Algorytmy ewolucyjne

Sprawozdanie z projektu nr 2

Kamil Matysek, Mateusz Roszkowski

Zadaniem w projekcie 2 było znalezienie rozwiązania problemu plecakowego stosując algorytm ewolucyjny.

Do wykonania projektu został stworzony skrypt w matlabie *proj2.m,* gdzie został zaimplementowany algorytm genetyczny. Funkcja celu wykorzystywana przez algorytm genetyczny znajduje się w pliku *fun.m*.

Czynniki, które będą testowane w czasie projektu:

* Rozmiar populacji
* Liczba iteracji
* Rozmiar elity
* Prawdopodobieństwo mutacji
* Stosunek mutacji krzyżowych do mutacji binarnych

Wstępne kryterium stopu: osiągnięcie założonej liczby iteracji

Dane bazowe:

* Rozmiar populacji - 100
* Liczba iteracji – 100
* Rozmiar elity - 33
* Prawdopodobieństwo mutacji – 1/16
* Stosunek mutacji krzyżowych do mutacji binarnych – 1:1

Graphical user interface, application

Description automatically generated

*Wyk 1. Wykres testowy wartości maksymalnej w funkcji numeru iteracji dla wartości bazowych*

Wynik: 1 1 0 0 0 0 1 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0

Wartość f. celu: 944

Poniższe testy mają na celu sprawdzenie jakie powinny być wartości zmiennych aby algorytm działał jak najlepiej, czyli znajdował najlepsze rozwiązanie funkcji celu.

**Testowanie liczności elity**

*Poniższy test odbył się przy użyciu wartości bazowych i jedynie przy zmianie wartości rozmiaru elity*

Graphical user interface

Description automatically generated with medium confidence

A picture containing graphical user interface

Description automatically generated

Graphical user interface, application

Description automatically generated

Chart

Description automatically generated

Z powyższego testu widać, że zbyt duży rozmiar elity powoduje dłuższe szukanie minimum. Mniejsza wartość elity pozwala na znalezienie lepszego minimum ze względu na dużą liczbę mutacji.

Przy kolejnych testach wartość rozmiaru elity będzie wynosić **5**

**Testowanie prawdopodobieństwa mutacji**

Graphical user interface

Description automatically generated with low confidence

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

Chart

Description automatically generated

Małe prawdopodobieństwo mutacji sprawia, że algorytm może utknąć w minimum lokalnym. Duże prawdopodobieństwo sprawia, że algorytm trochę dłużej szuka minimum lokalne.

Kolejne testy zostaną przeprowadzone z prawdopodobieństwem mutacji równym 1/8

**Testowanie proporcji mutacji jednopunktowych do liczby mutacji bitowych**

Graphical user interface, application

Description automatically generated

Chart

Description automatically generated with medium confidence

Graphical user interface, chart

Description automatically generated with medium confidence

Graphical user interface, chart

Description automatically generated

Mutacje krzyżowe są wydajniejsze w znajdowaniu minimum funkcji aniżeli mutacje binarne.

**Najlepszy wynik:**

Wartości czynników:

* Populacja – 2000
* Rozmiar elity – 5
* Prawdopodobieństwo – 1/16
* Stosunek mutacji krzyżowej do binarnej – 4:1

Kryterium stopu: osiągnięcie 100 iteracji

Zwiększenie liczności populacji znacznie poprawiło wyniki uzyskiwane przez algorytm, niestety spowodowało to z wydłużenie czasu obliczeń.

Graphical user interface, application

Description automatically generated

Chart

Description automatically generated

Graphical user interface, chart

Description automatically generated with medium confidence

Chart

Description automatically generated

Wynik: 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0

Wartość funkcji celu: 1034

Kryterium stopu: wariancja równa mniejsza niż 40

Graphical user interface, application

Description automatically generated Chart, line chart

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface, chart

Description automatically generated

W 116 iteracji wariancja osiąga wartość 36.97, co spełnia kryterium stopu. Wartość funkcji celu nie zmieniła się i nadal wynosi 1034.

Wynik: 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0

Wykorzystanie funkcji ga() do znalezienia optymalnego rozwiązania.

W celu sprawdzenia czy nasz algorytm działa dobrze, do rozwiązania problemu plecakowego wykorzystaliśmy również funkcję ga() (skrypt zawarty w pliku *gaa.m*). Najlepszy uzyskany wynik przy użyciu funkcji ga() z domyślnymi parametrami, był tożsamy z tym uzyskany naszym autorskim algorytmem.

Postanowiliśmy również porównać skuteczność obu algorytmów na podstawie 25 niezależnych uruchomień, w tym przypadku aby skrócić czas obliczeń parametry naszego algorytmu ustawiliśmy następująco:

rozmiar\_populacji=1000;

rozmiar\_elity=1;

liczba\_iteracji=50;

praw\_mutacji=1/16;

liczba\_pot\_krzyz=400;

liczba\_pot\_mut=700;

Funkcja ga() dała średni wynik 1019.3, podczas gdy nasz algorytm 1032.7, uzyskał on wynik optymalny (1034) w 22 z 25 prób, za każdym razem robił to w mniej niż 50 iteracji. Jednak dużą zaletą jest szybkość z jaką uzyskujemy wynik przy użyciu funkcji ga(), mimo że potrzebuje ona często więcej iteracji (średnio 66) to przez to, że populacja jest znacznie mniejsza niż w naszym algorytmie, obliczenia wykonują się znacznie szybciej.

Rozwiązanie problemu:

Przedmioty, które należy spakować: [1,2,3,6,7,10,11,14,15,16,24,25,26,27,29]

[1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0]

Waga optymalnego plecaka = 5.1

Dopuszczalna waga plecaka = 5.13

Wartość optymalnego plecaka = 1034