AKADEMIA INNOWACYJNYCH ZASTOSOWAŃ TECHNOLOGII CYFROWYCH (AI TECH)

„Uczenie maszynowe" – laboratorium

Laboratorium 2

Optymalizacja przy pomocy Algorytmów Genetycznych

data aktualizacji: 22.03.2022

Cel ćwiczenia

Celem ćwiczenia laboratoryjnego jest zapoznanie się ze wskazanym problemem optyma-lizacji oraz zastosowanie metaheurystyki **Algorytmu Genetycznego** (GA) dla rozwiązania problemu komiwojażera (TSP, *Travelling Salesman Problem*) lub optymalizacja sieci dystrybucyjnej (cVRP, *Capacitated Vehicle Routing Problem)*. Sprawdzenie jak na skuteczność GA wpływają jego parametry (krzyżowanie, mutacja, selekcja turniejowa, rozmiar populacji i liczba pokoleń). W zakresie zadania jest też porównanie wyników działania z algorytmem (heurystyka!) zachłannego oraz losowego.

Student wybiera do realizacji zadania **jeden problem**: TSP lub cVRP.

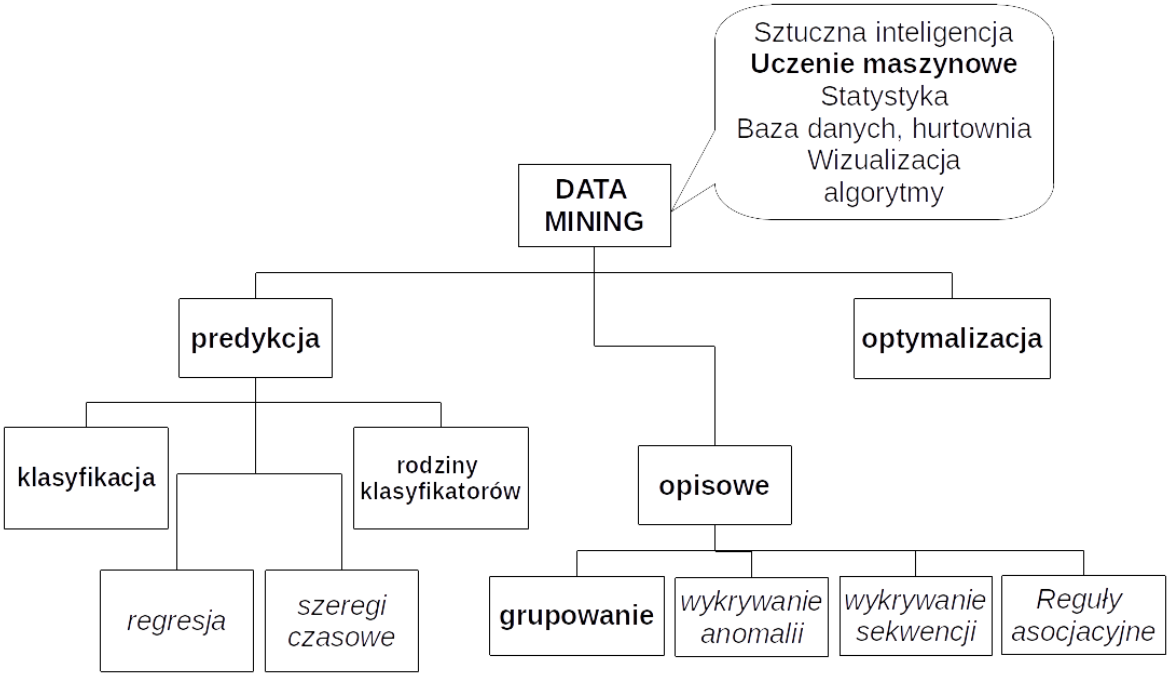
- TSP jest łatwiejszy do zdefiniowania dla rozwiązania przez GA – w treści zadania zasugerowano reprezentacje, operatory (mutacja, krzyżowanie) i funkcję oceny. Przy wyborze tego problemu maksymalna punktacja za zadanie to 10pkt.
- cVRP wymaga zbudowania szczególnego rodzaju reprezentacji (innej od TSP) lub modyfikacji funkcji oceny, tak aby rozwiązania w GA były poprawne, tj. zachowane

ograniczenie związane z ładownością pojazdu. Z tego powodu przy wyborze tego problemu punktacja maksymalna to 12pkt.

