Sztuczna inteligencja i inżynieria wiedzy (laboratorium)

Ćwiczenie 1. Algorytmy genetyczne

opracowanie: P.Myszkowski. M.Laszczyk

Wrocław, 28.02.2019

Cel ćwiczenia

Zapoznanie się z metaheurystyką (algorytmy genetyczne) w praktyczny sposób poprzez samodzielną implementację.

Realizacja ćwiczenia

- Zapoznanie się z metaheurystyką (algorytmy genetyczne)
- Określenie problemu optymalizacyjnego do rozwiązania – minimalizacja w problemie Mobilnego Złodzieja TTP (ang. *Travelling Thief Problem*)
- Zbudowanie algorytmu genetycznego: osobnik, funkcja oceny, krzyżowanie, mutacja, selekcja i inicjalizacja
- Implementacja modelu w dowolnym języku obiektowym (sugerowana Java, C/C++/C#, ew. python)
- Zbadanie wpływu różnych parametrów (prawd. mutacji *Pm*, krzyżowania *Px*, selekcji, rozmiar populacji *pop_size*, liczba pokoleń *gen*) na efektywność i skuteczność metody
- Sporządzenie sprawozdania z ćwiczenia
- Pokazanie na wykresach zmianę wartości przystosowania (wartość optymalizowanej funkcji celu) w poszczególnych pokoleniach: najlepszy osobnik, średnia wartość w populacji i najgorszy osobnik
- Porównaj działanie algorytmu genetycznego z wybranymi przez siebie, nieewolucyjnymi metodami optymalizacji (np.: metoda losowego przeszukiwania, algorytm zachłanny). Proszę zwrócić uwagę na uzyskany wynik, czas działania oraz liczbę wartościowań optymalizowanej funkcji celu. Proszę pokazać i omówić najciekawsze wyniki
- Raport z ćwiczenia powinien zawierać wszystkie punkty wymagane w realizacji zadania

Problem Mobilnego Złodzieja (*Travelling Thief Problem,* TTP)

W problemie TTP, roboczo dla potrzeb zajęć nazywany „Problemem Mobilnego Złodzieja" złączone są dwa ważne problemy NP-trudne – Problem komiwojażera (ang. *Travelling Salesman Problem, TSP)* oraz problem plecakowy (ang. *Knapsack Problem, KNP)*. W skrócie problem polega na tym, że złodziej odwiedza *n* lokalizacji i z nich *wybiera* przedmioty. W złodziejskim fachu ważne jest aby najmniej się nachodzić i *zdobyć* najcenniejsze przedmioty. Mamy jednak ograniczenia, główne: plecak jednak ma ograniczoną pojemność, a przenoszenie przedmiotów to ciężki kawałek chleba.

W problemie TTP wprowadzono dodatkowe parametry, które uzależniają od siebie jego poszczególne elementy. To sprawia, że znalezienie ich optimów niezależnie nie jest jednoznaczne ze znalezieniem optimum dla całego problemu.

Bardziej formalnie: problem dzielimy na problemy TSP i KNP.

Sub-Problem 1 – Problem Komiwojażera (TSP)

Problem Komiwojażera składa się z *n* miast oraz macierzy odległości pomiędzy tymi miastami (w naszym przypadku macierz jest symetryczna, co oznacza, że odległość z miasta *i* do miasta j jest taka sama jak odległość z miasta *j* do miasta *i*). Celem jest znalezienie najkrótszej trasy, która odwiedzi wszystkie miasta dokładnie raz:

$$f(x)=\sum_{i=1}^{n-1}\left(t_{x_i,x_{i+1}}\right)+t_{x_n,x_1}, x=(x_1,\ldots,x_n),$$

gdzie *x* reprezentuje trasę, a *t* jest czasem pomiędzy dwoma kolejno odwiedzonymi miastami:

$$t_{x_i,x_{i+1}}=\frac{d\,x_i,x_{i+1}}{v_c},$$

gdzie v_c jest prędkością, a *d* jest dystansem pomiędzy dwoma miastami.

