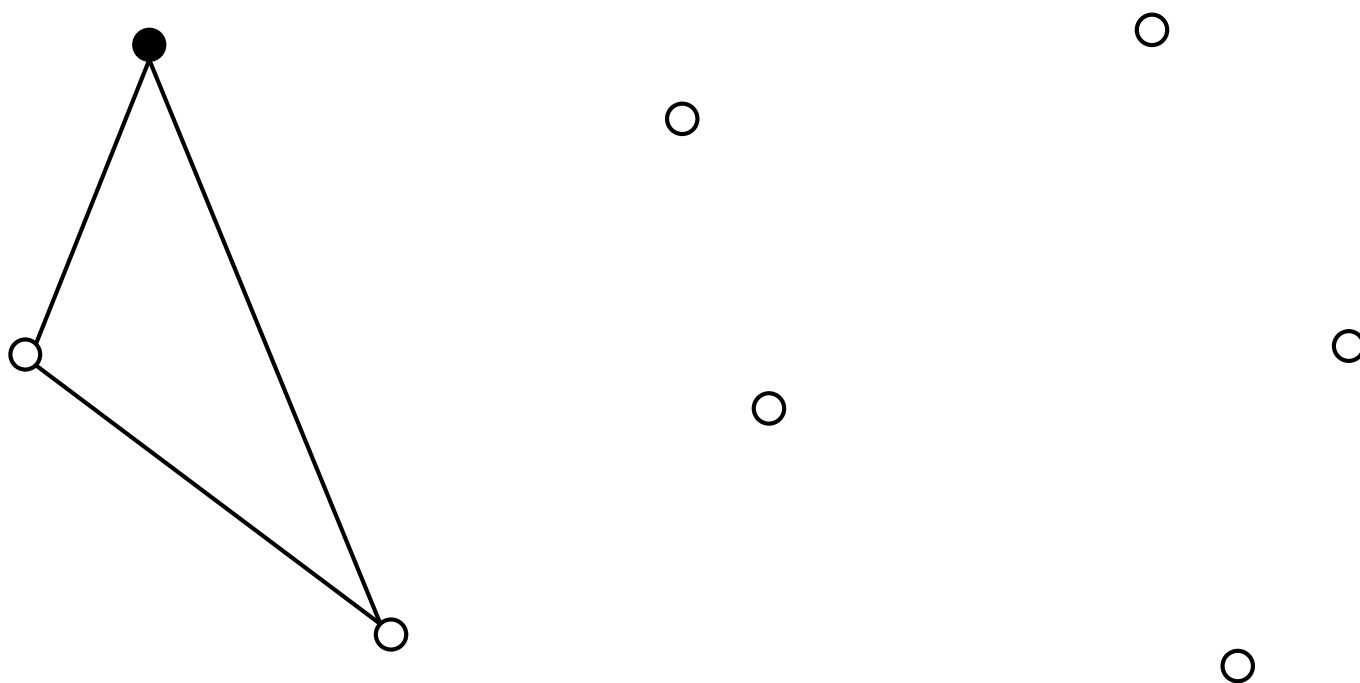
 wstaw do bieżącego cyklu w najlepsze możliwe miejsce
 wierzchołek powodujący najmniejszy wzrost długości cyklu

dopóki nie zostały dodane wszystkie wierzchołki

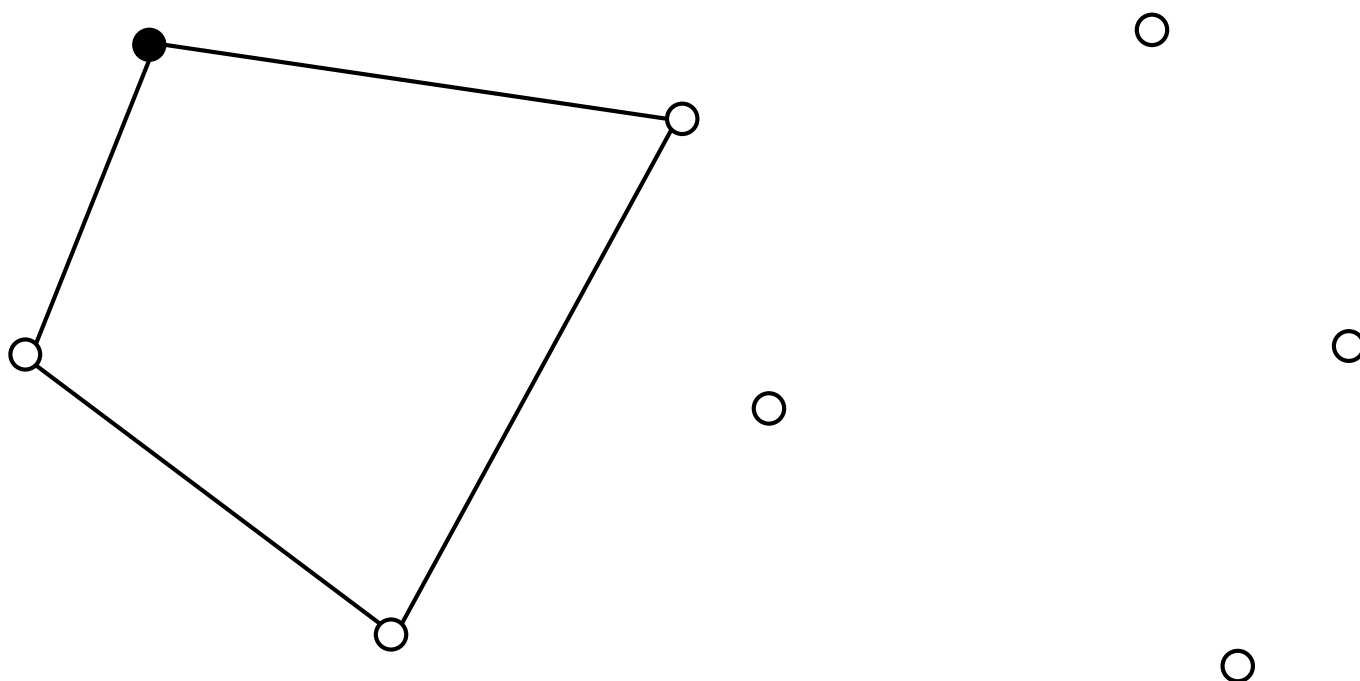
Metoda rozbudowy cyklu (greedy cycle)



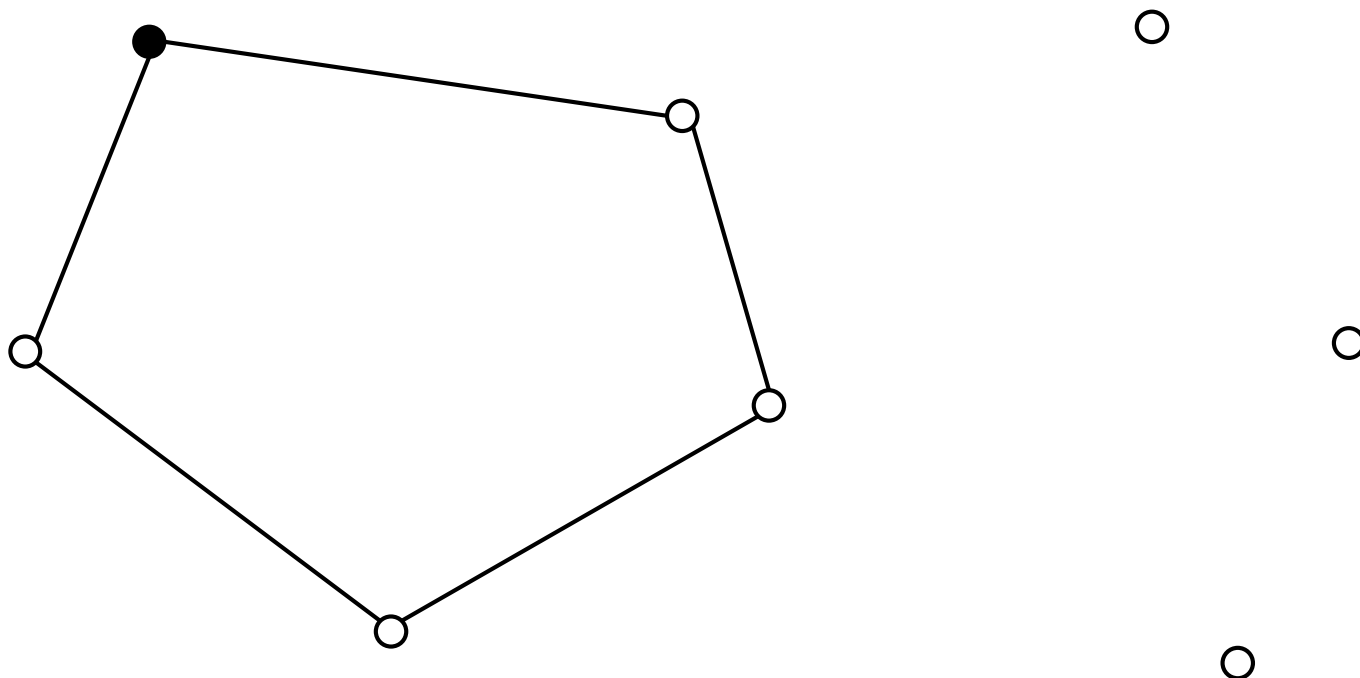
Metoda rozbudowy cyklu (greedy cycle)



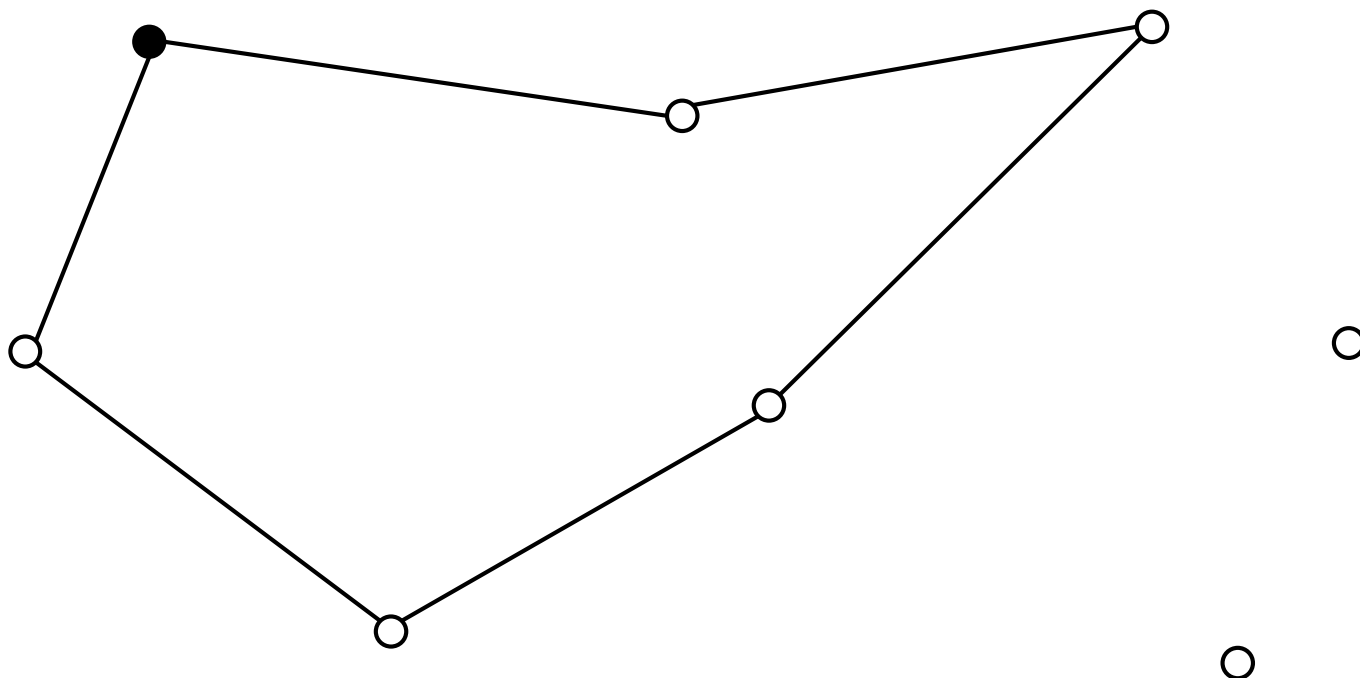
Metoda rozbudowy cyklu (greedy cycle)



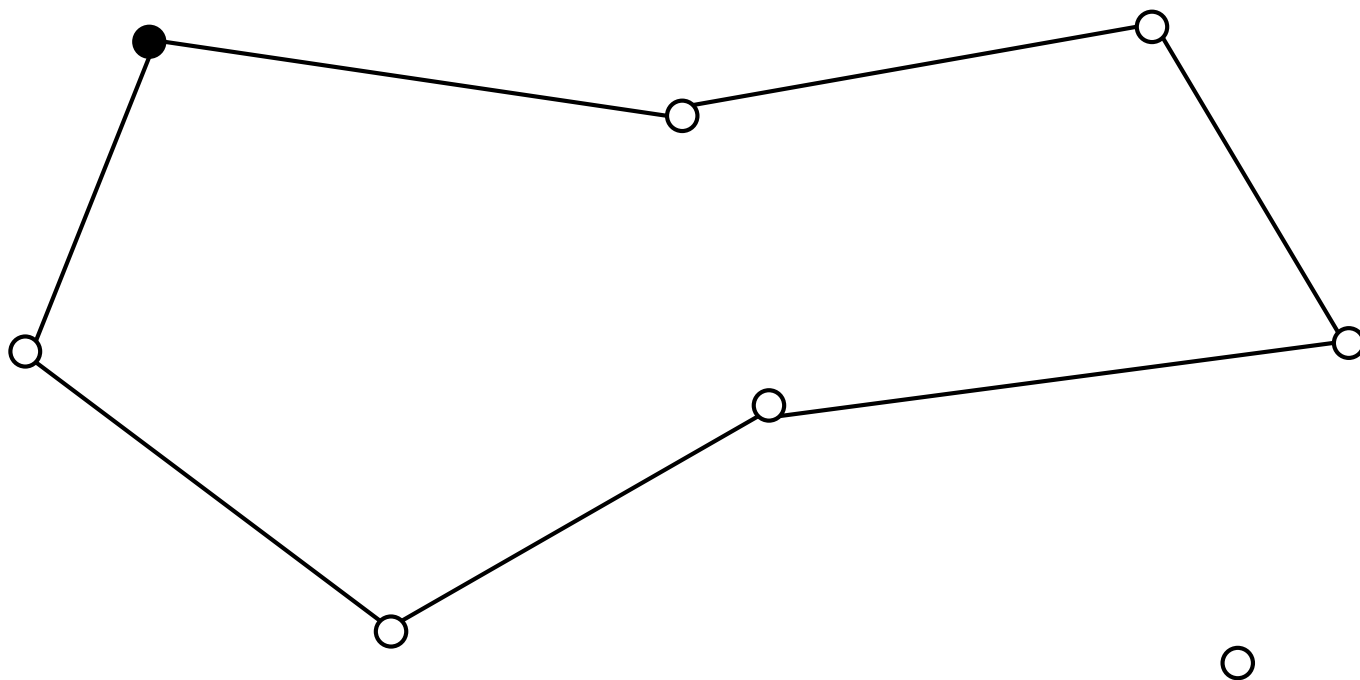
Metoda rozbudowy cyklu (greedy cycle)



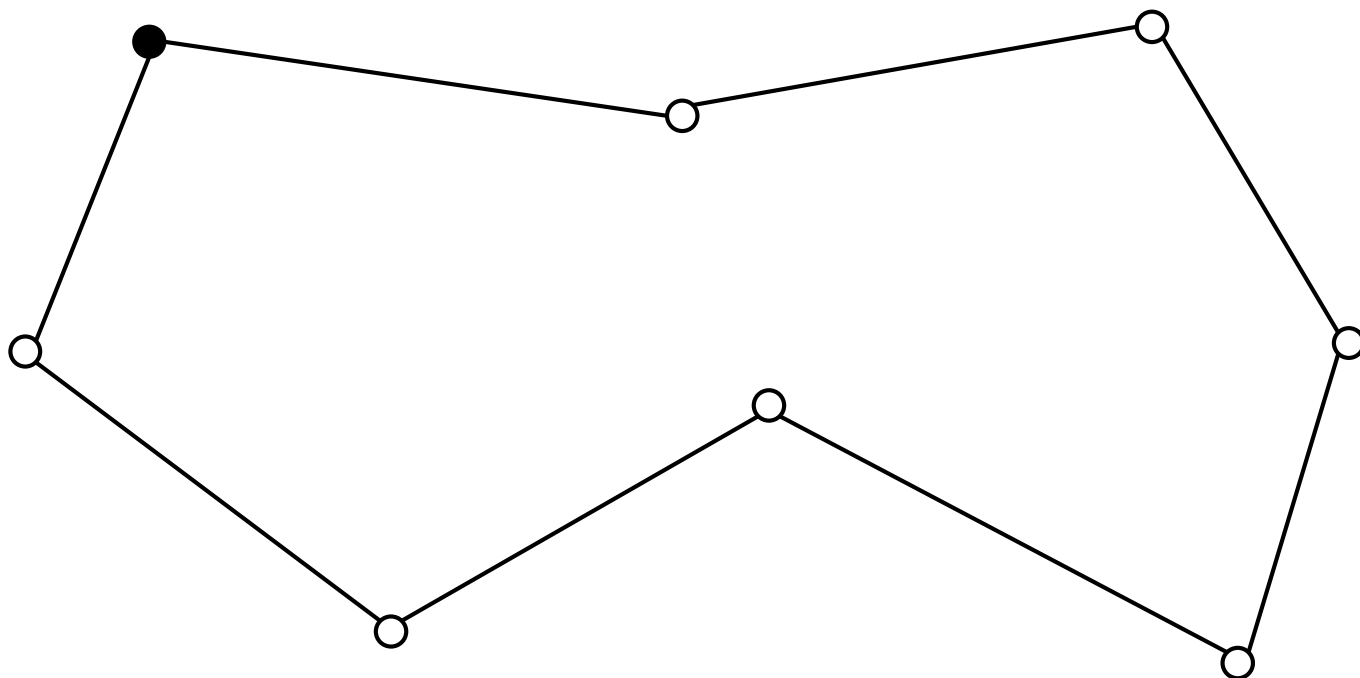
Metoda rozbudowy cyklu (greedy cycle)