Szczegóły punktacji zadania przedstawione są w treści dokumentu w postaci tabeli.

Wprowadzenie

Uczenie maszynowe to nie tylko tworzenie modeli klasyfikacji, grupowania, predykcji czy inne (np. regresja, reguły asocjacji), ale także **modele optymalizacji.**



Są problemy (np. NP-trudne), dla których klasyczna informatyka nie jest w stanie zaproponować (algorytmu) rozwiązania w akceptowanym dla nas czasie. Z tego powodu stosowane są heurystyki[1] oraz metaheurystyki, takie jak algorytm genetyczny[2]. Zastosowanie (meta)heurystyk jest bardzo częste dla rozwiązywania problemów NP-trudnych, które mają

1 Heurystyka, tym różni się od algorytmu, że nie gwarantuje znalezienia najlepszego rozwiązania. Zwykle daje rozwiązanie lokalnie optymalne.

2 Algorytm genetyczny (też czasem zwany algorytmem ewolucyjnym, metoda z rodziny obliczeń ewolucyjnych) jest metaheurystyką. Możemy to tłumaczyć: heurystyką stosowania innych heurystyk. Mamy więc dany schemat postępowania (GA), ale poszczególne składowe zwykle są heurystykami, np. operator krzyżowania/ selekcji/ mutacji jest swojego rodzaju heurystyką.

niebagatelne znaczenie dla praktycznych zastosowań, szczególnie dla gospodarki (i przemysłu). Tego typu problemy to nie tylko klasyczny problem komiwojażera (TSP), który musi odwiedzić *n* lokalizacji. Tam „tylko" musimy tylko pilnować aby każda z lokalizacji pojawiła się raz i żadnej nie brakowało. Funkcją oceny (oceny) jest długość przejechanej trasy.

Co jeśli mamy problem centrum logistycznego, gdzie musimy dostarczyć towary wg zapotrzebowania odbiorców w różnych lokalizacjach? Taki problem występuje nie tylko w sieciach handlowych, firmach kurierskich ale także… w firmach obsługi nieczystości w miastach (akurat tutaj śmieciarki „zbierają" a nie rozwożą, ale problem jest analogiczny). Wtedy problem TSP może wymagać rozszerzeń, a pojazd ma ograniczoną pojemność niepozwalającą na obsłużenie całej trasy; dodatkowo: liczba przesyłek dla każdego adresata jest różna. Wtedy mamy do czynienia z problemem cVRP, tj. musimy odwiedzić wszystkie lokalizacje, ale dodatkowo mamy „magazyn", który musimy odwiedzić gdy pojazd nie jest w stanie obsłużyć kolejnej lokalizacji.

Problem Kuriera, MPO i *Biedronki* – zarys praktyczny

Problem komiwojażera (TSP) to problem, który mają teoretycznie wszyscy kurierzy świata – muszą odwiedzić *N* lokalizacji i rozwieźć wszystkie paczki. Oczywiście, chcą oni zminimalizować długość trasy, alby mieć szybciej wolne. W czym problem? Jeśli mamy ponad 100 pakunków (lokalizacji) przestrzeń rozwiązań jest zbyt duża aby rozwiązać problem „algorytmicznie".

Innym, podobnym problemem do TSP jest problem cVRP, który oprócz ograniczenia związanego z potrzebą odwiedzenia wszystkich lokalizacji, musi brać pod uwagę zdefiniowane ograniczenie: pojemność pojazdu i różne wymagania „pakunkowe" w lokalizacjach. Z takim problemem w praktyce mamy do czynienia, gdy obsługujemy sieć sklepów (każdy z nich zgłasza różne zapotrzebowanie na towar) lub przy obsłudze trasy śmieciarek MPO (każdy z adresów/ulic generuje różne ilości śmieci). Funkcją celu, którą chcemy minimalizować, pozostaje długość trasy.

