PSZT Dokumentacja do projektu

Paweł Martyniuk, Iwo Sokal

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 najbliższą 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 C++. Mamy N kart z wartościami od 1 do N. Każda karta musi być przypisana do jednego ze stosów kart. Zastosowane zostało kodowanie binarne: jeżeli karta ma wartość 0 to należy do zbioru A, natomiast w przeciwnym przypadku do zbioru B. Celem algorytmu jest przypisanie tak kart do obu grup, aby suma w zbiorze A oraz iloczyn kart w zbiorze B były najbliższe wartościom nadanym z satysfakcjonującym poziomem dopasowania. Przyjęliśmy, że dla najlepszych osobników wartość funkcji przystosowania zbiega do 0. Im wartość jest dalej od zera tym osobnik jest gorzej przystosowany. Warunkiem zatrzymania algorytmu uznaliśmy za osiągnięcie satysfakcjonującego poziomu dopasowania albo wykonanie 1000 iteracji algorytmu.

Podział zadań

Iwo Sokal:

- Inicjalizacja pierwszego pokolenia
- Ocena osobników
- Selekcja osobników

Paweł Martyniuk:

- Krzyżowanie osobników
- Mutacja
- Testowanie i tworzenie statystyk

Sposób realizacji

W celu znalezienia najefektywniejszego rozwiązania problemu zdecydowaliśmy się zaimplementować kilka różnych metod dla niektórych funkcji.

Zaimplementowane metody:

1) Inicjalizacja pierwszego pokolenia - wypełnienie losowymi osobnikami.
2) Krzyżowanie
   a) Jednopunktowe z tym samym punktem krzyżowania dla wszystkich osobników
   b) Jednopunktowe z losowym punktem krzyżowania dla każdej krzyżowanej pary
   c) Dwupunktowe z tymi samymi punktami krzyżowania dla wszystkich osobników
   d) Dwupunktowe z losowymi punktami krzyżowania dla każdej krzyżowanej pary

3) Mutacja - dla danego osobnika zmieniany jest jeden bit na przeciwny. Przy starcie jako jeden z argumentów przekazywane jest prawdopodobieństwo bazowe mutacji. Jest to mutacja dla karty o wartości 1. Każda następna karta ma prawdopodobieństwo mutacji o 20% mniejsze od poprzedniej.
4) Ocena przystosowania: Max (dopasowanie A, dopasowanie B)

   dopasowanie A: |aspiracjaA - sumaA|/ aspiracjaA

   dopasowanie B: |aspiracjaB - iloczynB|/ aspiracjaB

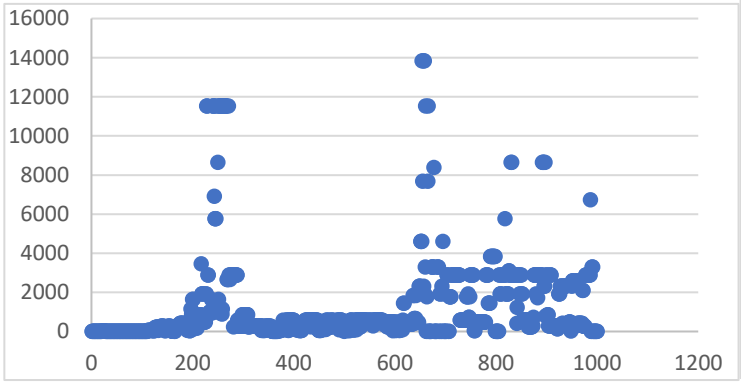
5) Selekcja:
   a) Losowa - spośród populacji rodziców wybieramy N losowych osobników.
   b) Turniejowa - spośród populacji losujemy dwóch osobników i wybieramy spośród nich tego który ma lepszą ocenę (funkcję przystosowania).
6) Sukcesja generacyjna – nowe pokolenie składa się wyłącznie z dzieci, a więc otrzymanych w danej iteracji nowych osobników.

Testowanie

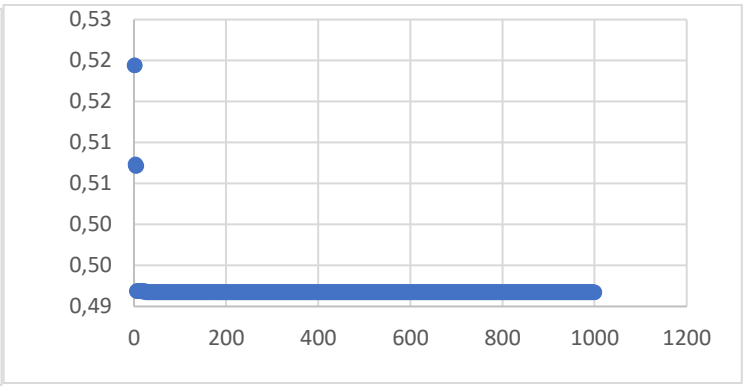
Dane dobrane do tej części testów: A = 19400, B = 6.89557e+197, poziom dopasowania = 0.3, bazowe prawdopodobieństwo mutacji = 50, prawdopodobieństwo krzyżowania = 80, liczba osobników = 100, liczba kart = 200.