Metoda rozbudowy cyklu (greedy cycle)



Metoda rozbudowy cyklu (greedy cycle)



Heurystyka Clarke-Wright (dla VRP)

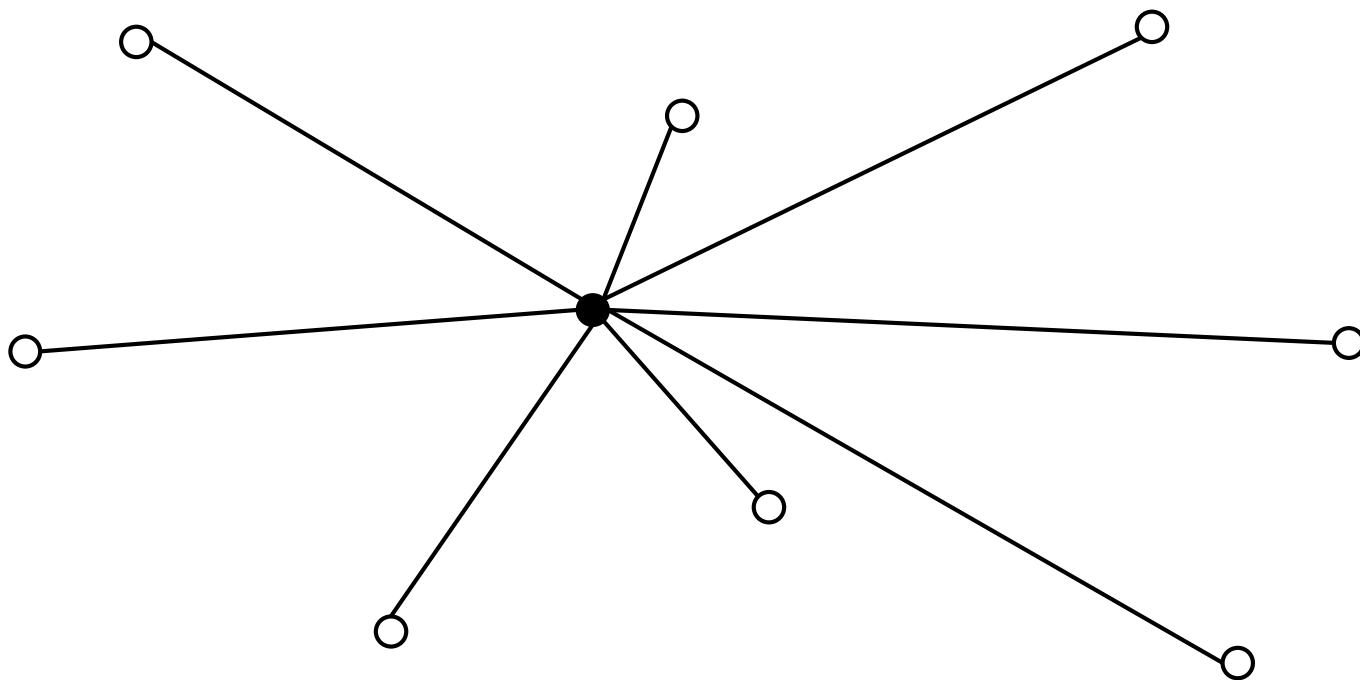
Połącz każdy wierzchołek z bazą tworząc oddzielną trasę do każdego wierzchołka

powtarzaj

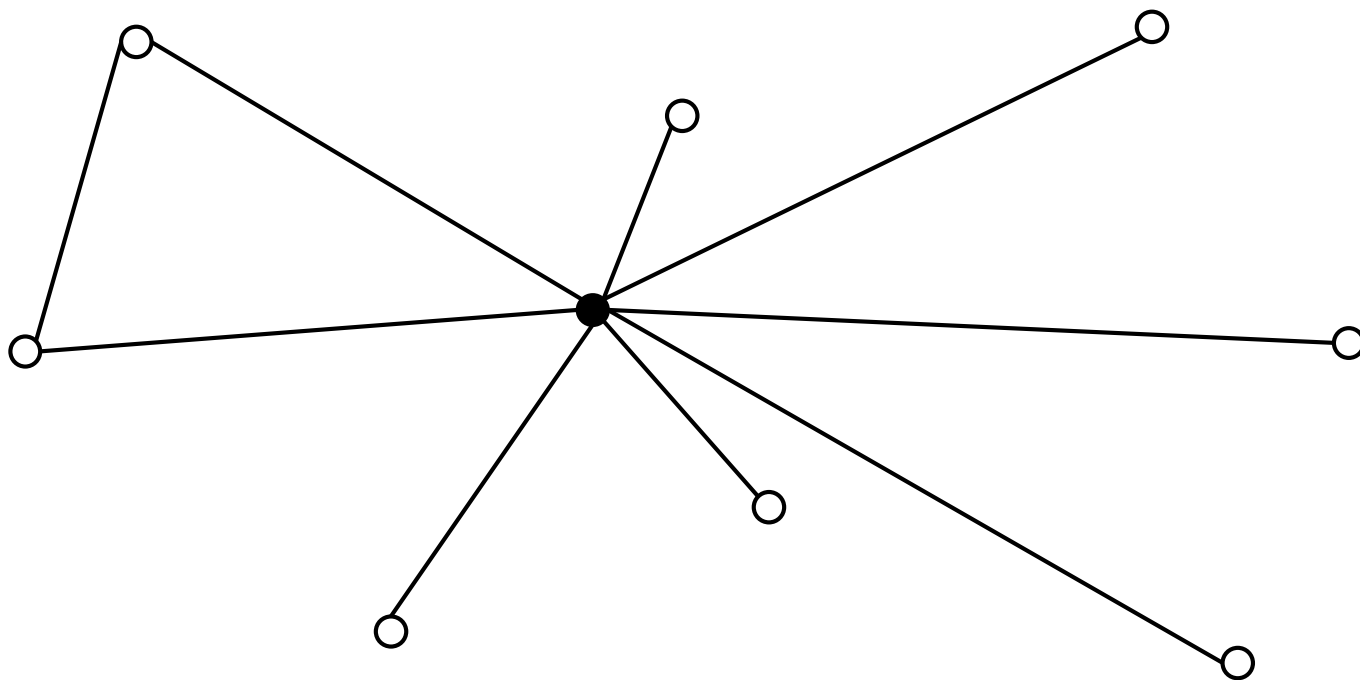
Wybierz i połącz w jedną dwie trasy, które dają najlepszą poprawę wartości funkcji celu, i których połączenie jest dopuszczalne

dopóki da się połączyć jakieś trasy nie łamiąc ograniczeń

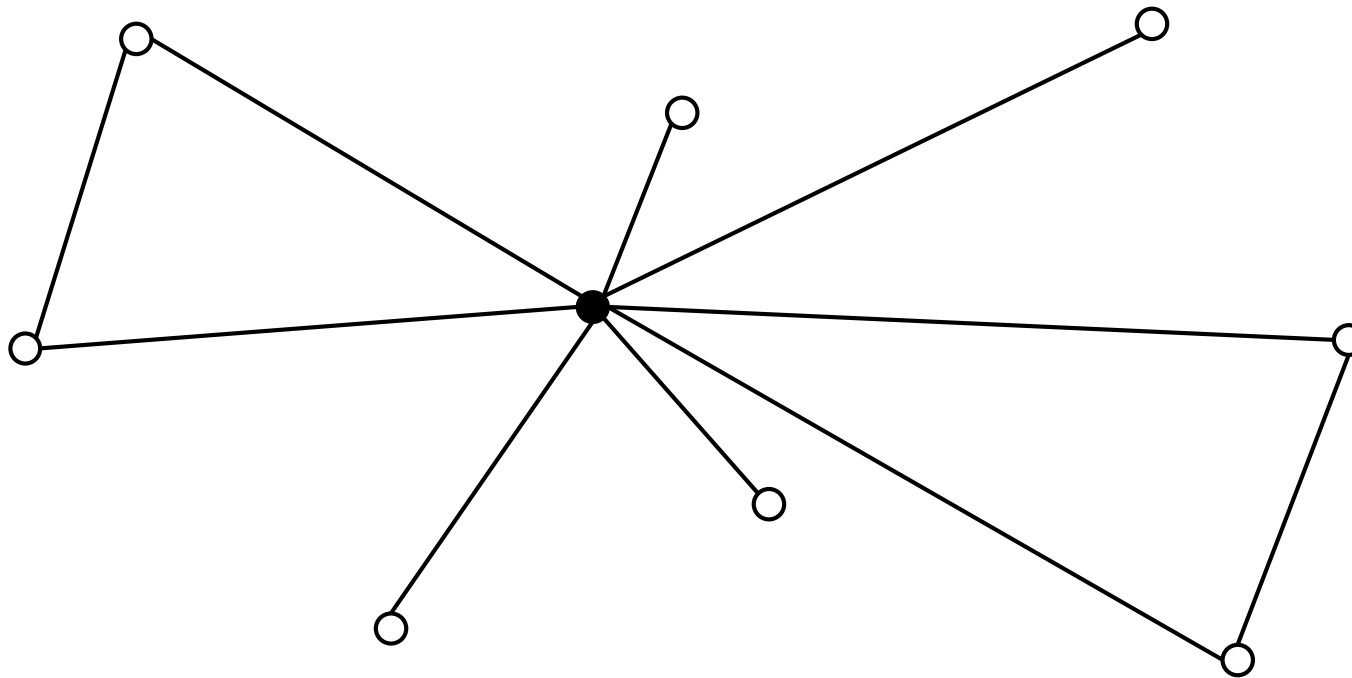
Heurystyka Clarke-Wright (dla VRP)



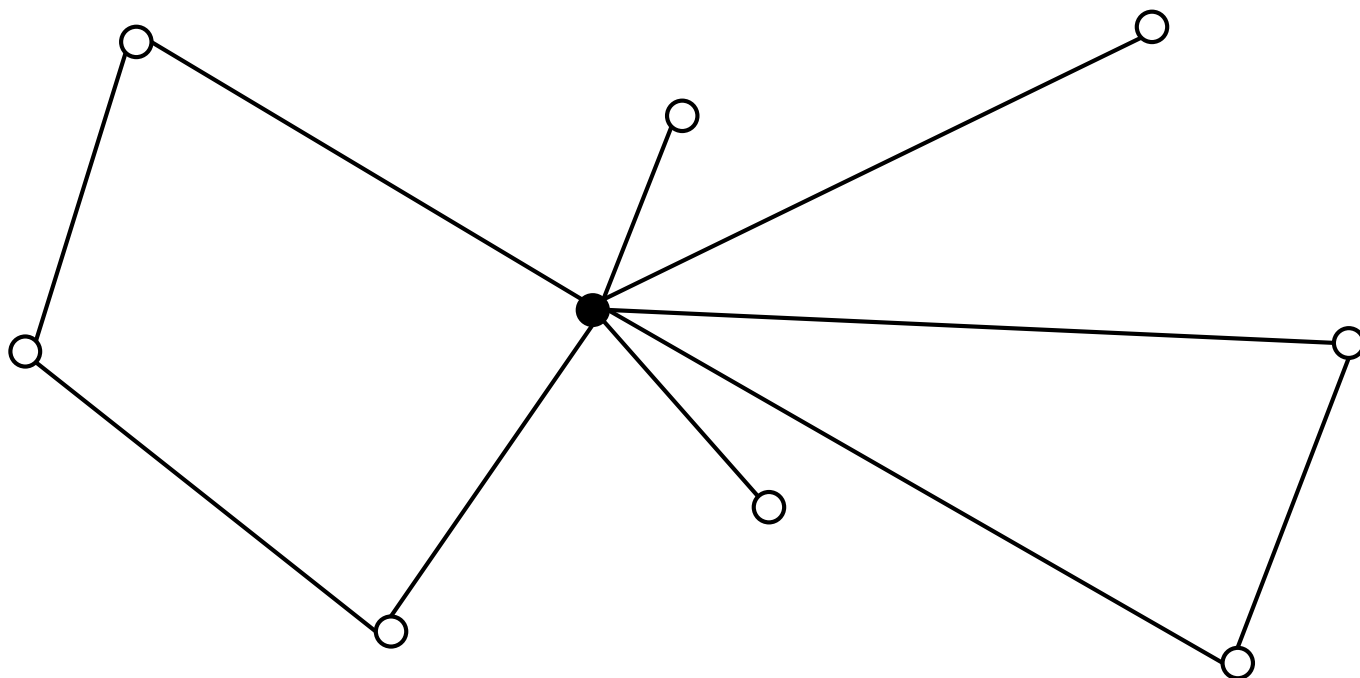
Heurystyka Clarke-Wright (dla VRP)



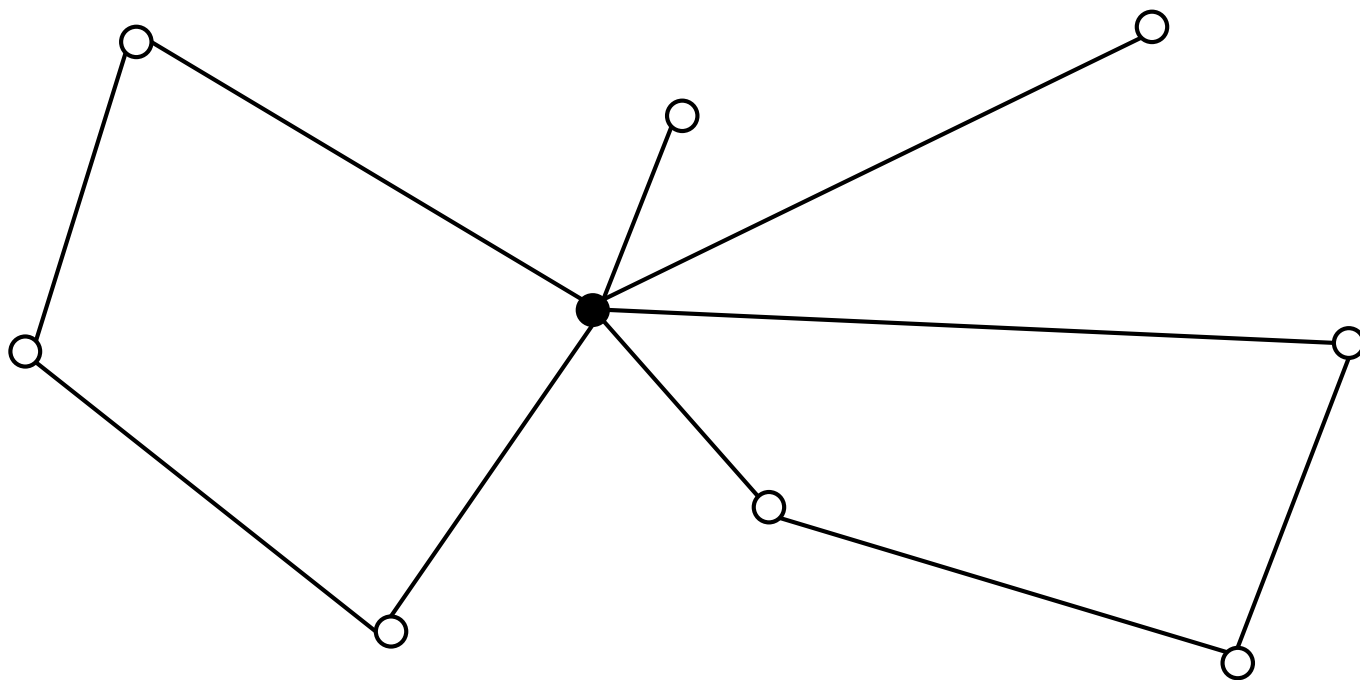
Heurystyka Clarke-Wright (dla VRP)



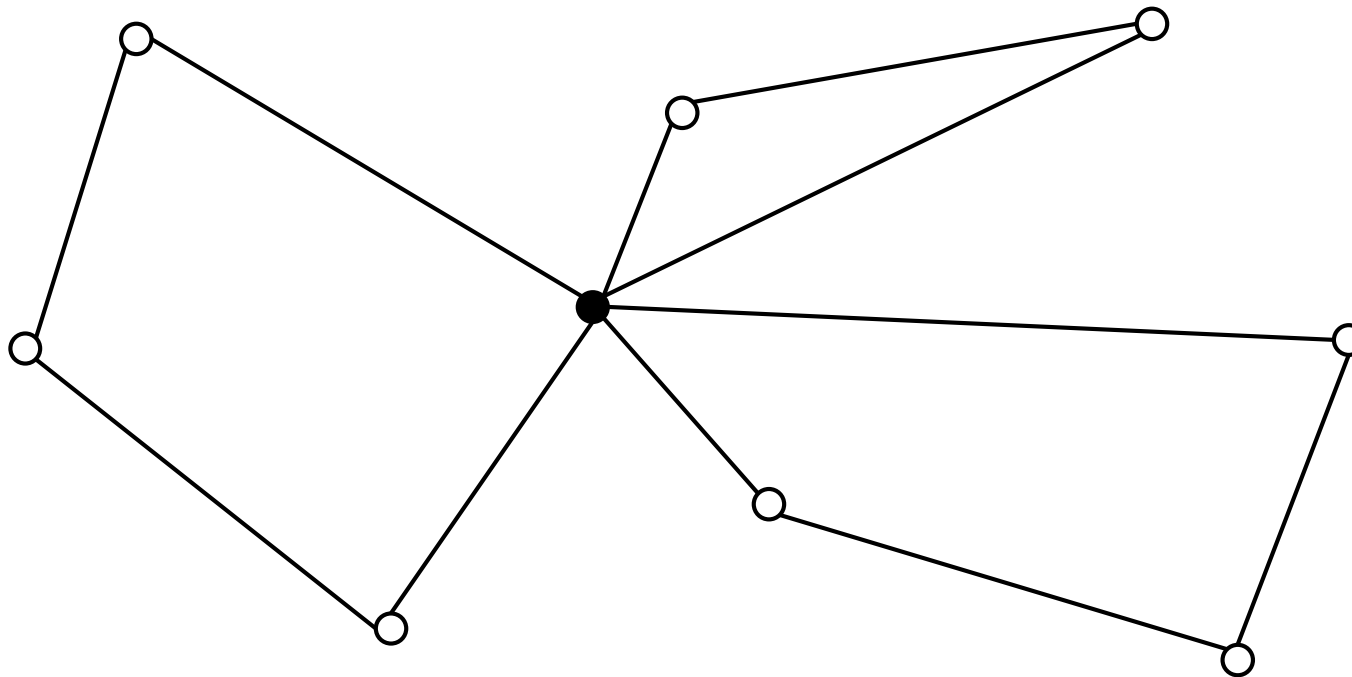
Heurystyka Clarke-Wright (dla VRP)