Opis problemu – TSP

Mając *N* miast (graf pełny nieskierowany, ważony odległością) chcemy wyznaczyć najkrótszą trasę (suma wag), która zwiera wszystkie miasta.

Przy rozwiązywaniu tego problemu można użyć pliki z problemu cVRP (poniżej), jednak bez uwzględniania czynnika „pojemności” pojazdu.

Opis problemu – cVRP

Tak jak w problemie TSP, obsługujemy *N* miast (lokalizacji), jednak mamy jedną specjalną lokalizację (magazyn), który musimy uwzględnić przy sprawdzaniu ograniczeń związanych z pojemnością pojazdu.

Formalny opis problemu i formatu plików wejścia/wyjścia znajduje się pod linkiem:
http://dimacs.rutgers.edu/programs/challenge/vrp/cvrp/

Plik wejściowy do testów:

```
NAME : toy.vrp
COMMENT : toy instance>
TYPE : CVRP
DIMENSION : 6
EDGE_WEIGHT_TYPE : EUC_2D
CAPACITY : 30
NODE_COORD_SECTION
1 38 46
2 59 46
3 96 42
4 47 61
5 26 15
6 66 6
DEMAND_SECTION
1 0          // magazyn, zapotrzebowanie „0”
2 16
3 18
4 1
5 13
6 8
DEPOT_SECTION  // tę sekcję możemy zignorować
1
-1
EOF
```

W powyższym pliku (formacie) mamy do czynienia z 5 lokalizacjami, oraz jednym magazynem. Są podane współrzędne lokalizacji (np. współrzędnie magazynu to [38,46]), a odległości pomiędzy nimi liczone za pomocą euklidesowej miary (EUC_2D). Każda z lokalizacji posiada zapotrzebowanie na towar (w DEMAND_SECTION, np. lokalizacja 2 wymaga 16 jednostek, a magazyn 0). Pojazd używany w tym pliku ma pojemność 30 (CAPACITY podane w nagłówku).

Powyższy plik warto użyć do „ręcznego" rozpoznania problemu (na kartce), ale też przy sprawdzaniu poprawności działania kodu funkcji oceny czy operatorów genetycznych. Optymalne rozwiązanie dla powyższego pliku testowego:

```
Route #1: 2 5
Route #2: 4 3 6
Cost 265
```

Algorytm Genetyczny – opis działania

Algorytm Genetyczny jest metaheurystyką pracującą na populacji rozwiązań i wykonywaną w oparciu o następujący schemat (w postaci pseudokodu przedstawiono na Pseudokod 1). Pierwszą operacją jest inicjalizacja, która ustala osobniki początkowe dla danego przebiegu GA – zwykle zaczynamy od losowych osobników. Dla ułatwienia prac zakładamy, że liczba osobników (parametr *pop_size*) w populacji przez cały czas działania GA jest stała.

W dalszej kolejności osobniki są oceniane (dla TSP jest to długość trasy). W następnym kroku sprawdzany jest warunek stopu (zwykle jest to limit „iteracji", tj. pokoleń *gener[ations]*).

Zasadniczy mechanizm GA to wybór osobników rodzicielskich (**selekcja** – tutaj używamy metody turnieju), a w dalszej kolejności sprawdzamy czy powinno zajść **krzyżowanie** (o tym decyduje parametr prawdopodobieństwo krzyżowania – *Px*). Dalej, może zadziałać operator **mutacji** (parametr prawdopodobieństwo mutacji *Pm* steruje częstością jej zachodzenia). Nowo utworzony osobnik jest **oceniany** (funkcja oceny, funkcja przystosowania) i trafia do nowej populacji. Cały cykl jest powtarzany, aż do osiągnięcia rozmiaru populacji (*pop_size*) i finalnie do przekroczenia liczby pokoleń.

```
begin
t := 0;
inicjalizacja( pop(t) );
ocena( pop(t) );
        while (not warunek_stopu) do
        begin
          while ( pop(t+1).size()!=pop_size )
               P1 := turniej( pop(t) );
               P2 := turniej( pop(t) );
               if ( rand[0,0..1,0] < Px)
                  Op:= krzyzowanie( P1, P2 );
               else Op:=P1;
               Op := mutacja( Op, Pm );
               ocena( Op );
               pop(t+1).dodaj( Op );
               if ( O_najlepszy > Op )  O_najlepszy := Op
          end
        t++;
        end
return O_najlepszy
end
```

