|  |  |
| --- | --- |
| Sprawozdanie 2 – Lokalne przeszukiwanie | |
| **Imie Nazwisko** | **Nr indexu** |
| Patryk Jedlikowski  Mikołaj Sienkiewicz | 136723  136309 |

# Opis zadania

Zadanie polegało na zaimplementowaniu dwóch algorytmów lokalnego przeszukiwania - w wersji stromej (steepest) oraz zachłannej (greedy). Każdy algorytm należało uruchomi na dwóch różnych rodzajach sąsiedztwa startując z losowym rozwiązaniem początkowym lub rozwiązaniem uzyskanym z jednej z heurystyk opracowanych w ramach poprzedniego zadania. Jako heurystykę początkową wybraliśmy algorytm zachłanny inspirowany metodą najbliższego sąsiada. Eksperymenty wykonaliśmy na zbiorach danych kroa i krob. Budowaliśmy dwa cykle po 50 wierzchołków każdy. Każde zaimplementowane podejście było uruchamiane 100 razy w celu uśrednienia wyników i zaprezentowania ostatecznych wyników.

# Pseudokody zaimplementowanych funkcji

Wszystkie algorytmy zaimplementowaliśmy składając odpowiednio funkcje implementujące typy przeszukiwań, rodzaje ruchów wewnątrztrasowych i inicjalizację cyklów startowych. Ich pseudokody zamieszczamy poniżej:

## Steep search

Lokalne przeszukiwanie w wersji steep

Wygeneruj rozwiązanie startowe x

powtarzaj

    Wygeneruj zbiór M(x) – [external\_moves,internal\_moves]

    znajdź najlepszy ruch m należy do M(x)

    jeżeli f(m(x)) > f(x) to

        x := m(x)

dopóki nie znaleziono lepszego rozwiązania po przejrzeniu całego M(x)

gdzie M(x) to przestrzeń możliwych ruchów wewnątrz i między trasowych.

## Greedy search

Lokalne przeszukiwanie w wersji zachłannej

Wygeneruj rozwiązanie startowe x

Powtarzaj

    Wygeneruj zbiór ruchów M(x) - [external\_moves,internal\_moves]

    dla każdego m należacego do M(x) w losowej kolejności:

        jeżeli f(m(x)) > f(x) to:

            x := m(x)

            zacznij pętle od nowa

dopóki nie znaleziono lepszego rozwiązania po przejrzeniu całego M(x)

gdzie M(x) to przestrzeń możliwych ruchów wewnątrz i między trasowych

## Random Wandering

Wygeneruj rozwiązanie startowe x

Powtarzaj

    dla losowego m z M(x):

        x := y

dopóki nie upłynął czas najdłuższego wykonania

## Change Vertices

Ruch wewnątrztrasowy wymiany wierzhołków i zmieniający zbiór wierzchołków między cyklami

funkcja zmiana\_wierzchołków ([] cykle, Int wierzchołek\_A, Int wierzchołek\_B, Bool ruch\_wewnętrzny):

jeżeli ruch\_wewnętrzny:

Znajdz wierzchołek A i B w jednym cyklu i zamień miejscami

w przeciwnym wypadku:

Znajdz wierzchołek A w jednym a B w drugim cyklu i zamień miejscami

## Change Edges

Ruch wewnątrztrasowy wymiany krawędzi

funkcja zmiana\_krawędzi(cykle, początekKrawędzi1, początekKrawędzi2):

    znajdź cykl, krórego dotyczy zmiana

    znajdź indeks początków krawędzi w cuklu

    podmień krawędzie pomiędzy początkami wybranych krawędzi obracając kierunek pierwotnego cyklu

## Start\_cycle\_random

Wygeneruj cycle startowe losowo

cykl\_pierwszy = wybierz losowo 50 liczb z zakresu (0-99)

cykl\_drugi = pozostałe liczby z zakresu (0-99) nie wybrane w cykl\_pierwszy

Zwróć cykl\_pierwszy, cykl\_drugi

## Start\_cycle\_heuristic

Wygeneruj cycle startowe posługując się heurystyką z Lab1 (Nearest Neighbour)