Sub-Problem 2 – Problem Plecakowy (KNP)

Problem Plecakowy składa się ze zbioru *m* przedmiotów oraz plecaka. Każdy przedmiot opisany jest wagą (w_i) oraz wartością (p_i), natomiast plecak ma swoją ograniczoną pojemność (maksymalną dopuszczalną wagę). Problem polega na wybraniu takich przedmiotów do plecaka, żeby zmaksymalizować jego wartość nie przekraczając przy tym jego pojemności wedle funkcji:

$$g(y)=\sum_{i=1}^{m} p_i\,y_i,(y_1,\ldots,y_m),$$

gdzie *p*, to wartość przedmiotu (profit), a *y* określa czy dany przedmiot został wybrany czy nie (1 lub 0), *w* to waga przedmiotu. Suma wag wybranych przedmiotów (W_c) nie może przekroczyć pojemności plecaka (W). Formalnie:

$$W_c=\sum_{i=1}^{m} w_i\,y_i\leq W,$$

Stanowi to główne ograniczenie problemu TTP.

Problem Mobilnego Złodzieja TTP

Problem TTP opiera się zatem na liście *n* miast oraz odległości między nimi. Dodatkowo istnieje *m* przedmiotów opisanych wagą (w_i) oraz wartością (p_i). Złodziej może odwiedzić każde miasto (lokalizację) jeden raz i z każdego zabrać pewne przedmioty. Uwaga! nie każdy przedmiot jest dostępny w każdym mieście - dostępność przedmiotów jest częścią definicji konkretnej instancji problemu. **Rozwiązanie TTP** składa się z trasy oraz planu wyboru przedmiotów. Warto zaznaczyć, że przedmiot można podnieść tylko przy wejściu do miasta. Oznacza to, że złodziej na końcu swojej podróży wraca do miasta, z którego wyruszył, natomiast nie może już podnieść w nim przedmiotów.

Funkcja oceny maksymalizuje całkowity zysk złodzieja - sumę wartości przedmiotów. Jednakże wprowadzone są dodatkowe parametry, które tworzą zależność pomiędzy komponentami problemu.

Prędkość podróży jest zmienna i jest zależna od zapełnienia plecaka w myśl zasady, że trudniej wędrować z cięższym plecakiem. Formalniej:

$$v_c = v_{max} - W_c \frac{v_{max} - v_{min}}{W},$$

gdzie W_c jest aktualną sumą wag przedmiotów w plecaku, a W jest maksymalną dopuszczalną wagą, v_c jest aktualną prędkością. Bo złodziej podróżuje wolniej, im więcej przedmiotów ma w plecaku. Celem problemu TTP jest maksymalizacja:

$$G(x, y) = g(y) - f(x, y),$$

gdzie *x* to trasa, *y* to plan wyboru przedmiotów, *g(y)* to suma wartości wybranych przedmiotów w plecaku, a *f(x, y)* to całkowity czas podróży z trasy *x* z uwzględnieniem przedmiotów z *y*.

Uwaga: Literatura podaje dodatkowy parametr *R*, który oznacza koszt „wynajmu" plecaka. W zbiorze danych ten parametr nie występuje i w tym laboratorium powinien zostać pominięty.

Algorytm genetyczny

Algorytm genetyczny jest metaheurystyką (choć nazwa sugeruje – nie jest to algorytm), która naśladuje ewolucję naturalną metodą ciśnienia selekcyjnego i doboru naturalnego. Aby ją zastosować, należy zdefiniować potencjalne rozwiązanie (osobnika), sposoby jego zmiany (mutacja), łączenia (krzyżowania) oraz oceny jakości rozwiązania (funkcja oceny).