Pseudokod 1. Pseudokod Algorytmu Genetycznego

Operatory genetyczne użyte w zadaniu:

- selekcja turniejowa – spośród populacji rodzicielskiej losowanych jest N osobników (z- lub bez- zwracania) i z pośród nich wybierany jest najlepszy. Selekcja ma parametr (*Tour*) związany z rozmiarem puli osobników biorących udział w selekcji.
- krzyżowanie (np. CX, OX lub PMX) – operatory, które działają na dwóch osobnikach rodzicielskich i w wyniku dają nowy osobnik (lub dwa).

Każdy z powyższych operatorów krzyżowania opiera się na innej heurystyce działania – staramy się zachować kolejność lokalizacji rodzicielskich, uwzględnić „cykl” lub wymianę lokalizacji.

Krzyżowanie OX (Ordered Crossover) wyznacza dwie pozycje w rozwiązaniu i tworząc rozwiązanie potomne bierze podciąg od jednego rodzica, starając się uzupełnić kolejnością z drugiego rodzica:

Przykład działania operatora OX:

P1 |123456789| x P2 |574913628| => O: |793456128|

Działanie operatorów opisano w pracach [3][5][6] (też można skorzystać z internetowych źródeł).

- mutacji (np. inwersja) – działa tylko w jednym osobniku i wprowadza tam zmianę. Przykładowy operator mutacji to inwersja, która „odwraca” kolejność lokalizacji we wskazanych pozycjach.

  Przykład działania inwersji: |5671234| → |5321764|

Trudność cVRP a TSP

Przy rozwiązywaniu problemu TSP rozwiązaniem może być dowolna sekwencja lokalizacji (np. 14325) i jeśli każda z lokalizacji jest odwiedzana tylko raz, uznajemy osobnika za poprawny.

W przypadku cVRP nie każda sekwencja miast jest poprawna, bo jeśli dla powyższej przykładowej instancji cVRP sekwencja 14325 nie może być obsłużona jednym transportem (złamano ograniczenie, tj. przekroczono ładowność pojazdu) – pojazd nie jest w stanie obsłużyć trzech pierwszych lokalizacji – po drugiej musi „zjechać” do magazynu.

Jest kilka sposobów radzenia sobie z cVRP, w przypadku realizacji zadania można wybrać jedno z dwóch poniższych:

- użycie algorytmu **zachłannego**, który „wstawi dynamicznie” w osobnika jak w TSP przy ocenie trasy magazyny gdzie one są niezbędne, np. jak powyżej rozwiązanie 14325 zmieni się w 14m325, doliczając do długości trasy powrót z 4 to magazynu i trasę z magazynu do 3.

- **rozbudowana reprezentacja**, która zawiera nie tylko lokalizacje, ale też magazyn. Dla N lokalizacji „dodajemy” też N magazynów (np. mogą to być wartości ujemne, lub jedna określona wartość). W ten sposób operatory genetyczne mutacji i krzyżowania działają bez większych zmian. Trzeba jednak sprawdzać sytuację, gdzie osobnik nie wstawi niezbędnego powrotu do magazynu (złamie ograniczenie) i gdy mamy obok siebie dwa symbole magazynowe (to proste: trasa z magazynu do magazynu to 0). Uwaga! Takie rozwiązanie jest nieco nadmiarowe (2x większy osobnik), jednak nie korzysta z algorytmu zachłannego ograniczając często rozwiązania do lokalnych minimów.

Sugerowane narzędzia

Zadanie może być realizowane przy pomocy języka python i bibliotek użytecznych przy manipulacji danymi oraz wizualizacji wyników (tabele/wykresy).

