Dokumentacja

Autorzy:
Artur Wyrozębski
Mateusz Szczepkowski

Problem:
Nauczycielka w przedszkolu chce rozdać ciastka dzieciom w swojej grupie. Dzieci siedzą w linii obok siebie (i nie zmieniają tych pozycji). Każde dziecko ma przypisaną ocenę s_i , $i \in (1,2,...,n)$, zgodnie z wynikiem testu umiejętności. Nauczycielka chce dać każdemu dziecku co najmniej jedno ciastko. Jeśli dzieci siedzą obok siebie, dziecko z wyższą oceną musi dostać więcej ciastek niż to z niższą oceną. Nauczycielka ma ograniczony budżet, więc chce rozdać jak najmniej ciastek.
Należy zaimplementować algorytm bazujący na algorytmie ewolucyjnym, który zwróci najmniejszą liczbę ciastek, które musi rozdać nauczycielka.

Informacje ogólne:
Zastosowany został algorytm (μ+λ). Wartość domyślna μ to 200. Wartość domyślna λ to 150.
Kod jest napisany w języku Python 3. Jedynymi zastosowanymi pakietami są random, math oraz cProfile. Wykorzystano cProfile do testowania czasu działania programu.

Tworzenie populacji początkowej:
Populację początkową tworzy się poprzez losowe tworzenie osobników.
Przedział wartości z którego są losowo pobierane liczby ciastek dla poszczególnych dzieci w danym osobniku jest ściśle określony, gdzie minimum to zawsze 1, natomiast maksimum to albo liczba dzieci, albo liczba unikalnych ocen, zależnie od tego która z tych wartości jest mniejsza.

Funkcja przystosowania:
Jest to funkcja, która zwraca wartość będącą sumą liczby ciastek danego osobnika oraz kar związanych z brakiem jego poprawności.
Kary są przyznawane, gdy dziecko na danej pozycji ma więcej punktów niż dziecko po swojej lewej lub prawej stronie, ale otrzyma mniej albo tyle samo ciastek niż któryś z sąsiadów mający mniej punktów.
Wartość kary to różnica ilości ciastek pary dzieci, którym błędnie przydzielono ciastka jak w poprzednio określonych przypadkach, oraz dodatkowo stała liczba w postaci dwóch ciastek.
Czasami zachodzi potrzeba propagacji tejże kary na inne dzieci. Należy tak postąpić, gdy dziecko, które ma mniejszą liczbę ciastek niż powinno mieć w danej parze dzieci, ma po swojej drugiej stronie osoby, których oceny tworzą rosnący ciąg liczb. Wynika to z tego, że jak poprawi się liczbę ciastek na określonej pozycji, to ta poprawa może wpłynąć na ilość ciastek sąsiadów, którzy mają większą ich liczbę.
Największym problemem jest sytuacja, gdzie dzieci będące obok siebie mają tę samą ocenę. Jeżeli będzie przyznawana kara do funkcji przystosowania za jakąkolwiek różnicę w ilości ciastek w tym przypadku, to może się zdarzyć, że liczba ciastek ciągu dzieci mających tę samą ocenę nigdy się nie zmieni i zawsze będzie liczbą wysoką, bo funkcja przystosowania bardziej karze niż nagradza za jakąkolwiek zmianę w tym wypadku.
Rozwiązaniem tego problemu jest określanie kary tylko między dziećmi o różnej ocenie, to znaczy jeżeli po prawej stronie od danego dziecka jest osoba, która ma tę samą ocenę, to sprawdza się kolejne dziecko po jego prawej stronie czy ma inną ocenę. Jeżeli ma znowu tę samą, to sprawdza się kolejne, aż dojdzie się do dziecka o innej ocenie. Względem niego określa się kary.
Analogicznie jest po lewej stronie od danego dziecka.
Napisany algorytm funkcji przystosowania ma złożoność liniową względem liczby dzieci. Do obliczania wartości kary stosowane są dwa indeksy – lewostronny i prawostronny.
Indeksy te określają pary, między którymi należy sprawdzić poprawność.