Heurystyka Clarke-Wright (dla VRP)



Heurystyka Clarke-Wright (dla VRP)



Optymalność algorytmów zachłannych

- Są optymalne (gwarantują znalezienie optimum globalnego) dla problemów, które są ważonymi matroidami z nieujemnymi wagami, gdzie waga sumy elementów jest sumą ich wag (np. minimalne drzewo rozpinające – spanning tree).
 - Matroid to para (E, L) , gdzie E to skończony zbiór bazowy a L to zbiór niezależnych podzbiorów E , tj. takich, że
 - Zbiór pusty jest niezależny
 - Każdy podzbiór zbioru niezależnego jest niezależny $A' \subset A \in L$ to $A' \in L$
 - Można wykazać, że wszystkie maksymalne zbiory niezależne mają tę samą liczbę elementów
 - Np. dla spanning tree E to zbiór wierzchołków, L to zbiór wszystkich drzew

$(1 - 1/e)$ -Aproksymacja algorytmów zachłannych

- Jeżeli funkcja celu jest submodularna i monotoniczna
- Submodularna – dodawanie do nadzbioru nie może być lepsze niż do podzbioru
 - $A \subseteq B \Rightarrow f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B)$
- Np. MaxCover problem – należy wybrać K podzbiorów maksymalizujących łączną liczbę (lub wagę) pokrytych elementów
- Hypervolume subset selection problem – wybór podzbioru rozwiązań Pareto-optimalnych maksymalizującego miarę jakości (reprezentacji) hypervolume

Praktyczne zachowanie heurystyk zachłannych dla innych problemów

- Powyższe założenia (bycie matroidem, submodularność i monotoniczność) mogą być spełnione „w przybliżeniu”
- Efektem może być generowanie rozwiązań znacznie lepszych od rozwiązań losowych

Randomizacja heurystyk zachłannych

- Wadą heurystyk zachłannych jest ich determinizm
- ... lub niewielka możliwość randomizacji
 - np. randomizacja poprzez wybór wierzchołka startowego w heurystyce najbliższego sąsiada TSP

Greedy Randomized Adaptive Search Procedure GRASP

Utwórz niepełne rozwiązanie początkowe, np. $\mathbf{x} := \emptyset$

powtarzaj

stwórz ograniczoną listę kandydatów (restricted candidate list – RCL)
złożoną z pewnej liczby najlepszych elementów nie wchodzących
jeszcze w skład rozwiązania \mathbf{x}

dodaj do \mathbf{x} losowo wybrany element z RCL

dopóki nie utworzono pełnego rozwiązania

Jak tworzyć RCL

- Reguła ilościowa/procentowa
 - Wybieramy $p\%$ najlepszych elementów
- Reguła wartościowa (bazująca na wartości f. celu)
 - $\Delta f(e)$ – zmiana funkcji celu po dodaniu elementu e
 - Δf_{\min} i Δf_{\max} – minimalna i maksymalna zmiana f. celu (zakładamy maksymalizację, czyli chcemy wybrać element o jak najmniejszej $\Delta f(e)$)
 - $\Delta f(e) \in [\Delta f_{\min}, \Delta f_{\min} + \alpha (\Delta f_{\max} - \Delta f_{\min})]$, $\alpha \in (0, 1)$
 - Co jeżeli $\alpha = 0$?
 - Co jeżeli $\alpha = 1$?

Inny sposób randomizacji

Utwórz rozwiązanie początkowe, np. $\mathbf{x} := \emptyset$

powtarzaj

przejrzyj $p\%$ losowo wybranych elementów (ale minimum 1) nie wchodzących jeszcze w skład rozwiązania i wybierz najlepszy z nich
dodaj do \mathbf{x} wybrany element

dopóki nie utworzono pełnego rozwiązania

- Co jeżeli $p = 0\%$?
- Co jeżeli $p = 100\%$?

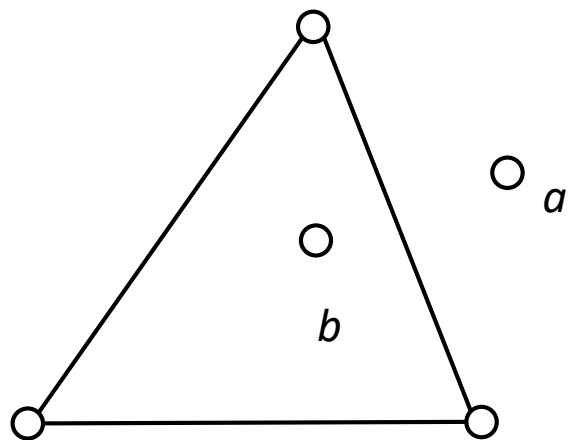
Heurystyki zachłanne oparte na żalu – regret heuristics

- Załóżmy, że element możemy wstawiać do rozwiązania w różne miejsca (np. wierzchołki w różne miejsca trasy TSP)
- Np. dwa elementy a i b :
- Koszty wstawienia a :
 - 1, 2, 3, 4, 5
- Koszty wstawienia b :
 - 5, 9, 10, 12, 15
- Zgodnie z regułą zachłanną wybralibyśmy a ($1 < 5$)
- Wtedy jednak możemy zablokować sobie możliwość wstawienia b przy koszcie 5 a zostanie nam jedynie miejsce z kosztem 9

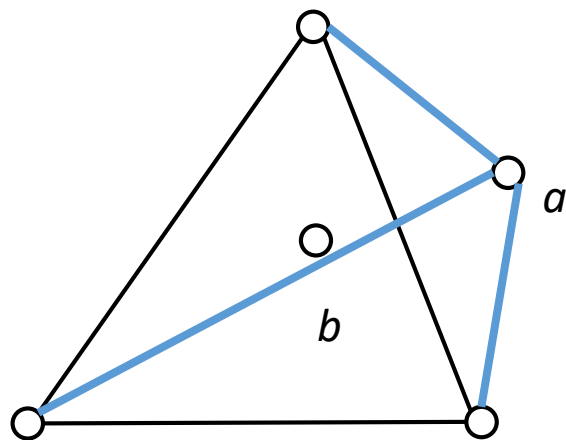
Heurystyki zachłanne oparte na żalu – regret heuristics

- k -żal (k -regret) to suma różnic pomiędzy najlepszym, a $k-1$ kolejnymi opcjami wstawienia
- 2-żal różnica pomiędzy pierwszą a drugą opcją
- Wybieramy element o największym żalu i wstawiamy go w najlepsze miejsce
- Możemy też ważyć żal z regułą zachłanną (oceną pierwszej opcji)

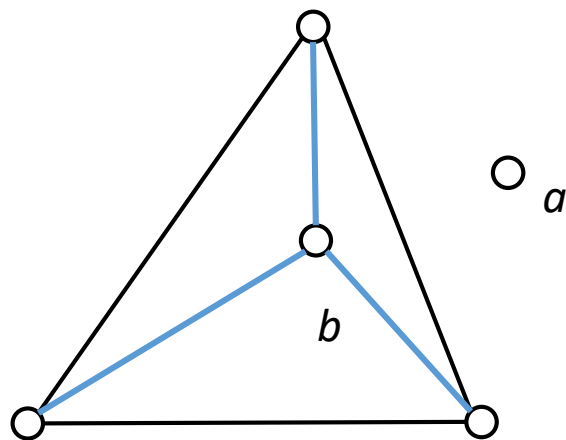
Metoda rozbudowy cyklu (greedy cycle)



Opcje wstawienia a



Opcje wstawienia b

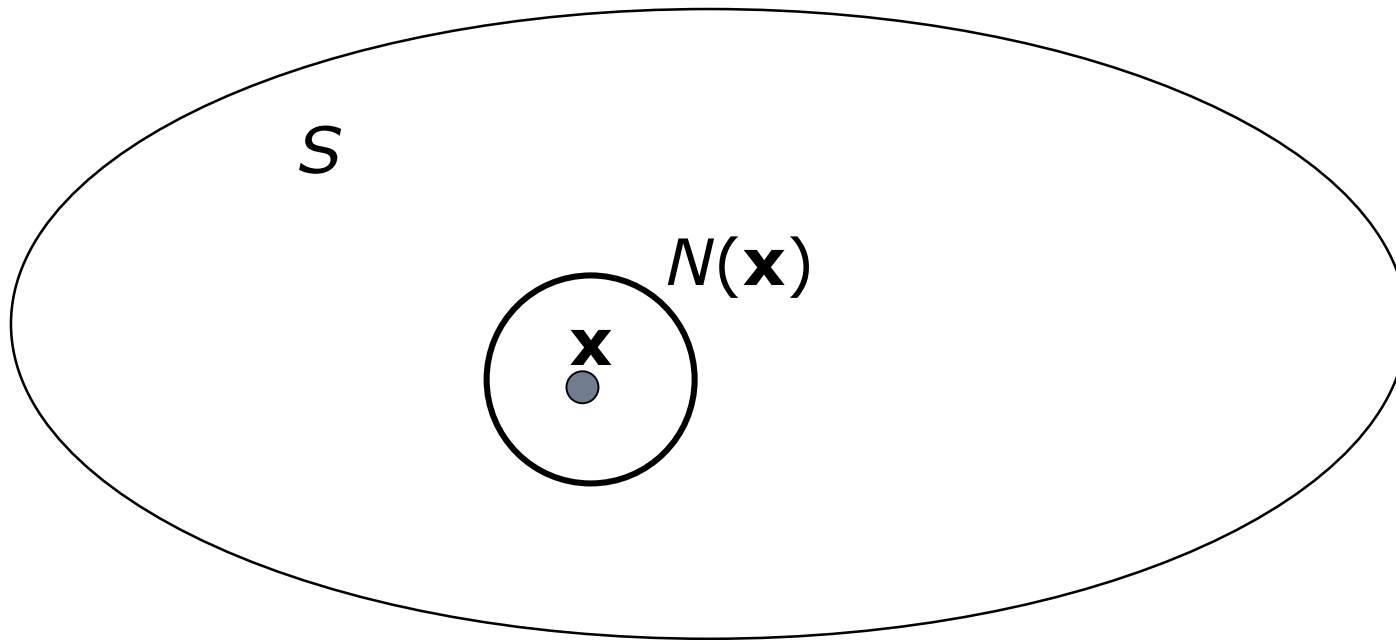


Beam search

- Zamiast wstawiania jednego elementu możemy oceniać wstawienie kilku do przodu (ale wstawiamy tylko ten pierwszy)
- Oznacza to częściowe rozwinięcie drzewa pełnego przeglądu
- W beam search rozważamy najpierw dodanie pewnej liczby najlepszych elementów, potem kilku kolejnych najlepszych (po dodaniu poprzedniego) itd.
- Np. beam search na bazie NN dla TSP
 - Rozważamy kilka najbliższych wierzchołków do ostatnio dodanego, dla każdego z nich kilka najbliższych do tego wierzchołka, itd.
 - Można połączyć z mechanizmem podziału i ograniczeń – odrzucać ścieżki, które nie mogą być lepsze od najlepszej już znalezionej

Lokalne przeszukiwanie

- Idea sąsiedztwa – topologia przestrzeni rozwiązań (dopuszczalnych)



Definicja sąsiedztwa oparta na odległości

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

$$d: S \times S \rightarrow \mathbf{R}$$

- sąsiedztwo

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

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

Definicja sąsiedztwa oparta na ruchach

- $N(\mathbf{x})$ można uzyskać z \mathbf{x} wykonując jeden ruch (modyfikację, transformację) 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 ta jest równoważna poprzedniej jeżeli odległości pomiędzy \mathbf{x} i \mathbf{y} jest zdefiniowana jako liczba ruchów jakie trzeba wykonać, żeby przejść z \mathbf{x} do \mathbf{y} i $\varepsilon = 1$

Pożądane cechy sąsiedztwa

- Z reguły zakładamy, że $\mathbf{y} \in N(\mathbf{x}) \Leftrightarrow \mathbf{x} \in N(\mathbf{y})$
 - W przypadku transformacji m oznacza to, że jest odwracalna
- Ograniczenia na rozmiar
 - $N(\mathbf{x})$ musi zawierać co najmniej jedno (w praktyce więcej) rozwiązań różnych niż \mathbf{x}
 - Rozmiar $N(\mathbf{x})$ powinien być dużo mniejszy niż rozmiar przestrzeni rozwiązań dopuszczalnych
 - Z reguły rozmiar $N(\mathbf{x})$ jest wielomianowy względem wielkości instancji (choć są wyjątki)

Pożądane cechy sąsiedztwa

- Podobieństwo sąsiadów
 - Wykonanie transformacji m powinno być proste (prostsze niż skonstruowanie rozwiązania od zera), uwzględniając:
 - Modyfikację struktur danych
 - Przeliczenie f. celu
 - Sprawdzenie ograniczeń
- Równouprawnienie
 - Z każdego rozwiązania powinno dać się przejść do każdego innego
 - Może nie być łatwe w przypadku problemów z ograniczeniami

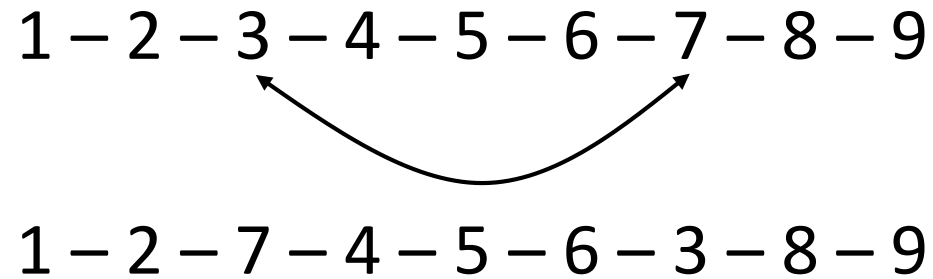
Typowe operatory sąsiedztwa

- Zmiana wartości jednego elementu, np. zmiana wartości binarnej na przeciwną (np. zmiana wyboru elementu w problemie plecakowych, zmiana przydziału elementu do zadania w problemie przydziału)
- Przesunięcie jednego elementu ze zbioru do zbioru (np. pomiędzy trasami VRP)
- Wymiana dwóch elementów pomiędzy dwoma zbiorami (np. pomiędzy trasami VRP)
- Przesunięcie elementu w sekwencji (np. TSP, szeregowanie)
- Zamiana pozycji elementów w sekwencji (np. TSP, szeregowanie)
- Odwrócenie kolejności części sekwencji (np. TSP, szeregowanie)
- Operacje mogą też dotyczyć większej (choć niewielkiej) liczby elementów

Definiowanie sąsiedztwa

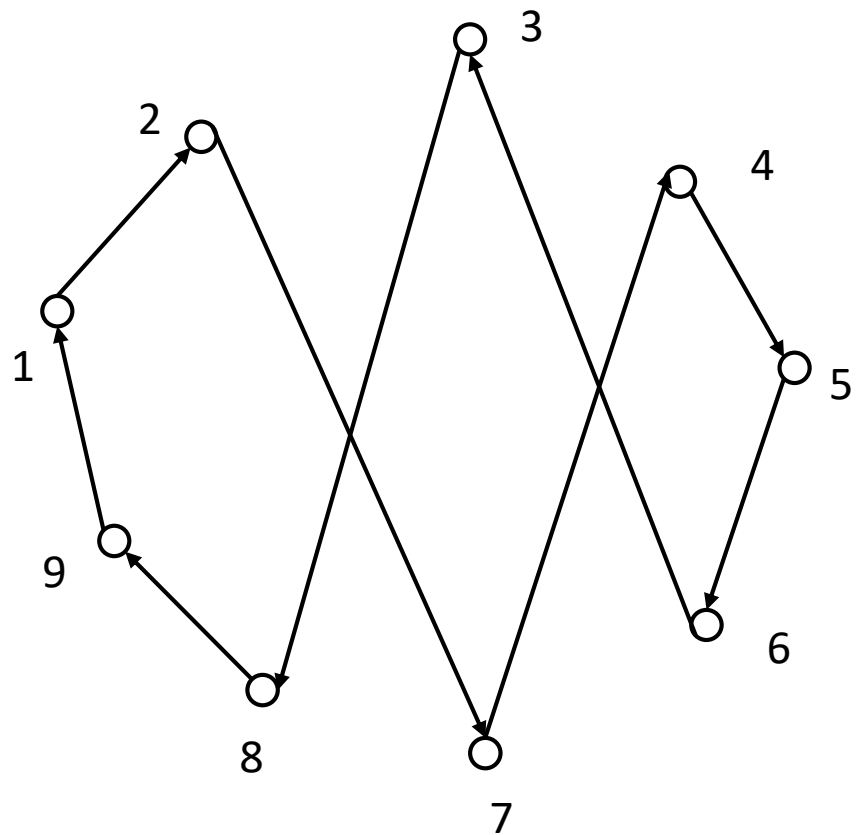
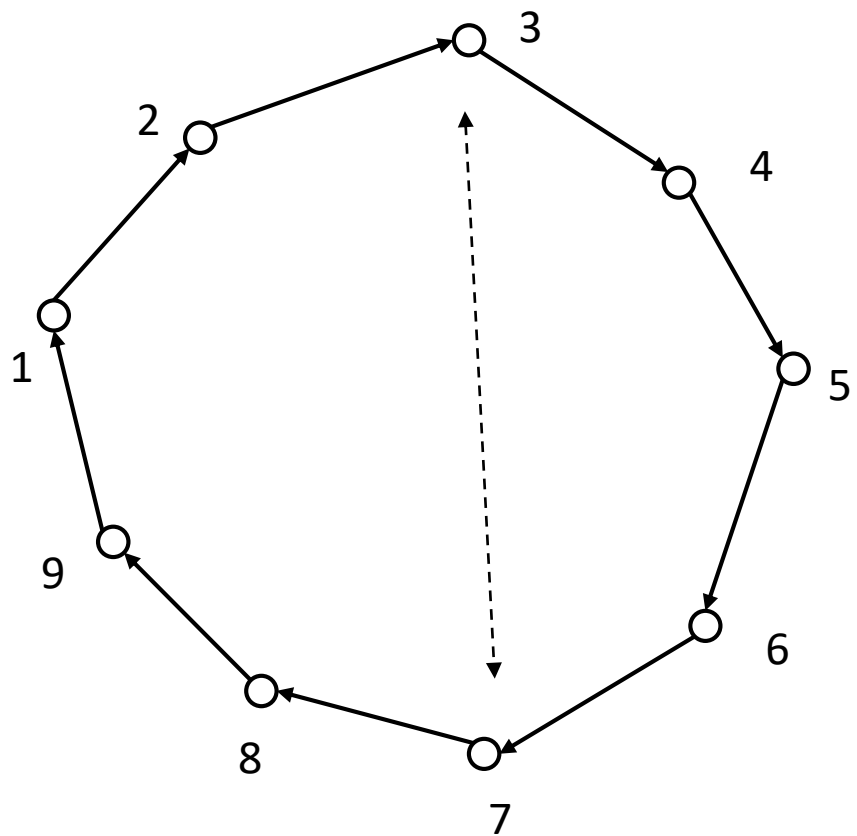
- Sąsiedztwo nie jest obiektywną cechą problemu, często mamy wiele opcji wyboru
- Z drugiej strony, tylko niektóre sąsiedztwa spełniają powyższe cechy