Wskazane narzędzia (python): jupyter, numpy, mathplotlib, seaborn itp.

Przy implementacji w python można użyć gotowej implementacji Algorytmu Genetycznego PyPi (https://pypi.org/project/geneticalgorithm/)

Uwaga! Python jest językiem interpretowanym i w porównaniu do innych (np. C++, nawet Java, C#) czas pracy programu zwykle jest dłuższy. Stąd sugeruje się użycie innych języków programowania. Sugerowane narzędzia (C++ / Java): debugger, profiler

Wstępne parametry przydane do strojenia GA:

pop_size=100, gen=100, Px=0,7, Pm=0,1, Tour=5

Przebieg ćwiczenia

1. Analiza problemu optymalizacyjnego. Czym jest: osobnik, czym jest rozwiązanie, czym jest funkcja oceny? Kiedy rozwiązanie jest niepoprawne? Jakie rozwiązanie ma cechy i jak można je zmieniać?

2. Implementacja *loadera* danych – dla przyśpieszenia badań warto zbudować raz macierz odległości, a potem tylko odczytywać pomiędzy lokalizacjami (tj. odległość[i][j]). Implementacja funkcji oceny.

3. Implementacja algorytmu zachłannego – na start ma kolejkę (nieodwiedzonych) lokalizacji i kolejno bierze pod uwagę tylko najbliższą lokalizację do aktualnego. Budując poprawne rozwiązanie dla cVRP sprawdza tylko czy ma jeszcze zapas do obsługi lokalizacji. Jeśli nie, dorzuca „zjazd" do magazynu.

4. Implementacja metody generowania kolejki *N* dla algorytmu zachłannego, tj. każda z lokalizacji zostaje (w jakiś sposób) umieszczona w kolejce.

5. Testowanie poprawności kodu (rozwiązanie / funkcja oceny) na postawie przypadku testowego.

6. Implementacja Algorytmu Genetycznego, tj.:

   - reprezentacji TSP (cVRP),
   - operatorów genetycznych (krzyżowanie, mutacja),
   - selekcji turniejowej,
   - oraz zarządzania populacją.

7. Testowanie poprawności kodu GA (reprezentacja, operatory, selekcja) na postawie przypadku testowego

8. Wstępne strojenie Algorytmu Genetycznego (na wybranym pliku)

Przy strojeniu/badaniu GA warto użyć **wykresu**, który pokazuje jak poprawia się jakość osobników w poszczególnych pokoleniach. W tym celu warto zrobić zestawienie dla każdego pokolenia (best, worst, avg) – najlepszego osobnika, najgorszego osobnika i średnie przystosowanie osobnika w populacji. Taki wykres złożony z trzech (best, worst, avg) x (pokolenie) pozwala na analizę postępu „uczenia" się GA.

9. Zbadanie jak na skuteczność GA wpływa prawdopodobieństwo mutacji (*Pm*) i krzyżowania (*Pk*)

Badania skuteczności GA na cVRP/TSP mają opierać się na 3 różnych plikach: A-n32-k5, A-n46-k7, A-n61-k9. Link do plików: http://vrp.galgos.inf.puc-rio.br/index.php/en/ Plik podany w treści zdania jest tylko do sprawdzenia poprawności działania implementacji GA. Osoby pracujące na problemie TSP ignorują w powyższych plikach część magazynową.

10. Zbadanie jak na skuteczność GA wpływa rozmiar populacji (*pop_size*) i liczba pokoleń (*gen*)

11. Zbadanie jak na skuteczność GA wpływa turniej (i rozmiar turnieju).

12. Porównanie skuteczności dla 3 plików

Punktacja

Przy realizacji zadania student może otrzymać **max 10 lub 12 punktów** wedle poniższej tabeli.