Algorytm funkcji przystosowania kończy się, gdy prawostronny indeks dojdzie do końca tablicy ciastek danego osobnika.
Indeks prawostronny jest przesuwany tylko w prawo zawsze o jedną pozycję, tak samo jak lewostronny. Są różne warunki przesuwania tych indeksów, które znajdują się w kodzie programu.
Idea sprawdzania poprawności par w kodzie jest taka sama jak poprzednio opisano.

Selekcja rodziców:
Została zastosowana selekcja turniejowa, która okazała się lepsza, niż metoda ruletki oraz metoda rankingowa po przeprowadzonych badaniach.
Wartości funkcji przystosowania nie różnią się wiele, więc szanse wyboru osobników w metodzie koła ruletki są do siebie zbliżone.
Selekcja rankingowa jest znacznie bardziej złożona obliczeniowo niż poprzednie dwie metody ze względu na potrzebę sortowania, przez co znacznie spowalnia algorytm ewolucyjny.
Wielkość zbioru do selekcji turniejowej wynosi 30/100 wielkości populacji.
Mniejsza wartość sprawia, że jako rodzice częściej wybierani są osobnicy o gorszej funkcji przystosowania, natomiast większa – spowalnia algorytm oraz częściej wybierani są osobnicy o lepszej funkcji przystosowania, co powoduje szybszą stagnację algorytmu.

Wpływ poszczególnych selekcji:
Czas wykonania, średnia wartość funkcji przystosowania najlepszego rozwiązania:
[4, 8, 8, 8, 9, 10, 7, 6, 6, 6, 6, 2, 3, 4, 10, 10, 3, 10, 10, 5, 4, 3, 3, 3, 3, 3, 2] (27 osobników)
Wielkość populacji: 200 Ilość potomków: 150 Liczba generacji: 100
Selekcja turniejowa z wielkością zbioru 5/100: 2,193 sekund, 88
Selekcja turniejowa z wielkością zbioru 30/100: 3,105 sekund, 70
Selekcja turniejowa z wielkością zbioru 50/100: 3,980 sekund, 72
Selekcja rankingowa: 35,115 sekund, 145
Metoda koła ruletki: 33 sekundy, 163 (wysoki czas ze względu na potrzebę sumowania odwrotności wszystkich wartości funkcji przystosowania danej populacji)

Krzyżowanie:
Wybrano krzyżowanie wielopunktowe.
Pozwala ono na przekazywanie ciągów ciastek osobników rodziców, przez co następuje minimalizacja kar funkcji przystosowania.
Przy każdym krzyżowaniu powstaje dwoje dzieci, gdzie najpierw do pierwszego osobnika przekazywany jest ciąg pierwszego rodzica, potem drugi ciąg drugiego rodzica, natomiast do drugiego osobnika przekazywany jest najpierw pierwszy ciąg drugiego rodzica, potem drugi ciąg z pierwszego rodzica i tak dalej, na zmianę. Ciągi są określone w tych samych pozycjach dla pierwszego rodzica co dla drugiego.
Średnia długość ciągu to 4 wartości.

Mutacja:
Jest 20% szansy, że dany osobnik potomny będzie podlegał mutacji.
Najpierw należy określić liczbę pozycji gdzie następuje mutacja, która wynosi 1/30 długości problemu. Liczba ta jest następnie zaokrąglana w górę do wartości całkowitej.
Następnie określa się losowo indeksy tych pozycji.
Mutacja polega na zmianie poprzednio określonych pozycji osobnika na losowe wartości ciastek z określonego przedziału.
Przedział ten jest taki sam jak przy tworzeniu populacji początkowej.

Tworzenie następnej populacji:
Łączy się zbiór dzieci oraz dotychczasowej populacji i następną populację tworzą osobniki najlepiej przystosowane, czyli o najmniejszej wartości funkcji przystosowania.