|  |  |
| --- | --- |
| Sprawozdanie 1 - Heurystyki konstrukcyjne | |
| **Imie Nazwisko** | **Nr indexu** |
| Patryk Jedlikowski  Mikołaj Sienkiewicz | 136723  136309 |

# Opis zadania

Zadanie polegało na zaimplementowaniu, a następnie porównaniu różnych metod heurystycznych (z rodziny heurystyk konstrukcyjnych) na przykładzie problemu komiwojażera (TSP). Dodatkową specyfiką problemu była jednoczesna budowa dwóch 50-wierzchłokowych cykli na bazie popularnych 100-wierzchołkowych zbiorów (kroa100 i krob100). Każde zaimplementowane podejście było uruchamiane 100 razy w celu uśrednienia wyników i zaprezentowania ostatecznych wyników.

# Pseudokody zaimplementowanych podejść

## 1. Algorytm zachłanny inspirowany metodą najbliższego sąsiada

Inicjalizacja zmiennych: (min\_result,max\_result,results[],dataset)

distance=Oblicz\_macierz\_dystansu(dataset)

distance(przekatna)=nieskonczonosc

Wybierz losowo wierzchołki startowe dla cycleA oraz cycleB

distance(kolumna=wierzchołki\_startowe) = nieskonczonosc

Dla i od 0 do 98:

    wybierz co drugi cykl do rozbudowy:

        kandydaci=[]

        Dla każdego licznik\_iteracji,index\_wierzchołka z wybranego cyklu:

            kandydaci.append([index\_NN,wartosc\_NN,licznik\_iteracji])

        najlepszy=min(kandydaci,klucz=kandydaci[1])

        wybrany\_cykl.insert(najlepszy[2],index\_wstawienia=najlepszy[0])

        distance(kolumna=najlepszy[0]) = nieskonczonosc

## 2. Algorytm zachłanny inspirowany metodą rozbudowy cyklu

Wybierz losowo wierzchołki startowe dla cycleA oraz cycleB

Dla i od 0 do 98:

    wybierz co drugi cykl do rozbudowy:

        minDistance = nieskończoność

        for vertex in range(100):

            if vertex w użyciu:

                pomiń bieżącą iteracje

            distance, newCycle = findNewCycle(vertex)

            if distance < minDistance:

                minDistance = distance

                minCycle = newCycle

        Podmień cycle na minCycle w aktualnie rozbudowywanym cyklu

Funkcja - findNewCycle(vertex):

    minDistance = nieskończoność

    for i in range(długość rozbudowywanego cyklu + 1):

        wstaw sprawdzany vertex w miejsce "i" do aktualnego cyklu

        distance = obliczDługośćCyklu()

        if distance < minDistance:

            minDistance = distance

            minVertex = vertex

            minPath = path

    return minDistance, minPath

## 3. Algorytm typu „regret heuristics” (2-regret)

Wybierz losowo wierzchołki startowe dla cycleA oraz cycleB

Dla i od 0 do 98:

    wybierz co drugi cykl do rozbudowy:

        minDistance = np.inf

        Dla vertex od 0 do 100:

            if vertex w użyciu:

                pomiń bieżącą iteracje

            regret, newCycle = findNewCycle(vertex)

            if regret > maxRegret:

                maxRegret = regret

                minCycle = newCycle

        Podmień cycle na newCycle w aktualnie rozbudowywanym cyklu

Funkcja - findNewCycle(vertex):

    minDistance = np.inf

    minSecondDistance = np.inf

    Dla i od 0 do długość rozbudowywanego cyklu + 1:

        wstaw sprawdzany vertex w miejsce "i" do aktualnego cyklu

        distance = obliczDługośćCyklu()

        if distance < minDistance:

            minDistance = distance

            minNewCycle = newCycle

            pomiń bieżącą iteracje

        if distance < minSecondDistance:

            minSecondDistance = distance

    return minDistance, minPath

# Tabela prezentująca wyniki eksperymentu obliczeniowego

|  |  |  |
| --- | --- | --- |
|  | Instancja 1 - kroa | Instancja 2 - krob |
| Metoda najbliższego sąsiada | 32994 (28762 - 35173) | 32546 (29848 - 34424) |
| Metoda rozbudowy cyklu | 28593 (26869 - 29981) | 181118 (168712 - 187804) |
| Metoda heurystyki 2-żal | 31250 (29167 - 33825) | 175306 (159430 - 190875) |

# Wizualizacje najlepszych rozwiązań

|  |  |
| --- | --- |
| **Najbliższy sąsiad – kroa100** | **Najbliższy sąsiad – krob100** |
|  |  |

|  |  |
| --- | --- |
| **Rozbudowa cyklu – kroa100** | **Rozbudowa cyklu – krob100** |
|  |  |

|  |  |
| --- | --- |
| **2 żal – kroa100** | **2 żal – krob100** |
|  |  |

# Wnioski

Zadanie uświadamia jak bardzo wybór konkretnej heurystyki wpływa na ostateczny wynik rozwiązania. Mierząc się z jakimkolwiek problemem, który decydujemy się rozwiązać takimi metodami, istotne jest w pierwszej kolejności przeanalizowanie istniejących podejść i refleksja czy w naszym przypadku, któraś z heurystyk nie dominuje pozostałych.

Implementacja najbliższego sąsiada okazała się dla naszego zmodyfikowanego problemu komiwojażera zdecydowanie gorsza niż w pozostałe dwie techniki. Prawdopodobnie może się tak dziać ze względu na fakt, że funkcja celu zamodelowana w ten sposób nie oddaje prawdziwej charakterystyki naszego problemu. Wybierając zawsze najbliższego sąsiada i wstawiając go w dowolne miejsce nie kontrolujemy bezpośrednio tego w jaki sposób ulegnie modyfikacji ścieżka grafu a patrzymy jedynie przez pryzmat odległości wierzchołków od siebie.

Cykl generowany drugą metodą (metodą rozbudowy cyklu) jest więc dużo lepszy. W tym przypadku rzeczywiście modelujemy dokładniej element systemu, na którego optymalizacji zależy nam najbardziej (kryterium jest dla nas rzeczywista długość fragmentu, który powstanie na skutek włączenia wierzchołka do cyklu). Nie mniej wciąż, ze względu na zachłanną charakterystykę tego podejścia mogą się zdarzyć sytuacje, w których nie uwzględniając przyszłości uniemożliwimy sobie lepszy ruch biorąc pochopnie to co najlepsze w teraźniejszości.

Trzecia metoda (Heurystyka Konstrukcyjna 2-żal) daje nam narzędzie do walki z sytuacją opisaną w poprzednim akapicie. Jeżeli jako analitycy zdamy sobie sprawę z prawdopodobieństwa istnienia takiego zagrożenia, możemy przeciwdziałać wykorzystując mechanikę żalu, żeby spróbować osiągnąć lepszy wynik (nie blokując sobie możliwości, które mogą przynieść więcej korzyści w następnym kroku).

# Kod programu

Repozytorium z całym kodem napisanym w jupyter\_notebook znajduje się na githubie pod linkiem

<https://github.com/mikolaj-sienkiewicz/PP_Inteligente_Metody_Optymalizacji> (plik Lab1.ipynb)