Przykład: Wymiana dwóch wierzchołków w TSP



- Rozmiar sąsiedztwa:
 - $|N(x)| = n(n-1)/2, O(n^2)$
 - Gdzie n to liczba wierzchołków

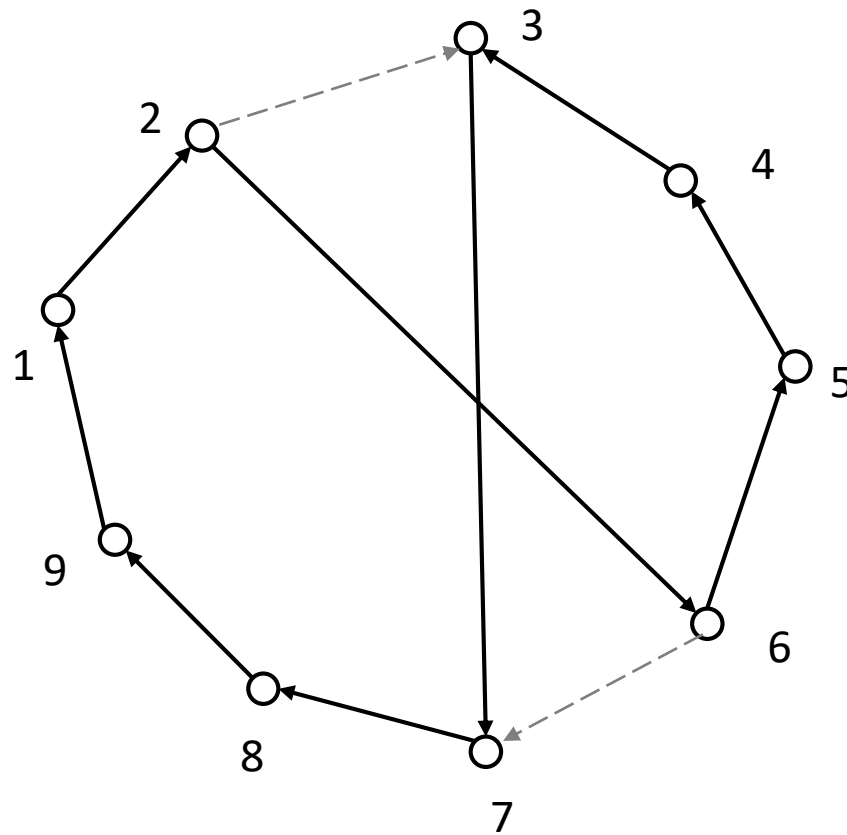
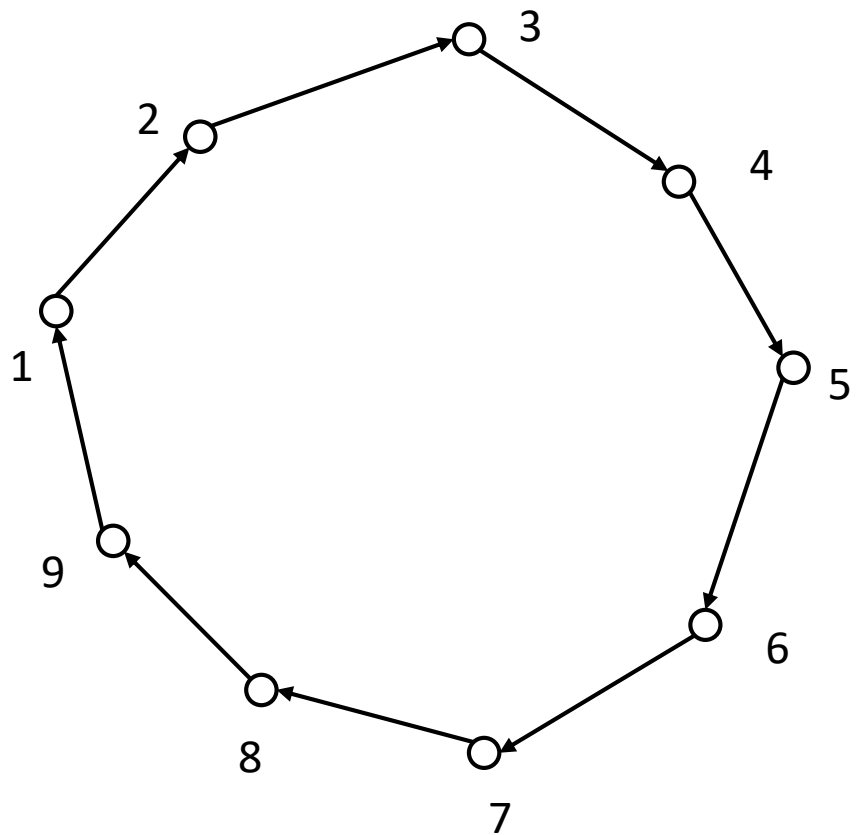
Przykład: Wymiana dwóch wierzchołków w TSP



Wymiana K wierzchołków w TSP

- Inny punkt widzenia: wyjmujemy z rozwiązania K wierzchołków i wstawiamy je w zwolnione miejsca w inny sposób
 - W ogólności mamy $K! - 1$ innych niż aktualny sposobów wstawienia – liczba permutacji innych niż obecna
 - Rozmiar sąsiedztwa $O\left(\binom{n}{K} K!\right) = O\frac{n!}{(n-K)!} = O(n^K)$

Wymiana krawędzi (łuków) w TSP




Wymiana krawędzi (łuków) w TSP

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

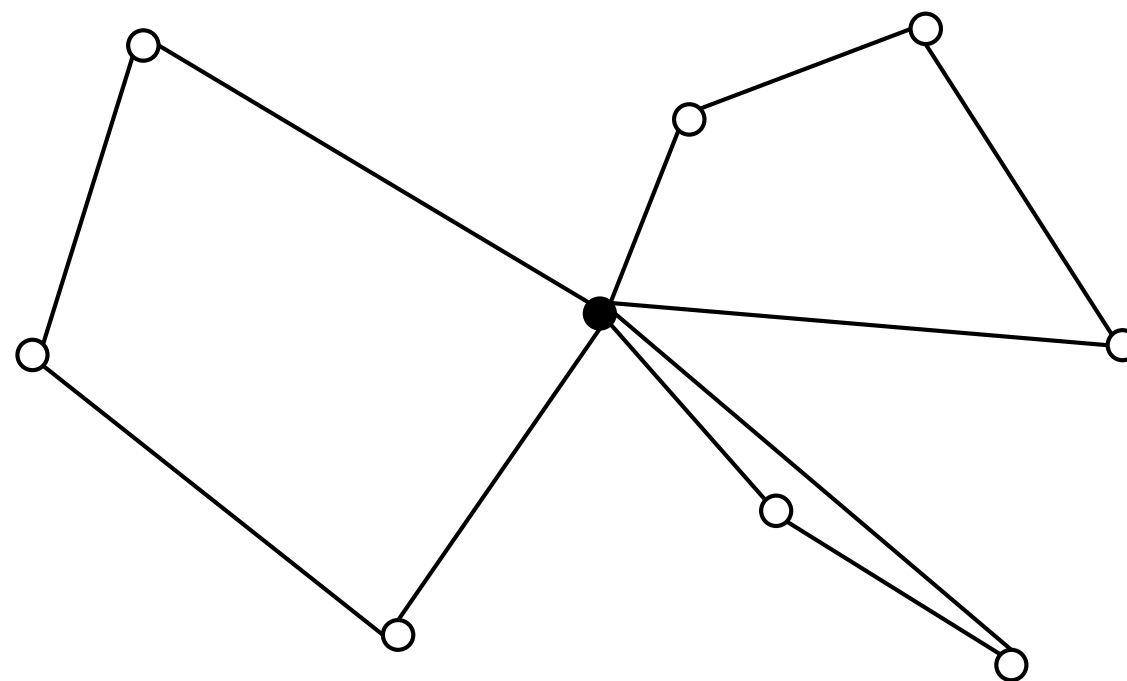
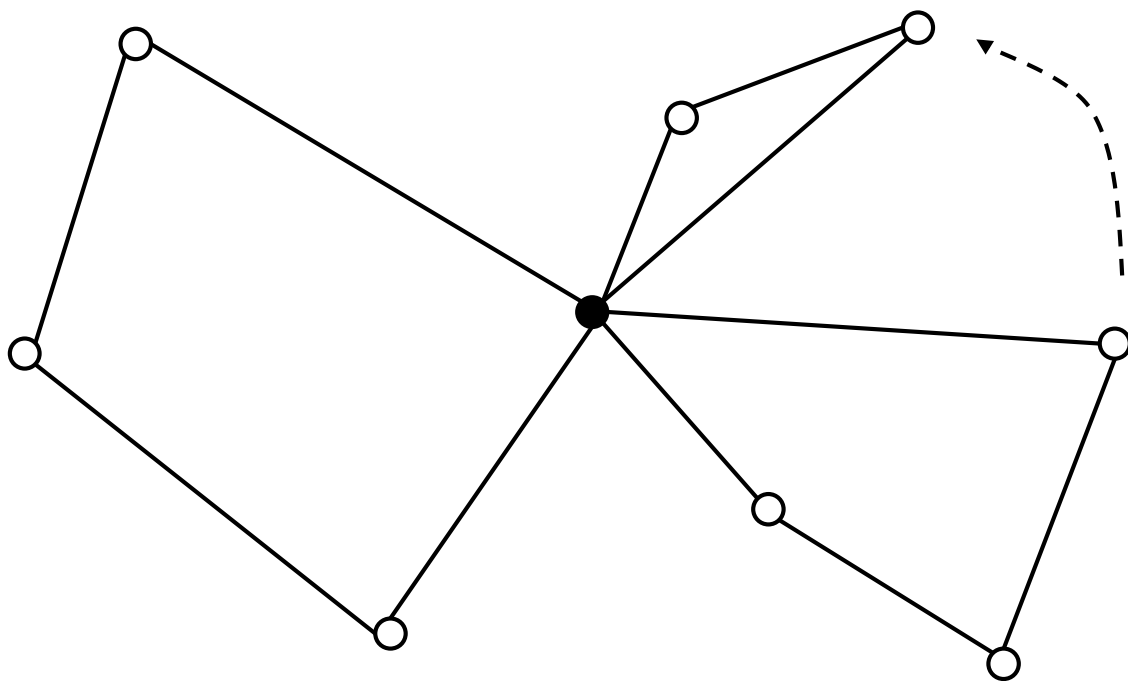
→

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

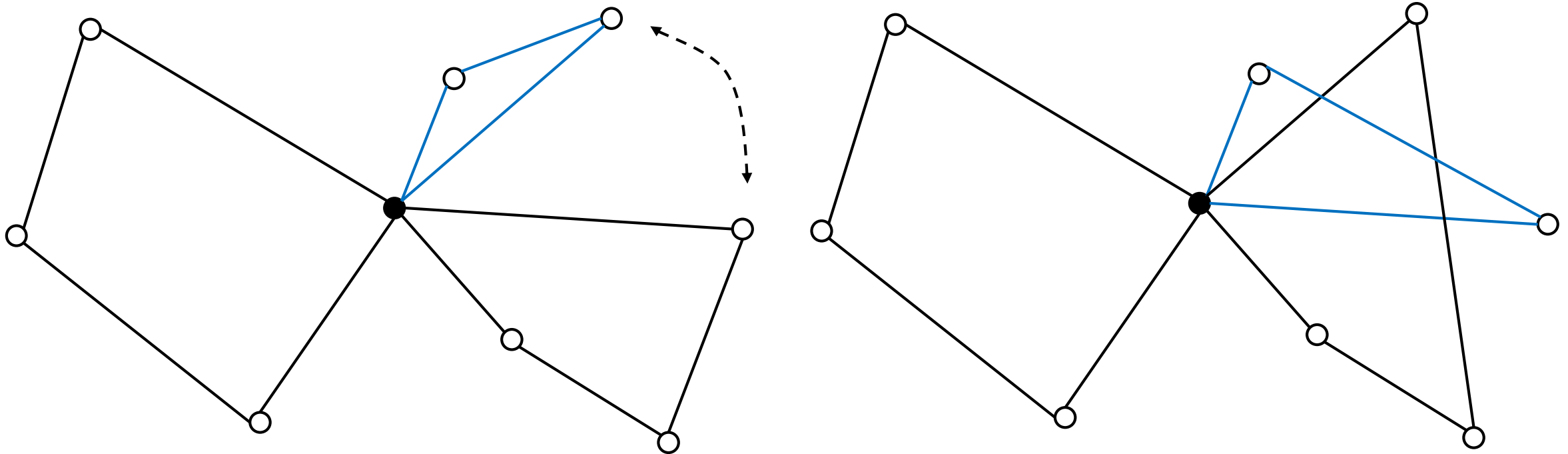


- Rozmiar sąsiedztwa:
 - $|N(x)| = n(n-3)/2, O(n^2)$
- Można też wymieniać K krawędzi

Przesunięcie wierzchołka w problemie VRP



Wymiana dwóch wierzchołków w problemie VRP



Rozmiary sąsiedztw VRP

- Przesunięcie wierzchołka
 - $|N(\mathbf{x})| = n(T - 1) = O(nT)$, gdzie T liczba tras
- Wymiana wierzchołków
 - $|N(\mathbf{x})| \approx n((n - n/T)) / 2 = O(n^2)$

Ogólna idea lokalnego przeszukiwania (local search)

- Tworzymy rozwiązanie początkowe
 - Losowe lub za pomocą (zrandomizowanej) heurystyki
- Poprawiamy rozwiązanie przechodząc do rozwiązań sąsiednich (wykonując ruchy) akceptując jedynie rozwiązania (ruchy) przynoszące poprawę, tak długo jak jest to możliwe (tj. istnieją rozwiązania sąsiednie/ruchy przynoszące poprawę)
- Używana też nazwa Hill-climber

Pojęcie lokalnego 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 w wersjach stromej (steepest) i zachłannej (greedy) – przypadek maksymalizacji

Wersja zachłanna (greedy)

Wygeneruj rozwiązanie startowe \mathbf{x}

powtarzaj

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

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

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

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

Wersja stroma (steepest)

Wygeneruj rozwiązanie startowe \mathbf{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 po przejrzaniu całego $N(\mathbf{x})$

Lokalne przeszukiwanie w wersji zachłannej

- Akceptowane jest pierwsze poprawiające rozwiązanie
- Jeżeli sąsiedztwo (ruchy) będą przeglądane w pewnej ustalonej kolejności może to prowadzić do ukierunkowania (bias) algorytmu (np. preferowanie wymiany pewnych wierzchołków)
- Rozwiązanie idealne to przeglądanie sąsiedztwa (ruchów) w losowej kolejności
- Rozwiązanie akceptowalne to rozsądna randomizacja eliminująca większość ukierunkowania, np. rozpoczynanie pętli po ruchach w losowym miejscu i losowanie kierunku przeglądania pętli

Wersja zachłanna vs. stroma

- Wersja zachłanna z reguły działa krócej, ale daje gorsze rozwiązania niż stroma
- Porównanie nie jest jednoznaczne, ponieważ w czasie działania wersji stromej często można uruchomić wersję zachłanną kilkukrotnie i wybrać najlepsze rozwiązanie

Efektywność lokalnego przeszukiwania

- W większości bardziej zaawansowanych algorytmów wykorzystujących lokalne przeszukiwanie, to LP będzie konsumowało gro czasu obliczeń
- W samym LP **operacją krytyczną** (wąskim gardłem) **jest ocena sąsiednich rozwiązań**. Liczba rozwiązań ocenianych jest znacznie większa niż liczba rozwiązań akceptowanych
 - W wersji stromej jedno akceptowane po przejrzeniu całego sąsiedztwa
 - W wersji zachłannej, kiedy bieżące rozwiązanie jest stosunkowo dobre, także większość ruchów przynosi pogorszenie
- Należy więc maksymalnie usprawnić ocenianie rozwiązań
 - pozostałe fragmenty algorytmu często nie muszą być optymalizowane pod kątem efektywności

Bezpośrednia (trywialna) implementacja lokalnego przeszukiwania

- Rozwiązania sąsiednie są jawnie konstruowane przed oceną. Funkcja celu jest obliczana od podstaw
 - np. sumowanie długości krawędzi w TSP

Obliczanie i wykorzystanie zmiany (delty) funkcji celu

- Żeby ocenić rozwiązanie wystarczy obliczyć deltę f. celu $\Delta f_m(\mathbf{x})$
- Dla funkcji maksymalizowanej rozwiązanie $m(\mathbf{x}) \in N(\mathbf{x})$ jest lepsze od bieżącego (\mathbf{x}) , jeżeli:
 - $\Delta f_m(\mathbf{x}) > 0$
- ruch $m1$ jest lepszy od ruchu $m2$, jeżeli:
 - $\Delta f_{m1}(\mathbf{x}) > \Delta f_{m2}(\mathbf{x})$

Lokalne przeszukiwanie z wykorzystaniem delty

Wersja stroma

Wygeneruj rozwiązanie startowe \mathbf{x}

powtarzaj

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

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

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

dopóki nie znaleziono lepszego
rozwiązania po przejrzaniu całego
 $M(\mathbf{x})$

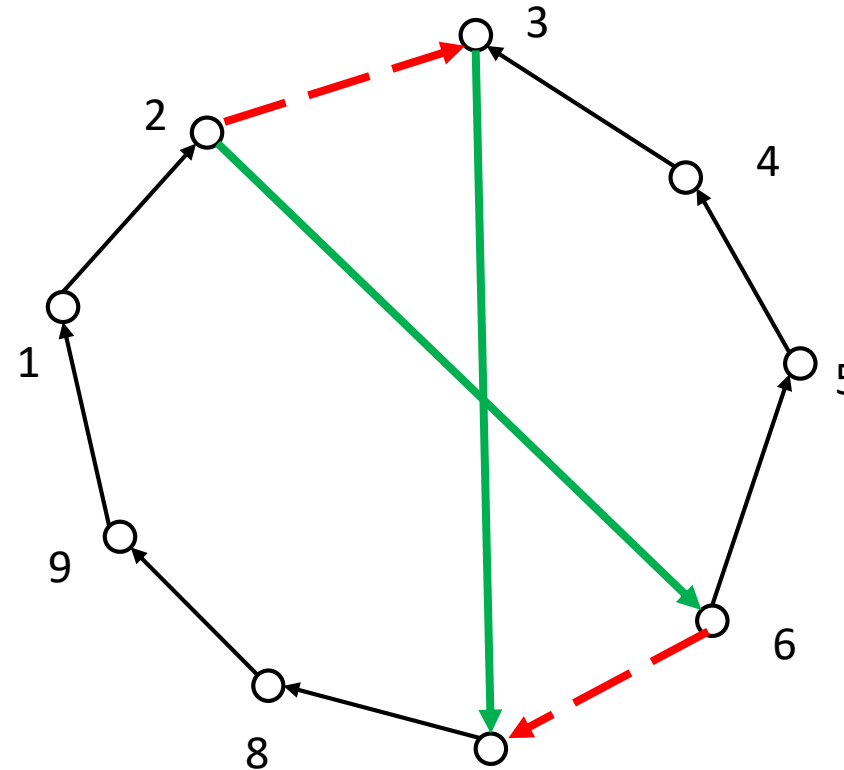
Ocenia się ruchy (zmianę f.
celu), nie rozwiązania!