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

distance=Oblicz\_macierz\_dystansu(dataset)

distance(przekatna)=nieskonczonosc

Wybierz losowo wierzchołek startowe dla cycleA oraz najdalszy względem niego wierzchołek dla 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

## Main – złączenie funkcji

Dla datasetu kroa i krob:

    Dla inicjalizacji startowej typu random i heuristic:

        Dla zmiany wewnątrztrasowej switch\_edges i switch\_vertices:

            Dla przeszukiwania typu steep i greedy:

                Wykonaj 100 razy:

                    Wywołaj odpowiednie funkcje dla tej iteracji

                    Zapisz wyniki -> średnie max min czas i trasy\

                    i graf prezentujący najlepszy wynik

# Tabela prezentująca wartości funkcji celu eksperymentu obliczeniowego

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Search type** | **Internal move type** | **Initialization type** | **kroa** | **krob** |
| Greedy search | Change vertices | Random | 41186 (32117-54106) | 41247 (34434-52149) |
| Heuristic | 29738 (25429-32908) | 41322 (33127-49132) |
| Change edges | Random | 27832 (25757-30745) | 28490 (25761-30429) |
| Heuristic | 26460 (22590-29250) | 28694 (26469-31221) |
| Steep search | Change vertices | Random | 42006 (31692-53839) | 43534 (36098-55849) |
| Heuristic | 29186 (25102-32582 | 41858 (32666-51554) |
| Change edges | Random | 27927 (25412-30980) | 28500 (24917-31000) |
| Heuristic | 26130 (23363-28653) | 28340 (25876-29983) |

# Tabela prezentująca czasy obliczeń eksperymentu obliczeniowego

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Search type** | **Internal move type** | **Initialization type** | **Kroa (sec)** | **Krob (sec)** |
| Greedy search | Change vertices | Random | 11.16 (6.65-20.42) | 9.12 (6.18-16.84) |
| Heuristic | 2.42 (1.23-5.80) | 11.26 (6.84 -22.00) |
| Change edges | Random | 11.11 (5.88-24.35) | 8.88 (6.68-14.61) |
| Heuristic | 4.28 (2.92-7.55) | 9.81 (6.90-17.38) |
| Steep search | Change vertices | Random | 81.34 (59.17-119.87) | 78.73 (61.86-109.38) |
| Heuristic | 13.88 (7.91-21.48) | 78.09 (55.18-109.87) |
| Change edges | Random | 78.98 (55.42-101.35) | 70.06 (55.97-81.78) |
| Heuristic | 20.63 (14.68 – 34.02) | 76.51 (54.52-107.07) |

# Wizualizacje najlepszych rozwiązań

Wizualizacje oznaczone są w sposób:

**Typ przeszukiwania – Typ ruchu wewnątrztrasowego – Typ inicjalizacji cyklu – dataset**

|  |  |
| --- | --- |
| **Greedy – Vertices – Random – Kroa** | **Greedy – Vertices – Random – Krob** |
|  |  |

|  |  |
| --- | --- |
| **Greedy – Vertices – Heuristic – Kroa** | **Greedy – Vertices – Heuristic – Krob** |
|  |  |

|  |  |
| --- | --- |
| **Greedy – Edges – Random – Kroa** | **Greedy – Edges – Random – Krob** |
|  |  |

|  |  |
| --- | --- |
| **Greedy – Edges – Heuristic – Kroa** | **Greedy – Edges – Heuristic – Krob** |
|  |  |

|  |  |
| --- | --- |
| **Steep – Vertices – Random – Kroa** | **Steep – Vertices – Random – Krob** |
|  |  |

|  |  |
| --- | --- |
| **Steep – Vertices – Heuristic – Kroa** | **Steep – Vertices – Heuristic – Krob** |
|  |  |

|  |  |
| --- | --- |
| **Steep – Edges – Random – Kroa** | **Steep – Edges – Random – Krob** |
|  |  |

|  |  |
| --- | --- |
| **Steep – Edges – Heuristic – Kroa** | **Steep – Edges – Heuristic – Krob** |
|  |  |

# 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 Lab2/Lab2.ipynb)