Wykresy zależności najlepszego osobnika danej populacji od generacji:

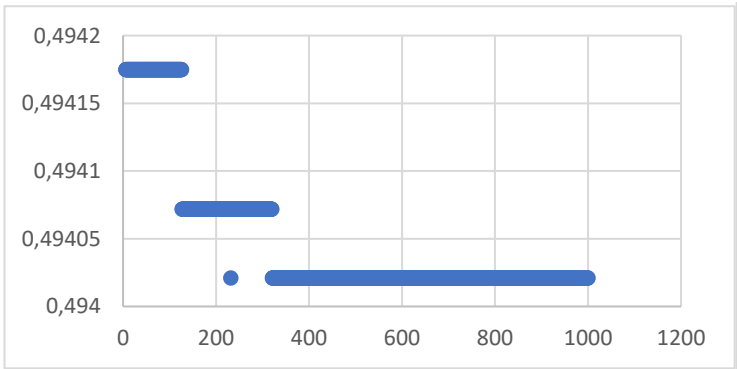
Selekcja losowa, krzyżowanie jednopunkowe



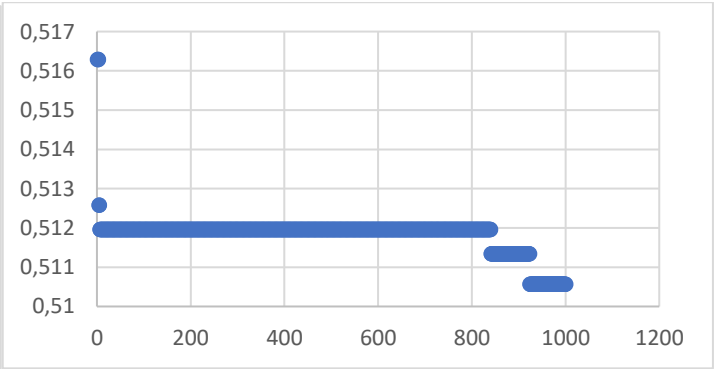
Selekcja turniejowa, krzyżowanie jednopunkowe



Selekcja turniejowa, krzyżowanie dwupunktowe (80, 150)



Selekcja turniejowa, krzyżowanie dwupunktowe losowe



Czas pracy programu w zależności od N kart i K osobników:

N \ K	10	100	200	400
10	0.076	0.548	1.117	2.222
50	0.150	1.128	2.185	4.432
100	0.180	1.390	2.709	5.349
200	0.326	2.517	4.900	9.560

Wnioski i przemyślenia

Sprawienie, aby populacja początkowa była inicjalizowana nie w sposób losowy, ale bardziej zróżnicowany, pozwoliłoby na zwiększenie przestrzeni poszukiwania i być może znalezienie lepszego rozwiązania. Byłoby to jednak kłopotliwe do realizacji, dlatego użyliśmy jedynie inicjalizacji losowej.

Bez porównania selekcja turniejowa daje lepsze wyniki od losowej, natomiast krzyżowanie dwupunkowe pozwala lepiej znajdować bardziej zróżnicowane wyniki.

Wybranie takiego działania mutacji miało na celu to, aby karty o wyższych wartościach rzadziej mutowały od tych o wartościach mniejszych. Gdybyśmy często zmieniali przydział do kupki kart o dużych wartościach, wtedy wartość funkcji dopasowania by się zmieniała skokowo, a dzięki temu odbywa się to łagodnie.

W celu usprawnienia działania algorytmu, można by było dodać sukcesję, czyli losować n elementów z n-elementowej populacji, posortować względem jakości dopasowania i wybrać tylko połowę najlepszych z nich. Dzięki temu mielibyśmy pewność, że każda kolejna generacja byłaby nie gorsza od poprzedniej.

Obserwując zjawisko, że krzyżowanie dwupunktowe jest lepsze od jednopunktowego, można by zaimplementować krzyżowanie trójpunktowe, które być może okazałoby się jeszcze lepsze.

Nie jest łatwo stwierdzić czy w danym przypadku eksploracja jest lepsza od eksploatacji i ten projekt jest na to dowodem. Wydaje nam się jednak, że algorytm działa lepiej, gdy trochę bardziej skupia się na eksploatacji, ze względu na liczne ekstrema lokalne. Mimo to powinien być utrzymany balans pomiędzy obiema cechami.

Instrukcja dla użytkownika programu

Program jest uruchamiany z konsoli za pomocą komendy: ./bin/program [liczba kart N] [suma A] [iloczyn B] [poziom dopasowania G]. Program wyświetla najlepszego osobnika w populacji początkowej, a następnie przechodzi do wykonania algorytmu. Po wykonaniu algorytmu wyświetla informację o tym w której populacji znaleziono rozwiązanie oraz informację o najlepszym znalezionym osobniku. Na końcu wyświetla czas wykonania programu.