Rozwiązania sąsiednie nie
muszą być jawnie
konstruowane!

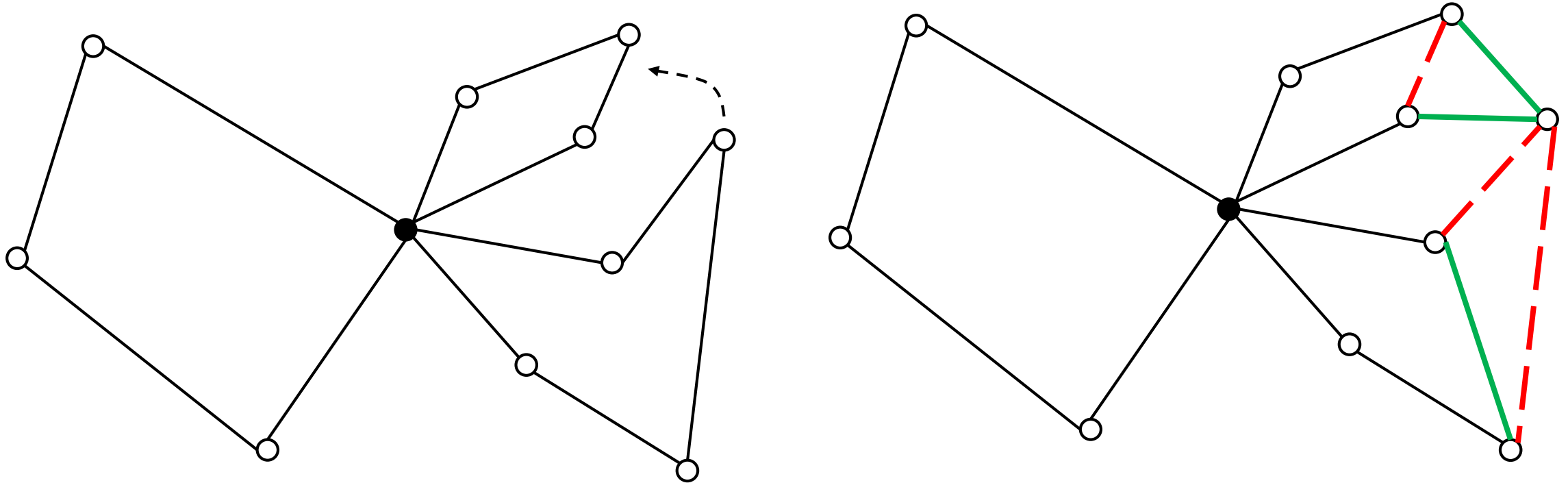
Dopiero tu modyfikowane
jest rozwiązanie

Delta w problemie komiwojażera (TSP) - wymiana 2 krawędzi

- Żeby obliczyć deltę wystarczy odjąć długości usuwanych krawędzi i dodać długości dodawanych krawędzi – 4 operacje arytmetyczne
- Obliczenie f. celu od zera wymaga n operacji



Delta w problemie VRP – przesunięcie wierzchołka



Dodanie długości trzech krawędzi i odjęcie długości trzech krawędzi – 6 operacji

Delta w problemie VRP – przesunięcie wierzchołka

- Liczba ruchów:
 $O(n^2)$, $\approx n^2/2$
- Obliczanie f. celu od zera (bez delty)
 - Trzeba zsumować długości $O(n)$ krawędzi
 - Złożoność jednej iteracji $O(n^2)$ $O(n) = O(n^3)$, $\approx n^3/2$
- Obliczanie f. celu przy użyciu delty
 - 6 operacji
 - Złożoność jednej iteracji $\approx 6 n^2/2 = 3 n^2$
- Przyspieszenie
 - $\approx n/6$

Wykorzystanie ocen ruchów z poprzednich iteracji lokalnego przeszukiwania

- 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 poprzedniego ruchu
 - Czyli choć rozwiązania sąsiednie są inne, to wiele ruchów jakie można wykonać i ich ocen pozostaje takich samych – nie trzeba ich ponownie oceniać

Lokalne przeszukiwanie w wersji stromej

Wygeneruj rozwiązanie startowe \mathbf{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 przejrzaniu całego $M(\mathbf{x})$

Wykorzystanie ocen ruchów z poprzednich iteracji LP dla VRP – przesunięcie wierzchołka

- Liczba przesunięć wierzchołków
 - $\approx n \times (T - 1) \times n / T \approx n^2$, gdzie T to liczba tras



- Kosz przeliczenia od zera nowego rozwiązania
 - $\approx n$
- Razem koszt jednej iteracji bez delty
 - $\approx n^3$
- Razem koszt jednej iteracji z deltą
 - $\approx 6n^2$

Wykorzystanie ocen ruchów z poprzednich iteracji LP dla VRP

- Załóżmy najpierw, że przeliczamy wszystkie ruchy dotyczące zmodyfikowanych tras (a nie tylko zmodyfikowanych miejsc tras)
 - To może mieć uzasadnienie w pewnych przypadkach np. gdy okna czasowe wpływają na całą trasę

Wykorzystanie ocen ruchów z poprzednich iteracji LP dla VRP – przesunięcie wierzchołka

- W poprzedniej iteracji zmieniły się tylko dwie trasy
- Należy przeliczyć (z wykorzystaniem delty) tylko ruchy dotyczące zmienionych tras
- Liczba przesunięć wierzchołków z tych tras

- $\approx 2 n / T \times (T - 1) \times n / T \approx 2 n^2 / T$, gdzie T to liczba tras

Liczba wierzchołków
w zmienionych
trasach

Średnia liczba pozycji wstawienia w trasie
Liczba pozostałych tras

- Liczba przesunięć obiektów do tych tras

- $\approx (n - 2 n / T) 2 \times n / T \approx 2 n^2 / T$

Liczba wierzchołków
w pozostałych
trasach

- Łącznie liczba ruchów dotyczących zmodyfikowanych tras
 - $\approx 4 n^2 / T$

Wykorzystanie ocen ruchów z poprzednich iteracji LP dla VRP – przesunięcie wierzchołka

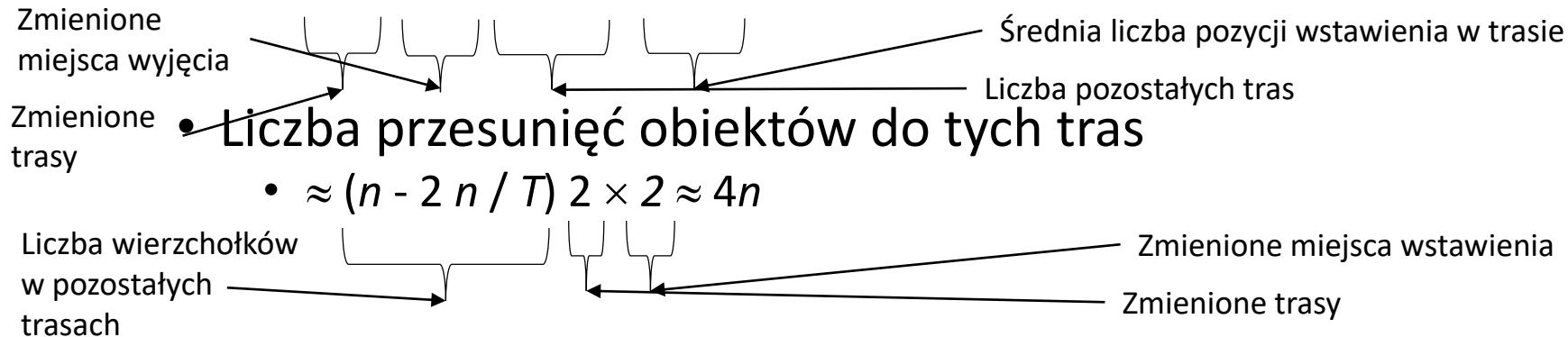
- Razem liczba nowych ruchów, które trzeba ocenić
 - $\approx 4n^2 / T$
- Złożoność jednej iteracji z deltą
 - $\approx (4n^2 / T) \times 6 = 24n^2 / T$
- Przyspieszenie (względem wersji bazowej z deltą)
 - $\approx (6n^2) / (24n^2 / T) = T / 4$
- Przyspieszenie względem wersji bazowej bez delty
 - $\approx n^3 / (24n^2 / T) = n \times T / 24$
- Dla $n=1000$, $T = 100$ (w pełni realne w praktyce) przyspieszenie względem wersji bazowej bez delty to 4166, a względem wersji bazowej z deltą to 25 - bez zmiany działania algorytmu

Wykorzystanie ocen ruchów z poprzednich iteracji LP dla VRP

- Wystarczy jednak (w niektórych przypadkach) przeliczać ruchy, których otoczenie się zmieniło, tj. zmienił się sąsiedni wierzchołek – tylko 2 na trasę

Wykorzystanie ocen ruchów z poprzednich iteracji LP dla VRP

- Wystarczy jednak przeliczać ruchy, których otoczenie się zmieniło – tylko 2 na trasę
- Liczba przesunięć wierzchołków z tych tras
 - $\approx 2 \times 2 \times (T - 1) \times n / T \approx 4n$, gdzie T to liczba tras



- **Liczba przesunięć obiektów do tych tras**
 - $\approx (n - 2n / T) 2 \times 2 \approx 4n$
- Łącznie liczba ruchów dotyczących zmodyfikowanych tras
 - $\approx 8n$

Wykorzystanie ocen ruchów z poprzednich iteracji LP dla VRP – przesunięcie wierzchołka

- Razem liczba nowych ruchów, które trzeba ocenić
 - $\approx 8n$
- Złożoność jednej iteracji z deltą
 - $\approx 8n \times 6 = 48n$
- Przyspieszenie (względem wersji bazowej z deltą)
 - $\approx (6n^2) / (48n) = n / 8$
- Przyspieszenie względem wersji bazowej bez delty
 - $\approx n^3 / (48n) = n^2 / 48$
- Dla $n=1000$, $T = 100$ (w pełni realne w praktyce) przyspieszenie względem wersji bazowej bez delty to 20833, a względem wersji bazowej z deltą to 125 - bez zmiany działania algorytmu

Wykorzystanie ocen ruchów z poprzednich iteracji LP

- Rozwiązania techniczne
 - Cache ruchów z poprzedniej iteracji
 - lub
 - Uporządkowana wg. delty lista ruchów przynoszących poprawę
 - Związany z tym narzut może redukować efekty – efektywne przyspieszenie będzie mniejsze niż teoretyczne (lub nawet żadne)

Uporządkowana wg. delty lista ruchów przynoszących poprawę

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

Wygeneruj rozwiązanie startowe \mathbf{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 aplikowalny ruch m **to**

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

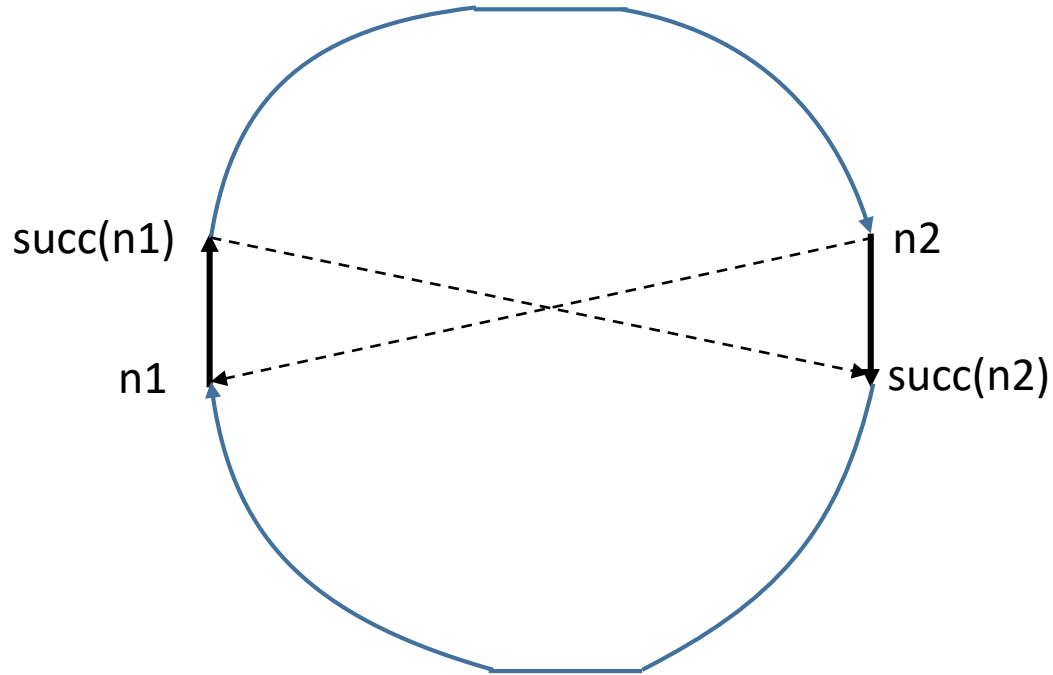
dopóki nie znaleziono ruchu aplikowalnego m po przejrzaniu całej listy LM

Ruchy aplikowalne

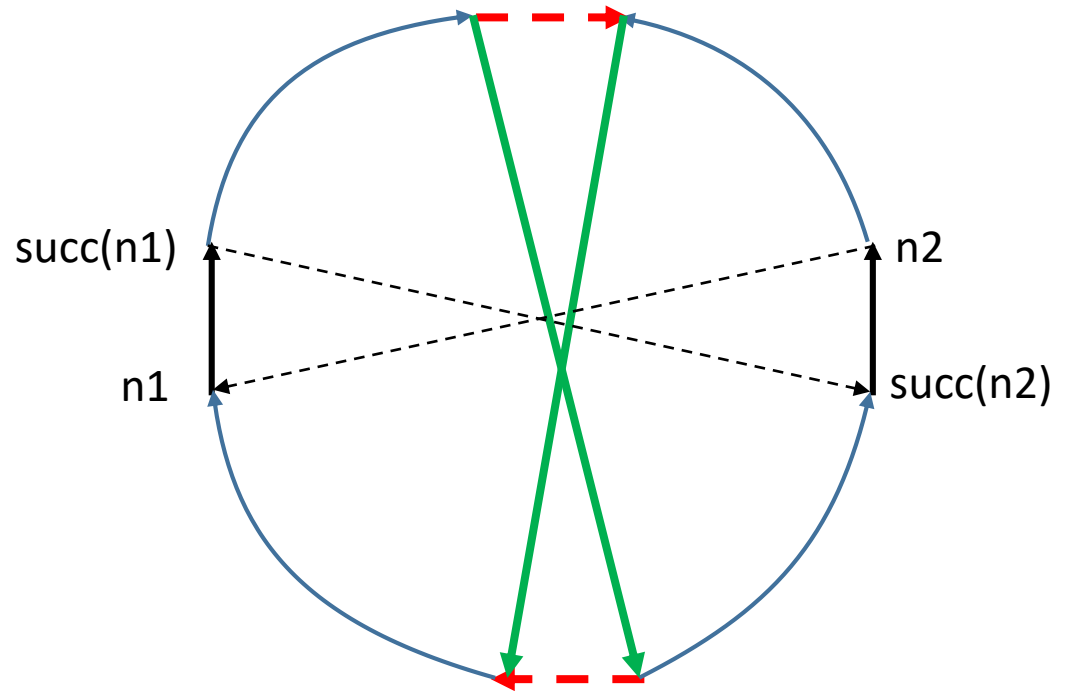
- Ruch może nie być aplikowalny (nie da się go zastosować), ze względu na zmiany w rozwiązaniu spowodowane przez inne ruchy

