Sztuczna inteligencja i inżynieria wiedzy laboratorium

Ćwiczenie 1. Algorytmy genetyczne

opracowanie: P.Myszkowski. M.Laszczyk

Wrocław, 12.02.2018

Cel ćwiczenia

Zapoznanie się z metaheurystyką algorytmów genetycznych w praktyczny sposób poprzez samodzielną implementację.

Realizacja ćwiczenia

- Zapoznanie się z metaheurystyką algorytmów genetycznych
- Określenie problemu optymalizacyjnego do rozwiązania – minimalizacja W kwadratowy problemie przydziału QAP (*Quadratic Assignment 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 kwadratowego przydziału zasobów QAP

Problem QAP jest jednym z ważniejszych problemów alokacji zasobów spotykanymi w przemyśle i obecnym w literaturze naukowej. Zadanie sprowadza się do wskazania lokalizacji dla *N* fabryk, tak aby zminimalizować koszt transportu (*flow*) pomiędzy nimi. W tym praktyce problem ten jest rozszerzeniem klasycznego problemu komiwojażera (*TSP, Travelsman Problem*) i też problemem NP-trudnym.

W praktyce rozwiązanie możemy sprowadzić do znalezienia takiego przyporządkowania lokalizacji dla 5 fabryk *N: A,B,C,D,E*, dla lokalizacji *1,…,N.* Sugerowane jest użycie formatu rozwiązania jako przykładowego wektora:

A	B	C	D	E
3	2	1	5	4

Powyższy wektor podaje rozwiązanie, w którym dla fabryki *A* podano lokalizację 3 a dla fabryki *B* lokalizację 2 itp. Znając macierz odległości pomiędzy lokalizacjami i macierz wymaganego przepływu pomiędzy fabrykami *A* i *B* obliczamy koszt transportu. Obliczając analogicznie dla wszystkich par fabryk mamy sumaryczny koszt transportu dla danego rozwiązania. Docelowo ten sumaryczny koszt chcemy zminimalizować (funkcja oceny) i rozwiązanie z mniejszą wartością jest „lepsze".

Bardzo dobry formalny opis problemu QAP wraz z prostymi „interaktywnymi" przykładami są tutaj: https://neos-guide.org/content/quadratic-assignment-problem

Sugerowane 5 instancji do badań (hadXX): http://anjos.mgi.polymtl.ca/qaplib/inst.html#HRW
(format pliku jest tekstowy w formacie: <N><Macierz_Przepływu><Macierz_odległości>)

Algorytm gentyczny

Algorytm genetyczny jest metaheurystyką (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.

Operator mutacji dla reprezentacji wektorowej proponowanej powyżej 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). 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 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, ?rednia_ocen, najgorsza_ocena>
```

można „za darmo” uzyskać wykresy od Excela. Wtedy mozna 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*

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

Opracowano: 15.02.2018