[PSZT-P] Dokumentacja do zadania

Weronika Paszko, Sebastian Pietras

Treść zadania

MM.AG3 - Podział kart na 2 stosy

Masz N kart z wartościami od 1 do N. Przy pomocy algorytmu genetycznego należy podzielić je na dwa stosy, gdzie suma wartości kart na pierwszym stosie ma wartość jak najbliższą do A, a iloczyn wartości kart na drugim stosie jak najbliższa wartości B. WE: liczba kart N, suma kart A, iloczyn kart B, satysfakcjonujący poziom dopasowania w %. WY: podział kart na stosy z wynikami działań.

Założenia

Program został napisany w języku Python. Stos z kartami, które zostaną posumowane, nazywamy A, z kolei stos z kartami do przemnożenia nazywamy B. W celu efektywnego reprezentowania genotypów zdecydowaliśmy się na tablicę wartości logicznych. Jeśli wartość na indeksie i jest równa True, to karta o numerze i+1 znajduje się na stosie A, w przypadku False karta jest na stosie B.
Przyjęliśmy, że dla najlepszych osobników wartość funkcji przystosowania zbliża się do 0. Im dalej od 0, tym gorzej przystosowany osobnik.

Podział zadań

Weronika Paszko:

- Inicjalizacja pierwszego pokolenia
- Ocenianie osobników
- Wybieranie nowej populacji
- Testowanie

Sebastian Pietras:

- Stworzenie szkieletu programu
- Krzyżowanie osobników
- Mutacja
- Zbieranie statystyk
- Warunek stopu dla algorytmu

Sposób realizacji

W celu przeprowadzenia badań zdecydowaliśmy się na implementację kilku metod do najważniejszych elementów algorytmu genetycznego. Dzięki temu możemy zbadać, które z metod są najefektywniejsze i pozwolą najsprawniej znaleźć rozwiązanie problemu.

Ze względu na zwięzłość dokumentacji omawiamy tylko jeden z wariantów konfiguracji. Pozostałe metody są opisane komentarzami w kodzie.

Wybrane metody:

1. Pokolenie pierwsze – wektor losowych wartości logicznych.
2. Krzyżowanie – nad każdym genem rodzica wykonujemy rzut monetą, aby zdecydować, do którego z dzieci przekażemy dany gen. Drugie dziecko dostaje odpowiadający gen z drugiego rodzica.
3. Mutacja – polega na zaprzeczeniu wartości genu. Przechodzimy po całym genotypie osobnika. Jeżeli wypadnie reszka (prawdopodobieństwo wypadnięcia reszki jest podane jako parametr) to gen jest negowany, czyli karta przechodzi na inny stos.
4. Ocena przystosowania – fitness stosu A to logarytm dziesiętny z odległości sumy na stosie od oczekiwanej wartości. Fitness stosu B to logarytm dziesiętny ilorazu iloczynu kart na stosie B oraz celu B. Z obu fitnessów wylicza się średnią, która jest ostateczną wartością przystosowania osobnika.
5. Wybór nowego pokolenia – prawdopodobieństwo wybrania danego osobnika do nowego pokolenia jest proporcjonalne do $e^{-\frac{k(x-m)}{\sigma}}$, gdzie k – presja selekcyjna, x – wartość funkcji dopasowania, m – średnia wartości funkcji dopasowania w pokoleniu, σ – odchylenie standardowe funkcji dopasowania w pokoleniu. Na postawie listy prawdopodobieństw losuje się nowe pokolenie.

Raport z testowania

Testy polegają na przeprowadzeniu serii 10 pomiarów dla różnych, wybranych konfiguracji. Mamy 30 kart, cel na stosie A to suma od 4 do 20, a cel na stosie B to iloczyn liczb od 1 do 3 i od 21 do 30.

Badanie 1. Wpływ liczebności populacji na skuteczność znalezienia rozwiązania.
Założenia: max 1000 iteracji, presja selekcyjna 1.0, mutacja 5%

Liczebność populacji	10	100	250	500	750	1000	1100
Udane dopasowanie	0/10	8/10	10/10	10/10	10/10	10/10	10/10
Średni czas działania	1.109	2.951	5.006	4.479	14.583	10.265	18.808
Średnie ostatnie pokolenie	1000.0	349.0	236.1	103.4	223.7	105.1	223.6
Mediana czasu	1.100	1.426	4.813	3.423	5.814	7.085	15.149
Mediana ostatniego pokolenia	1000.0	156.5	224.5	78.5	88.5	72.0	180.5

Badanie 2. Szybkość osiągnięcia wybranych poziomów satysfakcji.
Założenia: 100 osobników w populacji, max 500 iteracji, presja selekcyjna 1.2, mutacja 1%