Przykład dla TSP

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

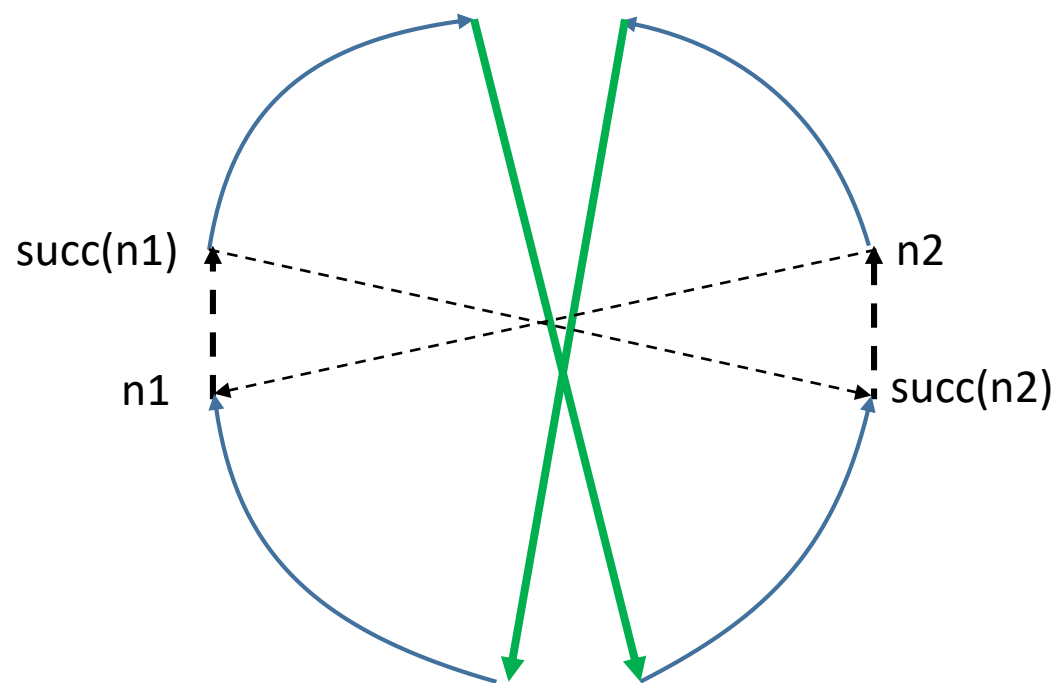


Zapamiętujemy ruch
Dodajemy krawędzie $(n1, n2)$, $(succ(n1), succ(n2))$

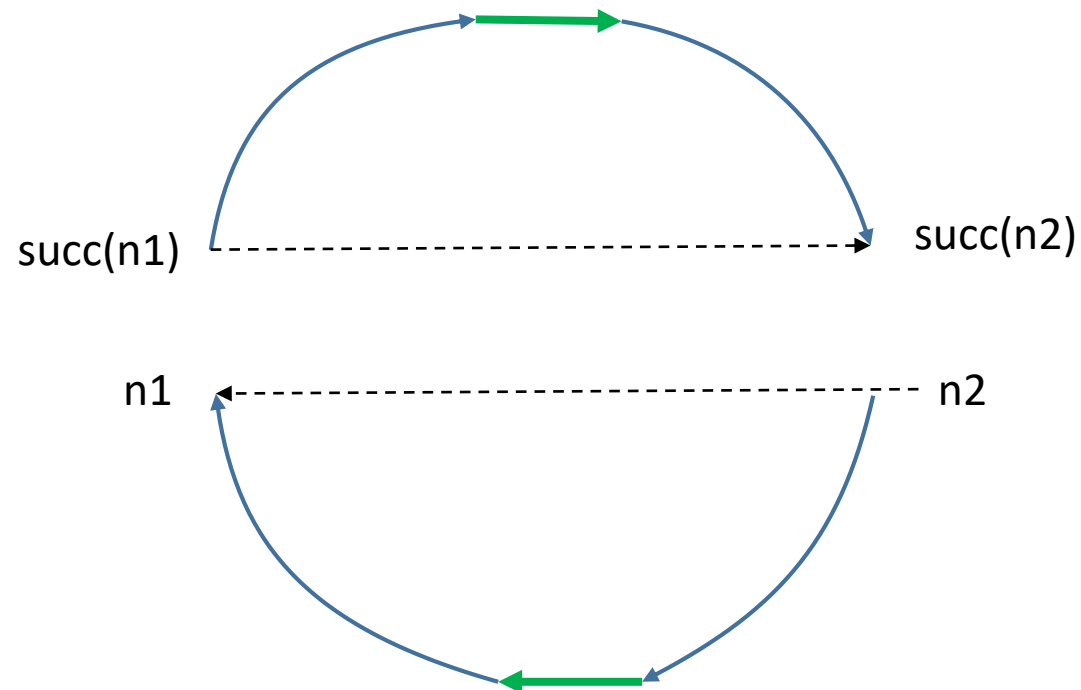


Wykonujemy inny ruch

Przykład dla TSP



Po wykonaniu innego ruchu



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

Wnioski z powyższego przykładu

- Musimy też brać pod uwagę kierunek przechodzenia krawędzi w bieżącym rozwiązaniu
- Każdy inny ruch może tę kolejność zmienić
- 3 sytuacje (kiedy przeglądamy ruchy od najlepszego):
 - Usuwane krawędzie nie występują już w bieżącym rozwiązaniu
 - -> usuwamy ruch z *LM*
 - Usuwane krawędzie występują w bieżącym rozwiązaniu w różnym od zapamiętanego kierunku – może być aplikowalny w przyszłości
 - -> zostawiamy ruch w *LM*, ale nie aplikujemy, przechodzimy dalej
 - Usuwane krawędzie występują w bieżącym rozwiązaniu w tym samym kierunku (także obie odrócone)
 - -> aplikujemy i usuwamy z *LM*
- Oceniając nowe ruchy trzeba też uwzględnić (dodawać do *LM*) także ruchy dla odróconego (względem obecnego) względnego kierunku krawędzi – nie są one aplikowalne do bieżącego rozwiązania, ale mogą się stać aplikowalne po wykonaniu innego ruchu

Wersja dla TSP i wymiany dwóch krawędzi

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

Wygeneruj rozwiązanie startowe \mathbf{x}

powtarzaj

przejrzyj wszystkie **nowe** ruchy (uwzględniając także ruchy z odwróconym względnym kierunkiem krawędzi) i dodaj do LM ruchy przynoszące poprawę

Przeglądaj ruchy m z LM od najlepszego do znalezienia aplikowalnego ruchu

Jeżeli co najmniej jednej z usuwanych krawędzi nie ma już w rozwiązaniu, usuń m z LM

Jeżeli obie usuwane krawędzie są w rozwiązaniu ale w odwróconym względnym kierunku (względem zapamiętanego), pomiń m ale pozostaw go w LM

jeżeli znaleziono aplikowalny ruch m **to**

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

dopóki nie znaleziono aplikowalnego ruchu m po przejrzeniu całej listy LM

Globalna pamięć delt

- Do tej pory zakładaliśmy zapamiętywanie bieżących delt w ramach jednego uruchomienia LP
- Lokalne przeszukiwanie może być jednak wielokrotnie powtarzane w ramach metody wyższego rzędu
- Ruchy i ich delty mogą się więc powtarzać w trakcie różnych uruchomień LP
- Można więc zapamiętywać wszystkie już obliczone delty w pamięci globalnej (która jednak może być bardzo duża)
 - Hashowanie w celu poprawy efektywności
 - Techniki zarządzania pamięcią, np. usuwanie najrzadziej używanych delt

Globalna pamięć delt

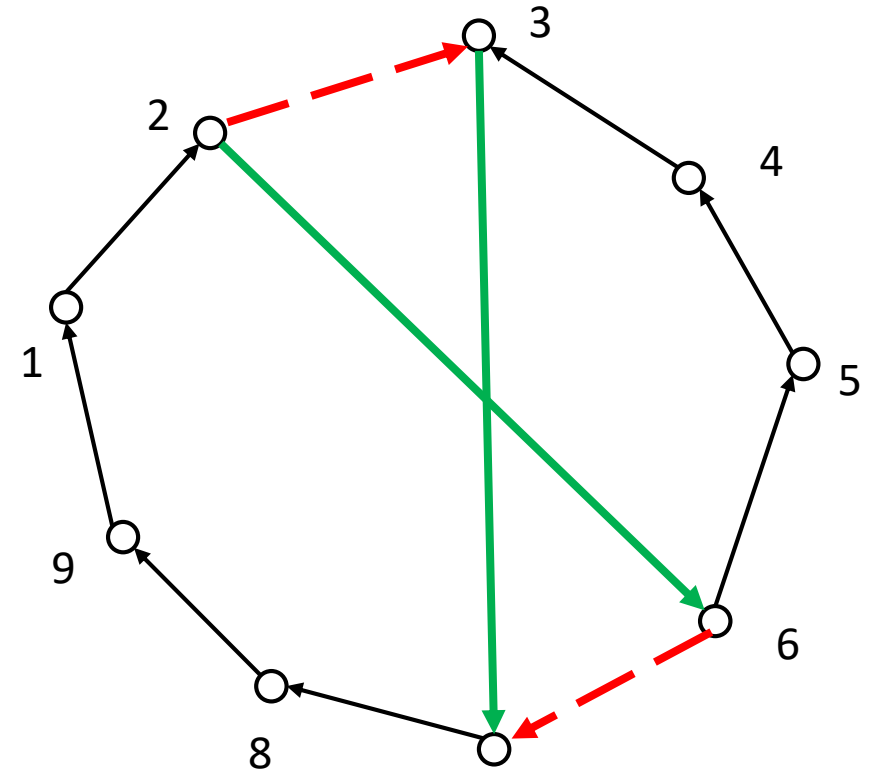
- Np. dla problemu komiwojażera istnieje $O(n^4)$ różnych ruchów wymiany dwóch krawędzi (czwórek krawędzi), przy $(n - 1)!$ rozwiązań

Pomijanie nieobiecujących ruchów na podstawie reguł heurystycznych

- Idealnie byłoby oceniać tylko ten ruch, który zostanie wykonany
 - Przynoszący poprawę w wersji zachłannej
 - Najlepszy w wersji stromej
- W praktyce z reguły niemożliwe, ale można wiele nieobiecujących ruchów pominąć (nie oceniać) 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
- W wersji idealnej skracamy czas obliczeń bez pogorszenia jakości
 - W praktyce, jeżeli pewne dobre ruchy nie są na liście kandydackich, następuje pogorszenie jakości znajdowanych rozwiązań

Przykład TSP (podobnie VRP)

- Dla każdego wierzchołka tworzymy listę o długości $n' \ll n$ najbliższych wierzchołków – i wynikających z tego krawędzi kandydackich
- Ruchy kandydackie wprowadzają co najmniej jedną krawędź prowadzącą do jednego z najbliższych wierzchołków



Standardowe przeglądanie sąsiedztwa TSP – wymiana dwóch krawędzi – wersja stroma

Dla każdego wierzchołka $n1$ od 0 do $N-1$

Dla każdego wierzchołka $n2$ od $n1+1$ do $N-1$

jeżeli łuki $n1\text{-succ}(n1)$ i $n2\text{-succ}(n2)$ nie są sąsiednie

Sprawdź czy ruch $(n1, n2)$ przynosi poprawę i jest najlepszym dotąd
znalezionym

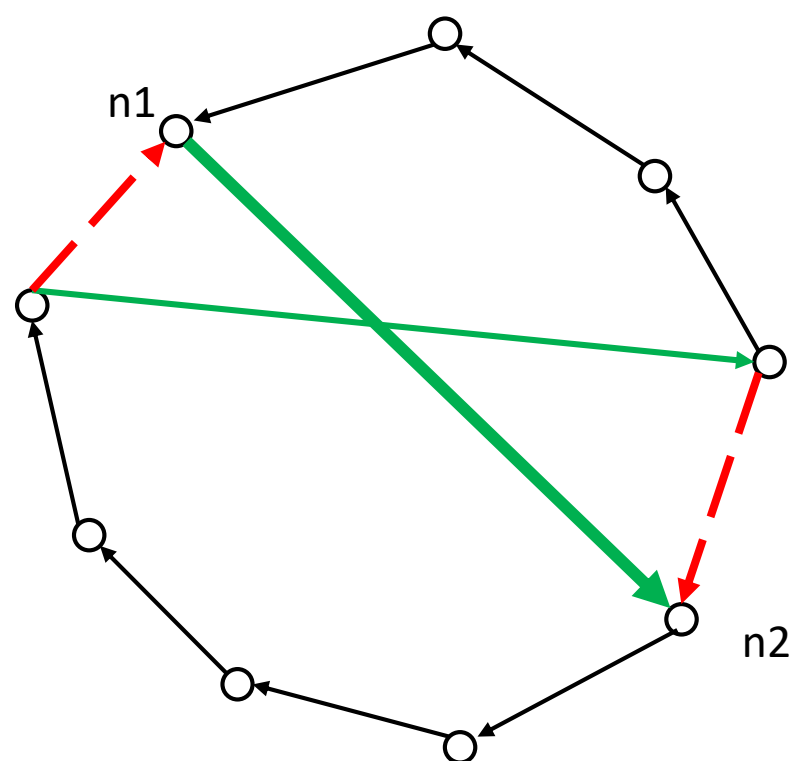
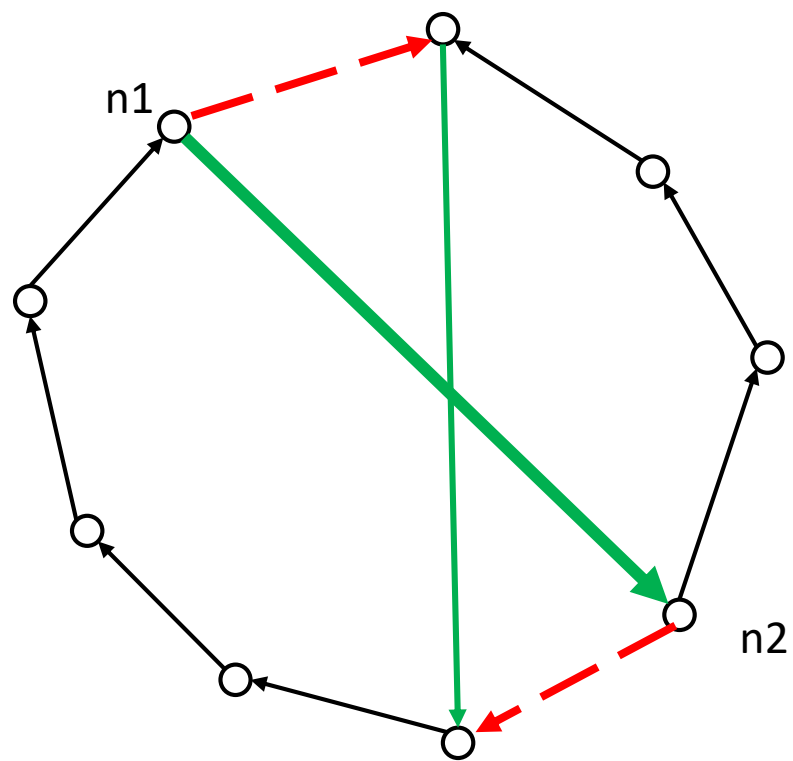
Przeglądanie sąsiedztwa TSP z ruchami kandydackimi – wymiana dwóch krawędzi – wersja stroma

Dla każdego wierzchołka n_1 od 0 do $N-1$

Dla każdej wierzchołka n_2 z listy najbliższych wierzchołków do n_1

sprawdź wszystkie ruchy polegające do dodaniu krawędzi n_1-n_2 i usunięciu jednej z obecnych krawędzi łączących n_1

Dwa możliwe ruchy



Uczenie się ruchów kandydackich

- Uczenie się ze zbioru rozwiązań:
 - Generujemy pewną liczbę rozwiązań stosując heurystykę lub lokalne przeszukiwanie bez ruchów kandydackich
 - Każda krawędź, która wystąpi choć raz (pewną liczbę/procent razy) w jednym z tych rozwiązań staje się krawędzią kandydacką
- Wykorzystanie metod wyższego rzędu, w których zanurzone jest LP
- Pamięć długoterminowa, np.
 - Lista dobrych krawędzi w TSP
 - Lista krawędzi występujących w ruchach przynoszących poprawę
- Wykorzystanie informacji z operatorów rekombinacji
 - Np. do ruchów kandydackich zalicza się ruchy wprowadzające krawędzie występujące u dowolnego z rodziców (nawet jeżeli nie znalazły się one w potomku)

Techniki zaawansowane poprawy efektywności LP

- Np. przeglądanie sąsiedztwa o rozmiarze wykładniczym w czasie wielomianowym

Wygeneruj rozwiązanie \mathbf{x}

powtarzaj

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

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

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

dopóki nie znaleziono lepszego rozwiązania

Problem optymalizacji

- W niektórych wypadkach zastosowanie programowania dynamicznego pozwala wybrać najlepszy ruch z wykładniczego sąsiedztwa w czasie pseudo wielomianowym

Techniki zaawansowane (przyszłościowe)

- Zastosowanie rozszerzeń kwantowego algorytmu Grovera dla optymalizacji black-box
- $\Theta(\sqrt{|M(\mathbf{x})|})$, np. $O(n)$ zamiast $O(n^2)$ przy wymianie dwóch krawędzi w TSP

Wady lokalnego przeszukiwania

- Kończą działanie w optimum lokalnym, które nie musi być optimum (globalnym)
 - Chyba, że sąsiedztwo obejmuje wszystkie rozwiązania co prowadzi do pełnego przeglądu
- Jakość optimów lokalnych zależy od rozwiązań startowych

Jak unikać wad lokalnego przeszukiwania?