Algorytm genetyczny opiera się na schemacie przedstawionym jako Pseudokod 1.

```
begin
t:=0;
initialise( pop(t0) );
evaluate( pop(t0) );
        while (not stop_condition) do
        begin
          pop(t+1) := selection( pop(t) );
          pop(t+1) := crossover( pop(t+1) );
          pop(t+1) := mutation( pop(t+1) );
          evaluate( pop(t+1) );
          t:=t+1;
        end
return the_best_solution
end
```

Pseudokod 1. Pseudokod Algorytmu Genetycznego

GA wstępnie inicjalizuje (zwykle losowo) populację rozwiązań. Następnie ocenia jakość poszczególnych osobników. W kolejnym kroku sprawdzane są warunki zatrzymania: czy osiągnięto akceptowalne rozwiązanie i/lub limit pokoleń został przekroczony. Wybór osobników (sugerowana metoda turnieju lub ruletki) do następnej populacji następuje przed działaniem operatora krzyżowania i mutacji, które budują osobniki nowego pokolenia. Dalej, następuje ocena nowych rozwiązań i cykl GA się zamyka przy sprawdzaniu warunków zatrzymania.

Osobnik dla problemu TTP mógłby być **reprezentowany** jako sekwencja miast, np. 35124. Wystartujemy z miasta nr 3, potem odwiedzimy miasto 5, potem 1 itd. Na koniec z miasta 4 wracamy do miasta 3 (początkowe). W takiej reprezentacji każde z miast musi wystąpić tylko raz.

Operator **mutacji** dla powyższej reprezentacji wektorowej sprowadzać się może do zamiany losowo dwóch lokalizacji (tzw *swap)*. Przykład działania mutacji: 32154 → 35124
Można także rozpatrzyć mutację typu inwersja – wyznaczamy losowo dwa miejsca w osobniku w ramach tego obszaru „odwracamy" kolejność elementów.

Operator **krzyżowania** łączy dwa osobniki tworząc potomny (lub dwa, zależnie od operatora). W najprostszej postaci krzyżowanie znajduje losowy punkt przecięcia i dla potomka pierwszą część bierze od jednego rodzica, drugą część od drugiego. Uwaga! Po tej operacji należy sprawdzić czy nie brakuje jakieś lokalizacji i/lub która występuje podwójnie w osobniku potomnym. Wymagana jest procedura „naprawy" osobnika.
Można też rozważyć specjalizowane operatory krzyżowania znane z problemu TSP, takie jak: OX, CX czy PMX. Google bardzo chętnie podpowie na czym one polegają...

Dla każdego osobnika, który jest rozwiązaniem problemu TSP, trzeba też rozwiązać problem KNP. Ten problem także możemy rozwiązać skutecznie algorytmem genetycznym, jednak dla uproszczenia zadania proponuje się aby ten problem rozwiązać metodą z **algorytmu zachłannego** (ang. *greedy*). Algorytm zachłanny polega na zdefiniowaniu reguły, która może być stosowana na każdym kroku algorytmu, np. (1) zawsze bierz najdroższy dostępny przedmiot, lub (2) zawsze bierz najlżejszy przedmiot lub (3) bierz przedmiot, który ma najlepszy stosunek wartość/waga. W ten sposób dla każdej trasy TSP algorytm zachłanny proponuje przedmioty. Trzeba jednak pamiętać o warunku ograniczonego rozmiaru plecaka.

*** dla chętnych** istnieje możliwość zmiany problemu na problem harmonogramowania (alokacji zasobów w czasie) ograniczeniami w problemie MS-RCPSP. Więcej informacji o problemie MS-RCPSP i opis wybranych metod rozwiązywania dostępne są pod adresem projektu iMOPSE: http://imopse.ii.pwr.edu.pl/

Ocena realizacji ćwiczenia

1pkt	Zbudowanie modelu algorytmu genetycznego
1pkt	Implementacja algorytmu genetycznego
2pkt	Zbadanie działania na 5 plikach testowych (pliki hadXX)
2pkt	Zbadanie wpływu prawd. krzyżowania *Px* i mutacji *Pm* na wyniki działania GA
2pkt	Zbadanie wpływu rozmiaru populacji *pop_size* i liczby pokoleń *gen* na wyniki działania GA
1pkt	Zbadanie wpływu selekcji na skuteczność GA – turniej i ruletka
1pkt	Porównanie skuteczności GA z wynikami dwóch innych metod nieewolucyjnych

Uwaga! Przy testach należy brać pod uwagę przynajmniej 10 uruchomień i uśrednianie wyniku (podajemy wartość średniej i odchylenie standardowe).