Zadana satysfakcja	1.00	0.99	0.96	0.93	0.90
Średni czas działania	4.091	3.580	1.561	0.175	0.177
Udane dopasowanie	0/10	1/10	7/10	10/10	10/10
Średnie ostatnie pokolenie	500.0	452.4	187.2	19.8	19.8

Badanie 3. Wpływ mutacji na skuteczność znalezienia rozwiązania.
Założenia: 100 osobników w populacji, max 1000 iteracji, presja selekcyjna 1.0

Procentowy udział mutacji	Sukcesy na 10 prób Switch/Swap	Średnie ostatnie pokolenie Switch/Swap	Mediana ostatniego pokolenia Switch/Swap
0.5%	0/1	1000.0/970.1	1000.0/1000.0
1%	2/0	932.5/1000.0	1000.0/1000.0
5%	8/9	349.0/568.3	156.5/487.5
10%	4/6	893.2/586.4	1000.0/564.0
20%	0/0	1000.0/1000.0	1000.0/1000.0

Założenia: 1000 osobników w populacji, max 200 iteracji, presja selekcyjna 1.0

Procentowy udział mutacji	Sukcesy na 10 prób Switch/Swap	Średnie ostatnie pokolenie Switch/Swap	Mediana ostatniego pokolenia Switch/Swap
0.5%	4/2	147.6/164.5	200.0/200.0
1%	4/4	145.4/142.8	200.0/200.0
5%	10/9	72.2/99.4	58.5/67.5
10%	2/6	178.8/131.1	200.0/122.0
20%	0/1	200.0/185.8	200.0/200.0

Wnioski, pomysły, usprawnienia

Badanie pierwsze pokazuje, że dla zadanego problemu istnieje określona liczba osobników, dla której działanie algorytmu jest najefektywniejsze. Jeśli liczba osobników jest za mała, to algorytm nie jest w stanie przeszukać odpowiednio dużej części przestrzeni, aby znaleźć rozwiązanie. Z kolei, jeśli liczba osobników jest zbyt duża, to działanie algorytmu jest spowolnione i przypomina bardziej brute force niż ewolucję. Badanie pokazało, że optymalną liczbą będzie około 500 osobników.

W badaniu drugim widzimy, że nasz algorytm bardzo szybko znajduje dobre rozwiązanie, jednak znalezienie rozwiązania idealnego wymaga dobrania odpowiednich narzędzi do skali problemu.

W badaniu trzecim obserwujemy, że istnieje współczynnik mutacji, którego wpływ na algorytm jest najkorzystniejszy. Jeśli współczynnik mutacji jest zbyt mały, to algorytm nie jest w stanie efektywnie poszukiwać rozwiązań, gdyż szybko zbiega do niekoniecznie najlepszych minimów lokalnych. Dla zbyt dużego współczynnika mutacji algorytm nie jest w stanie zbiec do swoich ekstremów, ponieważ genotypy są zbyt często zmieniane. Według badania optymalny współczynnik mutacji to około 5%.

Algorytm bardzo dobrze radzi sobie z szybkim znajdowaniem dobrego rozwiązania, jednak ma utrudnione zadanie przy poszukiwaniu najlepszego ze względu na specyfikę problemu. Znaczący jest ogrom przestrzeni do przeszukania, przy 30 kartach mamy ponad miliard możliwych ustawień. Populacja 1000 osobników jednorazowo sprawdza jedynie 1 milionową tej przestrzeni.

Funkcja zawiera wiele minimów. Czasami różnica pomiędzy minimum globalnym a tym, gdzie aktualnie znajduje się funkcja to kwestia jednej karty, jednak wymaga to odpowiedniej mutacji oraz wybrania dobrze zmutowanego dziecka. Przy małej liczebności pokolenia szansa na to jest znikoma.

Wybieranie osobników do nowego pokolenia z prawdopodobieństwem wykładniczym względem funkcji przystosowania pozwoliło osobnikom o mniejszym aktualnym przystosowaniu przetrwać. Osobniki te, pomimo słabego przystosowania, mogą posiadać potencjalnie dobre geny, które rozwiną się później.

Zauważyliśmy, że ważnym czynnikiem wpływającym na szybkość oraz jakość rozwiązania jest odpowiednie dobranie wielkości populacji. Powinna ona zależeć od rozmiaru problemu.

Zwięzła instrukcja obsługi

```
python algorithm.py [-s S] [-p P] N A B
```

N - liczba kart, A - oczekiwania wartość na stosie A, B - oczekiwana wartość na stosie B, S - oczekiwany poziom dopasowania, P - ilość osobników w populacji