- Zmiana/rozszerzenie definicji sąsiedztwa w celu przeglądania większej liczby rozwiązań
- Akceptowanie do pewnego stopnia rozwiązań pogarszających
- Uruchamianie LP z różnych punktów startowych
- Generowanie dobrych rozwiązań startowych
 - Często (z reguły?) lepsze rozwiązanie startowe to zarówno lepsze optimum lokalne, jak i krótszy czas obliczeń, może jednak oznaczać ukierunkowanie (bias) przeszukiwania

Multiple start local search – Lokalne przeszukiwanie z wieloma punktami startowymi

Powtarzaj

Wygeneruj zrandomizowane rozwiązanie startowe \mathbf{x}

Lokalne przeszukiwanie (\mathbf{x})

Do spełnienia warunków stopu

Zwróć najlepsze znalezione rozwiązanie

- **Uwaga!** Pojedyncze uruchomienie MSLS obejmuje wiele uruchomień LP, czyli wiele uruchomień LP traktujemy jako jedno uruchomienie MSLS

Multiple start local search – MSLS

- Najprostsze rozszerzenie LP
- Powinno być punktem odniesienia dla wszystkich bardziej zaawansowanych metod, które powinny być lepsze niż MSLS

Variable neighborhood Local Search - Lokalne przeszukiwanie ze zmiennym sąsiedztwem

- Motywacja
 - Lokalne optimum dla pewnego sąsiedztwa nie musi być lokalnym optimum dla innego rodzaju sąsiedztwa – można więc je nadal poprawić stosując inny rodzaj sąsiedztwa
- Globalne optimum jest lokalnym optimum dla wszystkich możliwych sąsiedztwa

Variable neighborhood Local Search

Wejście – lista (definicji) sąsiedztw (N_1, N_2, \dots)

Wygeneruj rozwiązanie początkowe x

Dla wszystkich sąsiedztw $N=(N_1, N_2, \dots)$

$x :=$ Lokalne przeszukiwanie (x, N)

Variable neighborhood Local Search

- Najczęściej zaczynamy od sąsiedztw mniej złożonych i przechodzimy do bardziej złożonych, np. dla TSP najpierw wymiana dwóch krawędzi, potem trzech krawędzi, itd.
- Typowy wpływ wielkości sąsiedztwa (choć są też inne czynniki):
 - Mniej złożone: krótszy czas obliczeń, gorsze optimum lokalne
 - Bardziej złożone: dłuższy czas obliczeń, lepsze optimum lokalne
- Dzięki powyższemu podejściu, szybko uzyskujemy stosunkowo dobre rozwiązanie (nie tracąc czasu na przeglądanie zbyt wielu rozwiązań sąsiednich), które możemy dalej poprawić w stosunkowo niewielkiej liczbie kroków bardziej złożonego sąsiedztwa

Iterated local search - Iteracyjne przeszukiwanie lokalne

Wygeneruj rozwiązanie początkowe \mathbf{x}

$\mathbf{x} :=$ Lokalne przeszukiwanie (\mathbf{x})

Powtarzaj

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

 Perturbacja (\mathbf{y})

$\mathbf{y} :=$ Lokalne przeszukiwanie (\mathbf{y})

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

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

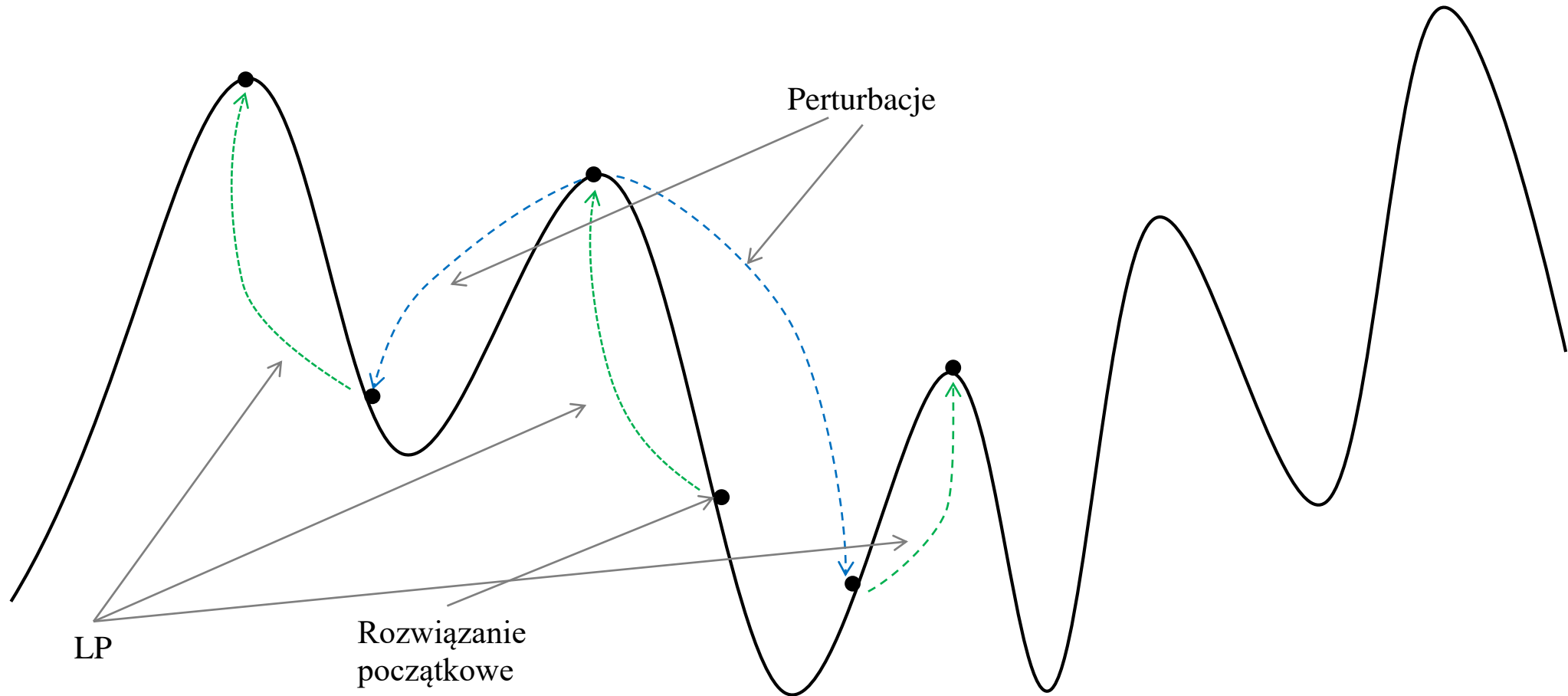
Do spełnienia warunków stopu

- W pewnym sensie lokalne przeszukiwanie w zbiorze optimów lokalnych (z bazowym sąsiedztwem) z dużym, losowo próbkowanym sąsiedztwem

Pojęcie perturbacji

- Zrandomizowana modyfikacja wykraczająca poza sąsiedztwo wykorzystywane w LP – nieosiągalna w jednym ruchu, np.:
 - Złożenie kilku ruchów
 - Większy ruch, np. wymania większej liczby krawędzi dla TSP
- Wielkość perturbacji
 - Zbyt mała powoduje powracanie do tego samego optimum lokalnego
 - Zbyt duża pogarsza jakość rozwiązań startowych dla LP, w skrajnym wypadku oznacza tworzenie rozwiązań losowych

Iterated local search - Iteracyjne przeszukiwanie lokalne



Adaptive multistart local search – Adaptacyjne lokalne przeszukiwanie z wieloma punktami startowymi

- Po każdym uruchomieniu LP aktualizowana jest tablica częstości występowania poszczególnych elementów rozwiązania (np. krawędzi) w lokalnych optimach
- Nowe rozwiązania startowe są tworzone przy wykorzystaniu tych danych – większe prawdopodobieństwo wybrania częstych elementów (np. krawędzi)
 - Np. zrandomizowana heurystyka konstrukcyjna, która przy ocenie wstawianym elementów bierze pod uwagę częstość ich występowania

Adaptive multistart local search - przykład

Effective neighborhood search with optimal splitting and adaptive memory for the team orienteering problem with time windows, Youcef Amaroucheab, Rym Nesrine Guibadjc, Elhadja Chaalalb, Aziz Moukrim, Computers & Operations Research, Volume 123, November 2020, 105039

- Team orienteering problem with time windows – rodzaj problemu marszrutyzacji pojazdów
- Tworzony jest duży zbiór różnorodnych tras, z których budowane są nowe rozwiązania startowe
- Połączenie z ILS

Large-scale neighborhood search – wielkoskalowe przeszukiwanie sąsiedztwa

- Ruch polega na złożeniu dwóch metod:
 - Destroy – usuwanie pewnych elementów z rozwiązania
 - Np. usuwa część wierzchołków z tras problemu VRP
 - Często łączenie losowości z wyborem heurystycznym – staramy się usuwać wierzchołki, krawędzie, fragmenty tras, które mogą być źle przydzielone, np. różne w różnych rozwiązaniach
 - Repair – ukierunkowana (funkcją celu) naprawa rozwiązania
 - Np. heurystyka zachłanna wstawiające usunięte wcześniej wierzchołki/krawędzie
- W ogólności taki ruch może powodować duże zmiany rozwiązania

Large-scale neighborhood search

Wygeneruj rozwiązanie początkowe \mathbf{x}

$\mathbf{x} :=$ Lokalne przeszukiwanie (\mathbf{x}) (*opcja*)

Powtarzaj

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

Destroy (\mathbf{y})

Repair (\mathbf{y})

$\mathbf{y} :=$ Lokalne przeszukiwanie (\mathbf{y}) (*opcja*)

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

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

Do spełnienia warunków stopu

Adaptive Large-scale neighborhoodsearch

- Możemy wykonywać wiele różnych typów ruchów typu destroy i repair
- Ich prawdopodobieństwo jest automatycznie modyfikowane na podstawie efektów stosowania poszczególnych operatorów (np. prawdopodobieństwa uzyskania nowego lepszego rozwiązania)
- Rodzaj hiperheurystyki

Hiperheurystki

- W dowolnej iteracyjnej heurystyce mamy możliwość stosowania różnych operatorów (np. sąsiedztwa, perturbacji, destroy/repair, rekombinacji...)
- Ich prawdopodobieństwo jest automatycznie modyfikowane na podstawie efektów stosowania poszczególnych operatorów (np. prawdopodobieństwa uzyskania nowego lepszego rozwiązania)
- Mogą to być proste reguły statystyczne lub bardziej zaawansowane techniki AI, np. uczenie ze wzmocnieniem

Kryteria stopu

- Liczba iteracji / czas obliczeń
- Kryterium dynamiczne – brak poprawy w zadanej liczbie iteracji /
zadany czas obliczeń

Simulated annealing – symulowane wyżarzanie

- Fizyczne wyżarzanie
 - Szybkie podniesienie temperatury ciała stałego w pobliże (ale poniżej) temperatury topnienia
 - Przy wysokiej temperaturze atomy/cząsteczki łatwo zmieniają swoje położenie w strukturze krystalicznej
 - Powolne obniżanie temperatury
 - Przy obniżaniu temperatury atomy/cząsteczki coraz trudniej zmieniają swoje położenie w strukturze krystalicznej
 - Prowadzi do ułożenia się atomów/cząsteczek w regularnej strukturze krystalicznej – stan o minimalnej energii

Fizyczne wyżarzanie – analogie z optymalizacją

- Struktura ułożenia atomów/cząsteczek – rozwiązania
- Energia – f. celu (minimalizowana)
- Idealna struktura krystaliczna – optimum globalne
- Temperatura – parametr kontrolny
- Szybkie schładzanie – lokalna optymalizacja
- Powolne schładzanie – symulowane wyżarzanie

Symulowane wyżarzanie

Wygeneruj rozwiązanie początkowe \mathbf{x}

Temperatura $T := T_0$

powtarzaj

powtarzaj L razy

 Wygeneruj losowe $\mathbf{y} \in N(\mathbf{x})$

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

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

w przeciwnym wypadku

jeżeli $\exp((f(\mathbf{y}) - f(\mathbf{x})) / T) > \text{random}[0,1)$

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

 obniż T

dopóki $T \leq T_K$

Zwróć najlepsze wygenerowane rozwiązanie

Warunek akceptacji

- Lepsze rozwiązania są zawsze akceptowane
- Gorsze rozwiązania są akceptowane z pewnym prawdopodobieństwem
 - Im mniejsze pogorszenie wartość f. celu, tym większe prawdopodobieństwo
 - Im większa temperatura T , tym większe prawdopodobieństwo

Bardzo wysoka temperatura – losowe błędzenie

Bardzo niska temperatura – lokalne przeszukiwanie

SW - pomiędzy

Zbieżność symulowanego wyżarzania

- Przy wystarczająco dużej temperaturze początkowej i wystarczająco powolnym obniżaniu temperatury prawdopodobieństwo uzyskania optimum globalnego dąży do 1
- Wynik raczej teoretyczny, w praktyce wymagana liczba ruchów podobna do pełnego przeglądu

Typowe sposób obniżania temperatury

$$T_{k+1} = \alpha T_k, \quad k = 1, 2, \dots$$

$$T_{k+1} = \alpha^k T_0$$

α jest stałą mniejszą ale bliską 1, z reguły [0.8 – 0.99]

Temperatura początkowa

- Podejście tradycyjne
 - Temperatura początkowa powinna być wystarczająco wysoka, żeby zapewnić akceptację większości ruchów
 - Jeżeli zbyt niska, podnoszenie temperatury
 - Zależy od typowej zmiany wartości f. celu Δf
 - Np. dla $\Delta f = 1000$, i prawdopodobieństwa akceptacji 0,98, $T_0 \approx 250$
- W praktyce często lepsze efekty uzyskuje się zaczynając od znacznie niższych temperatur

Temperatura końcowa

- Odpowiednio małe prawdopodobieństwo akceptacji

Great Deluge – Wielka powódź (deterministyczne SW?)