Podpowiedzi

Wstępne parametry przydane do strojenia GA: *pop_size=100*, *gen=100*, *Px=0,7*, *Pm=0.01*, *Tour=5*

Mając logowanie do pliku **.csv* wartości statystyczne dla każdego pokolenia w formacie:

```
<nr_pokolenia, najlepsza_ocena, średnia_ocen, najgorsza_ocena>
```

można „za darmo" uzyskać wykresy od Excela. Wtedy można skupić się na merytoryce zadania, mniej na interfejsie, GUI, interaktywnych wykresach itp.

Uwaga techniczna odnośnie kodu realizującego zadanie. Prosi się o użycie narzędzia PROFILER i wskazanie „najdroższej” metody w kodzie. Jeśli to możliwe, warto rozważyć optymalizację kodu.

Pytania pomocnicze

1. Jak selekcja (rozmiar turnieju *Tour*) wpływa na działanie GA? Co się dzieje jeśli mamy *Tour=0* a co się dzieje jak *Tour=pop_size*?
2. Czy mutacji może być za mało/dużo?
3. Czy krzyżowanie może być za mało/dużo?
4. Co się dzieje jeśli wyłączymy krzyżowanie i/lub mutację?
5. Jak rozmiar populacji i liczba osobników wpływa na efektywność/skuteczność GA? Przy badaniu metody proszę brać pod uwagę „liczbę urodzeń”, tj. liczbę osobników odwiedzonych przez GA. W praktyce można to sprowadzić się to do określenia wartości *pop_size*gen*

Realizacja zadaniowa

Zajęcia 1.
Zapoznanie się z problemem.
Implementacja loadera.
Implementacja losowego osobnika TSP.
Implementacja funkcji f(x)

Zajęcia 2.
Implementacja algorytmu zachłannego do KNP.
Implementacja funkcji *g* i *G*
Implementacja operatorów mutacji i krzyżowania.
Implementacja selekcji i zarządzania populacją w algorytmie genetycznym.
Implementacja narzędzi logowania → wykresy.

Zajęcia 3.
Strojenie algorytmu genetycznego.
Badanie wpływu wartości parametrów na skuteczność/efektywność algorytmu genetycznego.
Przygotowanie sprawozdania: tabelki, wykresy, wnioski.

Literatura

1. Materiały do ćwiczenia: dokument na Board'zie w katalog /Kwasnicka/Sztuczna_Inteligencja/ Calosc_ZeszytNaukowyNr1_Ostatni.pdf
2. Mitchell M., An introduction to Genetic Algorithms:
3. http://www.boente.eti.br/fuzzy/ebook-fuzzy-mitchell.pdf
4. Arabas J. ‘Wykłady z algorytmów ewolucyjnych” http://staff.elka.pw.edu.pl/~jarabas/ksiazka.html
5. Goldberg D. „Algorytmy genetyczne i ich zastosowanie”
6. Michalewicz Z. „Algorytmy genetyczne + struktury danych = programy ewolucyjne”
7. Michalewicz Z., Fogel „D.B. Jak to rozwiązać, czyli nowoczesna heurystyka” Rozdziały 1-6 anglojęzycznej wersji: https://paginas.fe.up.pt/~mac/ensino/docs/OR/HowToSolveIt/Chapters_1-6_How_to_Solve_It_Modern_Heuristics.pdf
8. Strona projektu iMOPSE MS-RCPSP: http://imopse.ii.pwr.edu.pl
9. Bonyadi, Mohammad Reza, Zbigniew Michalewicz, and Luigi Barone. "The travelling thief problem: The first step in the transition from theoretical problems to realistic problems." *2013 IEEE Congress on Evolutionary Computation*. IEEE, 2013.

Opracowano: 26.02.2019