2	Implementacja wczytanie problemu, heurystyki i algorytmu losowego, funkcji oceny
2	Implementacja Algorytmu Genetycznego dla problemu cVRP: osobnik, operator krzyżowania, mutacja, selekcja turniejowa
2	Zbadanie wpływu operatorów (mutacji, krzyżowania) na uzyskiwane wyniki
2	Zbadanie wpływu parametrów populacji (rozmiar populacji, liczba generacji) na skuteczność Algorytmu Genetycznego. Zbadanie jak na skuteczność Algorytmu Genetycznego wpływa rozmiar turnieju
2	Porównanie wyniku działania AG do algorytmu zachłannego oraz losowego. Należy uśrednić wyniki z losowego (100 razy), algorytmu zachłannego (ile jest lokalizacji, start z każdej), a Algorytmu Genetycznego (10 razy).
*2	Zastosowanie GA do problemu cVRP – cała implementacja oraz badania (wg powyższych punktów) dla problemu cVRP

Przy realizacji tego zadania wystarczy prosty raport PDF utworzony przy użyciu Jupyter. Przy wynikach badań należy dać komentarz, podać wnioski i podsumowanie.

Warto użyć wykresów, szczególnie polecam wykres jak przystosowanie (średnie, najlepsze, najgorsze) zmienia się w kolejnych pokoleniach.

Dla chętnych	Osoby, które nie chcą operować na problemie TSP lub cVRP, mogą użyć problemu z konkursu PACE 2021 Cluster Editing (https://pacechallenge.org/2021/) – Heuristic. Trudnością jest definicja problemu (klasteryzacja) oraz potrzeba zdefiniowania heurystyki, operatorów. Założenia, w ramach zajęć działamy: 1. jako metoda heuristic https://pacechallenge.org/2021/tracks/#heuristic-track 2. tylko na 5 grafach max 250 węzłów (https://pacechallenge.org/2021/tracks/#benchmark-for-heuristic-track)

Pytania pomocnicze

1. Co się stanie, gdy jako rozmiar turnieju przyjmiemy 1 osobnika?
2. Co, gdy rozmiar turnieju jest równy liczności populacji?
3. Co jest ważniejsze: krzyżowanie czy mutacja?
4. Czy może być za dużo/mało mutacji?
5. Czy może być za dużo/mało krzyżowania?
6. Czy może być za dużo/mało osobników w populacji?
7. Jak liczba pokoleń wpływa na skuteczność GA?

Literatura

1. Materiały do wykładu
2. Arabas J. Wykłady z algorytmów ewolucyjnych (http://staff.elka.pw.edu.pl/~jarabas/ksiazka.html)

3. Goldberg D. Algorytmy genetyczne i ich zastosowanie

4. Kwaśnicka H. Obliczenia ewolucyjne w sztucznej inteligencji, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, 1999.

5. Michalewicz Z. Algorytmy genetyczne + struktury danych = programy ewolucyjne, WNT.

6. Michalewicz Z., Fogel D.B. Jak to rozwiązać, czyli nowoczesna heurystyka, WNT 2006 (polecam lekturę całej książki; dla minimalistów EA - rozdział 7 i 10)

7. Zasoby Internetu, słowa kluczowe: genetic algorithm, evolutionary algorithm, optmisation, np-hard problem, *capacitated vehicle routing problem,* cvrp, travelling salesman problem, tsp, inversion, crossover, tournament selection, greedy algorithm,

Obliczenia w chmurze?

Algorytmy Genetyczne są dość wymagające obliczeniowo. Należy o tym pamiętać już w momencie wyboru narzędzi programistycznych i później, w czasie tworzenia kodu:

Warto zrobić:

1. aplikację bez klikania =linia komend, która pozwala ograniczyć „klikanie" w aplikacji
2. i automat do testów, który bardzo ułatwi realizację zadania
3. i zoptymalizować kod (użycie **profilera**)
4. oraz ew. użyć **obliczeń w chmurze**.

Garść linków, które dają dostęp do darmowych obliczeń w chmurze:

- https://aws.amazon.com/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc
- https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/free-tier-limits.html
- https://cloud.google.com/free/
- https://azure.microsoft.com/ np. z możliwością wystawienia aplikacji konsolowej https://azure.microsoft.com/ z zapisem do Azure Blob Storage https://azure.microsoft.com/