wygeneruj rozwiązanie początkowe \mathbf{x}

Poziom wody $P := P_0$

powtarzaj

powtarzaj L razy

 Wygeneruj losowe $\mathbf{y} \in N(\mathbf{x})$

jeżeli $f(\mathbf{y}) > P$ **to**

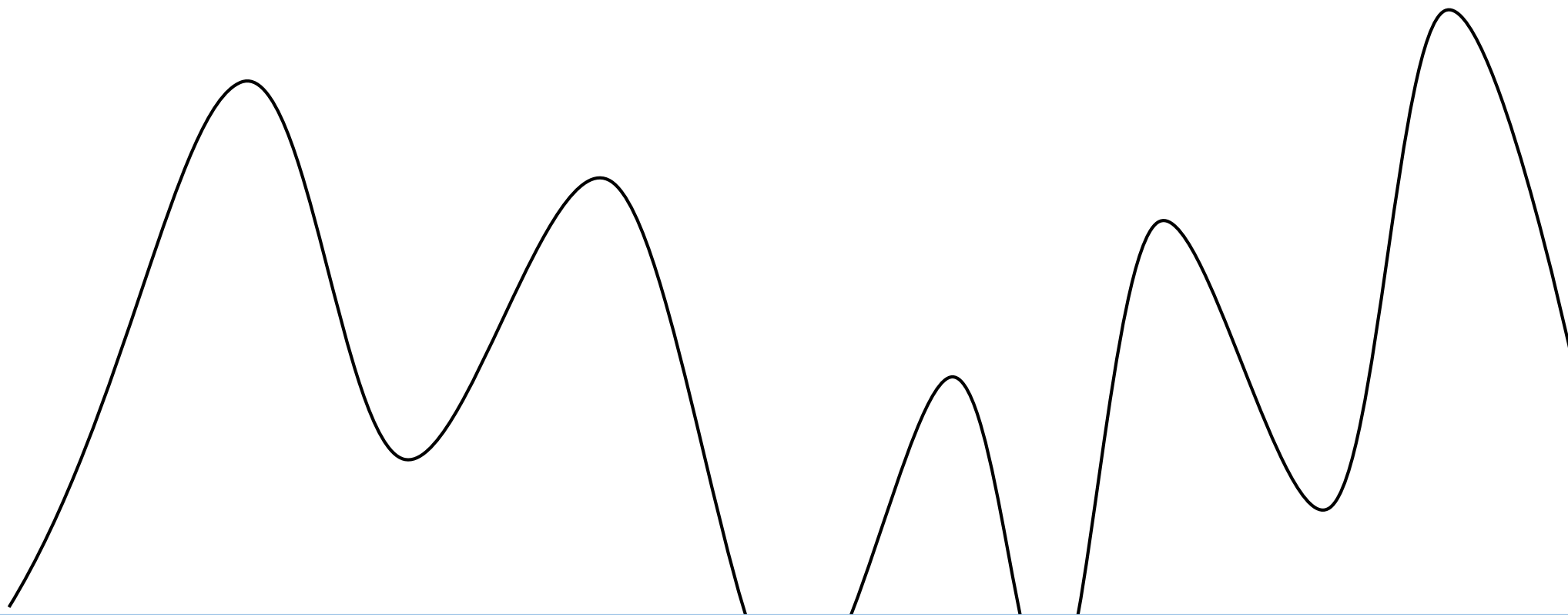
$\mathbf{x} := \mathbf{y}$

 podwyższ P

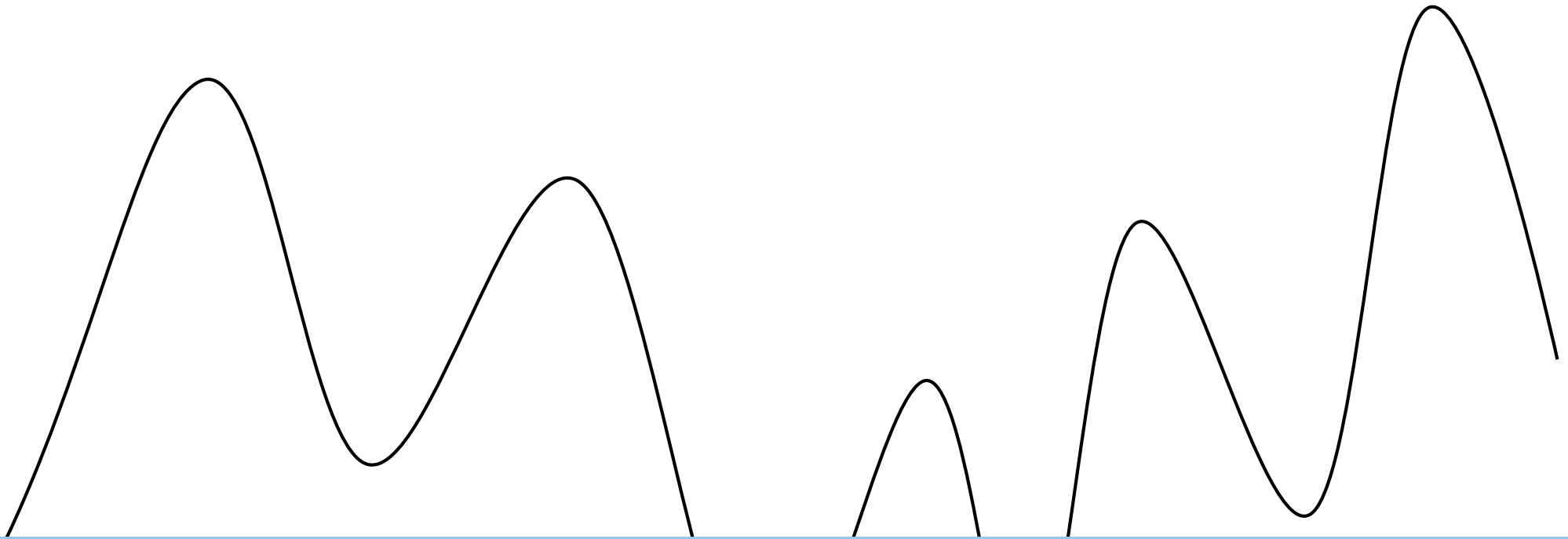
dopóki $P \leq P_K$

Zwróć najlepsze wygenerowane rozwiązanie

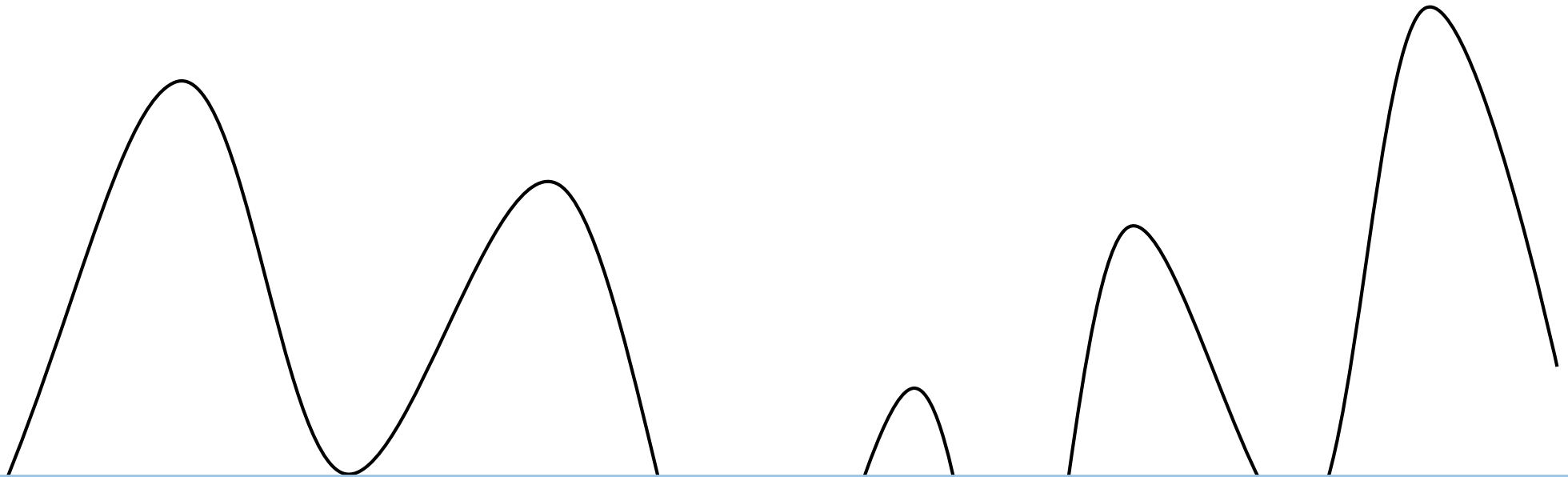
Great Deluge



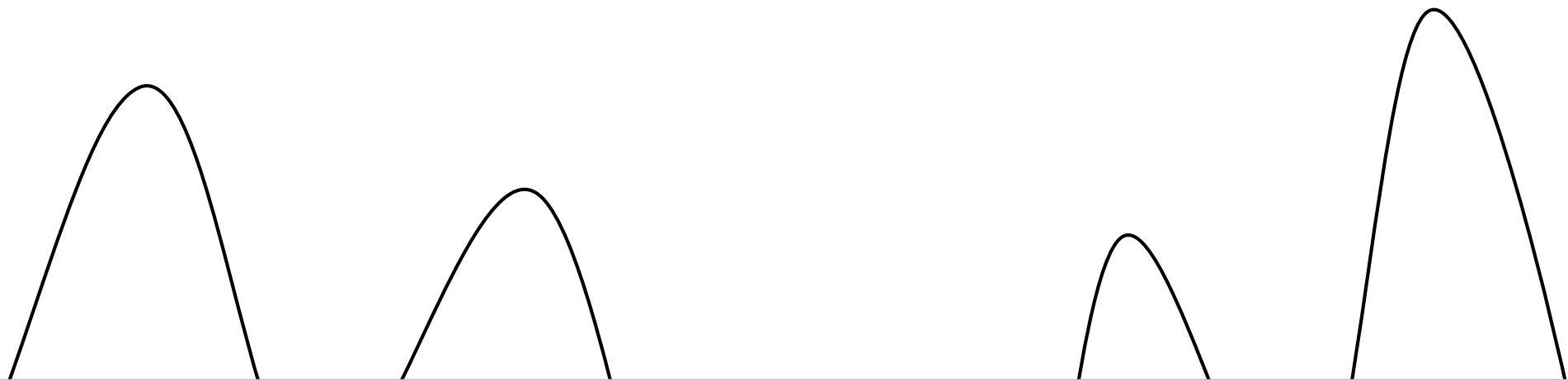
Great Deluge



Great Deluge



Great Deluge



Tabu search – przeszukiwanie Tabu

- Punktem wyjścia jest LP w wersji stromej
- Pytanie czy można zrezygnować z warunku, że nowe rozwiązanie musi być lepsze od bieżącego?
- To jednak prowadzi do pojawienia się cykli (powrotów do tych samych rozwiązań)

wygeneruj rozwiązanie początkowe \mathbf{x}

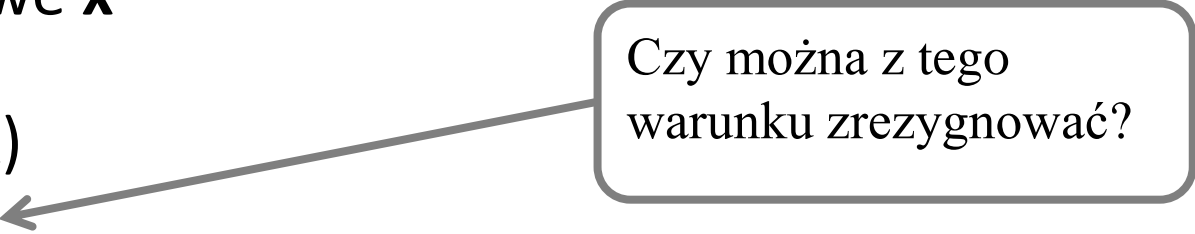
powtarzaj

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

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

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

dopóki nie znaleziono lepszego rozwiązania



Czy można z tego
warunku zrezygnować?

Problem cykli

- Proste wyeliminowanie powyższego warunku prowadziłoby do cykli – powrotów do tych samych rozwiązań
- Najprostszy sposób unikania cykli:
 - Zapamiętuj wszystkie odwiedzone dotąd rozwiązania
 - Rozwiązania te stają się zakazane – Tabu
 - Dla poprawy efektywności można zachowywać wartości haszowe i wyszukiwanie przez połowienie
 - Może prowadzić do „krążenia” wokół bardzo podobnych rozwiązań

Idea listy Tabu

- Lista zakazanych rozwiązań/ruchów/elementów ruchów

Algorytm przeszukiwania Tabu – Tabu search

wygeneruj rozwiązanie początkowe \mathbf{x}

lista Tabu $T := \emptyset$

powtarzaj

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

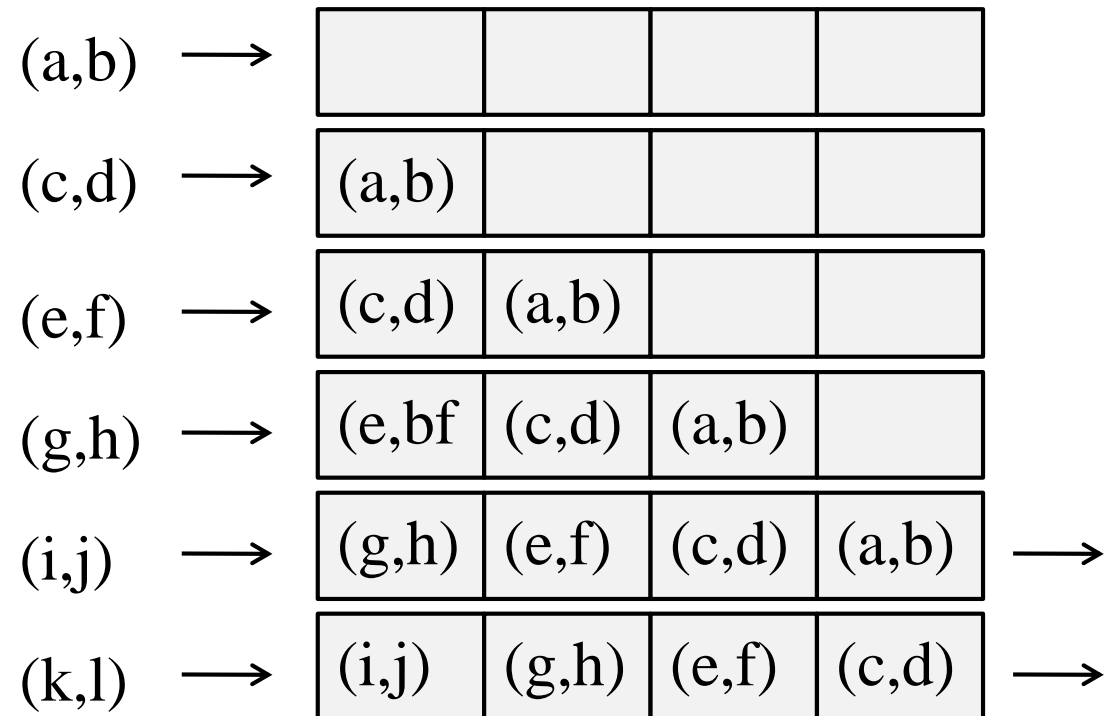
$\mathbf{x} := m(\mathbf{x})$

uaktualnij listę Tabu (T, m)

dopóki warunek stopu

Zwróć najlepsze wygenerowane rozwiązanie

Lista Tabu jako kolejka FIFO



Decyzje projektowe

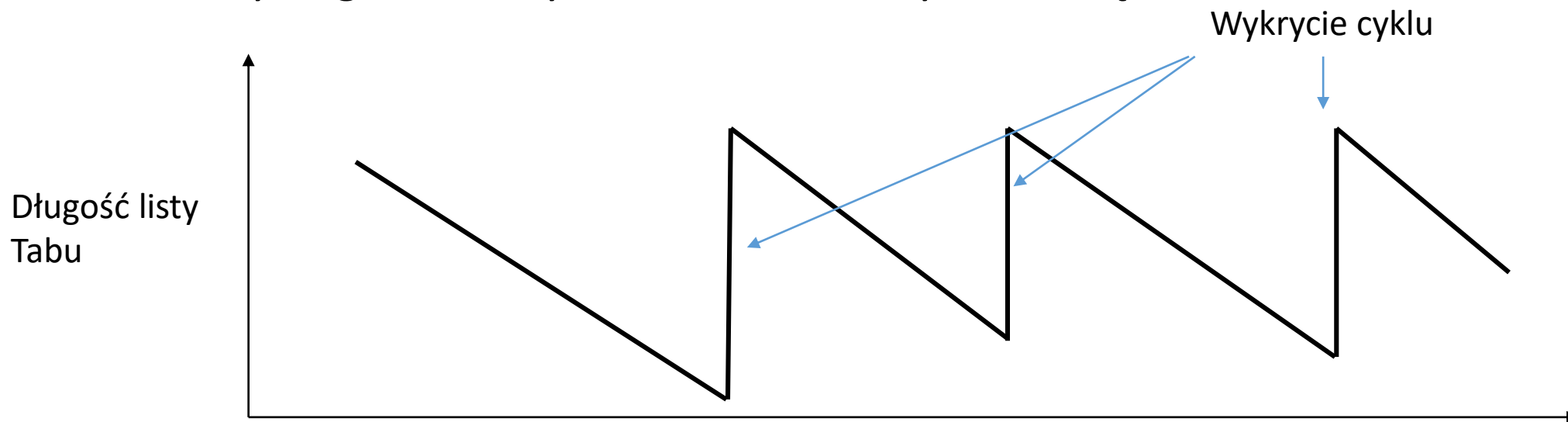
- Co przechowywać na liście Tabu?
 - Rozwiązania (patrz uwagi wcześniejsze)
 - Ruchy
 - Składowe ruchów (np. pojedyncze krawędzie, wierzchołki)
- Jak blokować ruchy?
 - Dokładne dopasowanie
 - Zakazane są tylko ruchy znajdujące się na liście Tabu, lub ruchy odwrotne
 - Stosunkowy mały zakres ograniczeń
 - Wymaga dłuższej listy Tabu
 - Ruchy częściowo się pokrywające się
 - Np. ruch (a,b), blokuje wszystkie ruchy, w których występuje a lub b
 - Naturalne przy przechowywaniu składowych ruchów
 - Stosunkowy duży zakres ograniczeń
 - Wymaga krótszej listy Tabu
- Długość listy Tabu

Długość listy Tabu

- Krótka lista
 - Bardziej „agresywny” algorytm
 - Ryzyko wpadnięcia w cykl
- Długa lista
 - Ryzyko pomijania zbyt wielu ruchów
 - Może nawet uniemożliwiać dojście do jakiegokolwiek lokalnego optimum

Reactive Tabu Search – automatyczne modyfikowanie długości listy Tabu

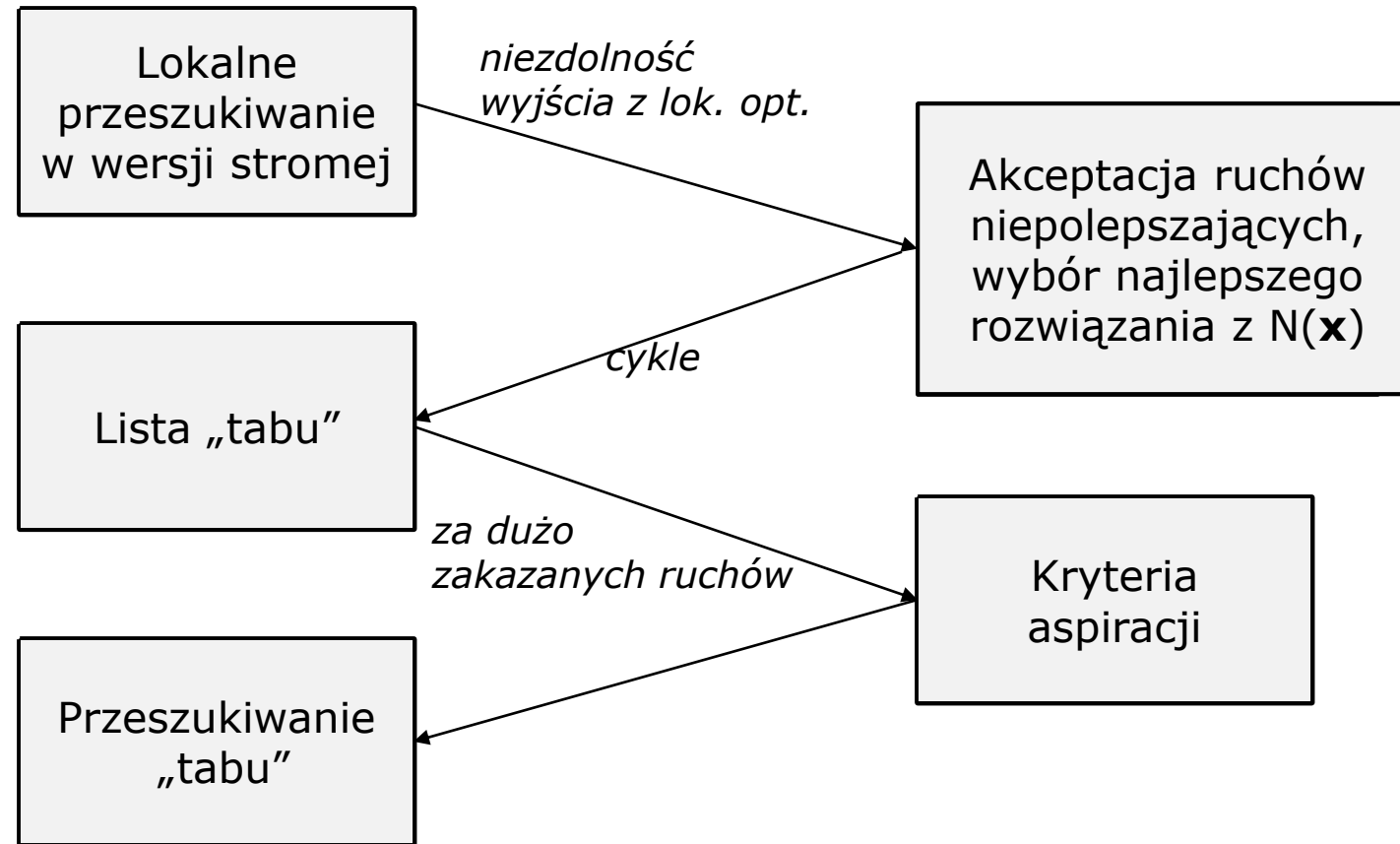
- Stałe, powolne zmniejszanie długości listy
 - Co L iteracji zmniejszana jest długość listy Tabu
- Zwiększanie długości listy w przypadku wpadnięcia w cykl
 - Wymaga zachowywania odwiedzonych rozwiązań



Kryteria aspiracji

- Lista Tabu może blokować zbyt wiele ruchów/rozwiązań
- Ruchy/rozwiązania Tabu, które są bardzo dobre – spełniają kryteria aspiracji – mogą zostać zaakceptowane
- Najbardziej oczywiste kryterium aspiracji – akceptuj rozwiązania lepsze od najlepszego znalezione dotąd, nawet jeżeli są Tabu
 - Takie rozwiązania na pewno nie zostały dotąd odwiedzone
- Można też akceptować np. stosunkowo dobre rozwiązania (ale nie lepsze od najlepszego) wyraźnie różne od poprzednich

Naturalny sposób powstania algorytmu



Pamięć długoterminowa – long term memory

- Koncepcja niezależna od Tabu search, choć w tym kontekście oryginalnie zaproponowana
- Przykłady:
 - Tablica częstości występowania poszczególnych krawędzi w odwiedzanych rozwiązaniach
 - Tablica częstości ruchów przynoszących poprawę
- Zastosowania:
 - Restarty – tworzenie nowych rozwiązań:
 - Dywersyfikacja – rozwiązania startowe odległe od dotychczas odwiedzanych
 - Intensyfikacja – rozwiązania startowe podobne do najlepszych dotąd odwiedzanych
 - Definiowanie ruchów kandydackich

Łączenie/hybrydyzacja różnych metod bazujących na LP

- Np. symulowane wyżarzanie, przeszukiwanie Tabu, Lokalne przeszukiwanie ze zmiennym sąsiedztwem w ramach iteracyjnego przeszukiwania lokalnego
- Iteracyjne przeszukiwanie lokalne, symulowane wyżarzanie, przeszukiwanie Tabu w ramach wielkoskalowego przeszukiwania sąsiedztwa
- itd...

