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:

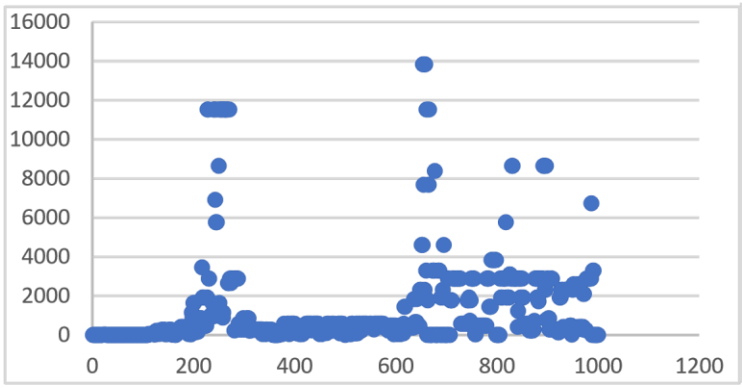
1) Inicjalizacja pierwszego pokolenia - wypełnienie losowymi osobnikami.
2) Krzyżowanie
   a) Jednopunktowe z tym samym punktem krzyżowania dla wszystkich osobników
   b) Jednopunktowe z losowym punktem krzyżowania dla każdej krzyżowanej pary
   c) Dwupunktowe z tymi samymi punktami krzyżowania dla wszystkich 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
   Ocena jest tym lepsza, im jest bliżej 0.
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

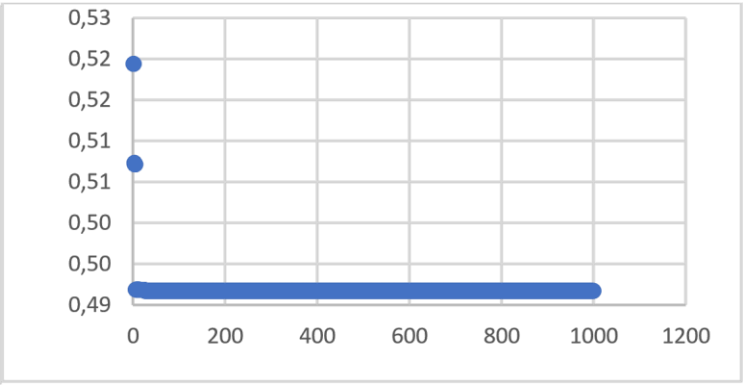
Dla wszystkich wykonanych testów istniało rozwiązanie o ocenie 0, czyli o dokładnie takiej sumie A i iloczynowi B jak to szukane. Dane dobrane do tej części testów: A = 19400, B = 6.89557e+197, poziom dopasowania = 0.3, bazowe prawdopodobieństwo mutacji = 50, prawdopodobieństwo krzyżowania = 80, liczba osobników = 100, liczba kart = 200.

Wykresy zależności najlepszego osobnika danej populacji od generacji(na osi pionowej wartość dopasowania dla najlepszego osobnika generacji, a na osi poziomej numer generacji). Ten test pokazuje tylko różnice pomiędzy zastosowanymi algorytmami.

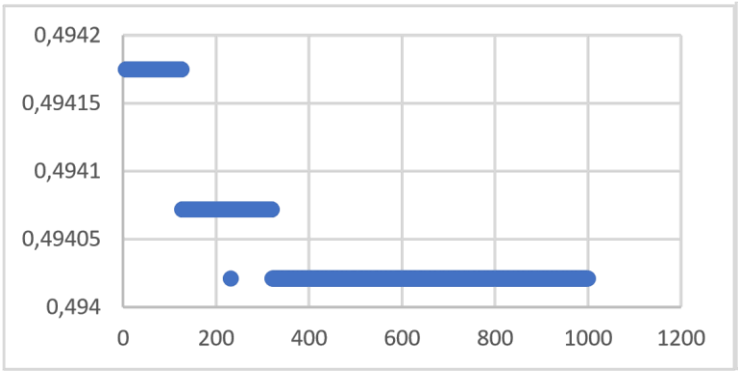
Selekcja losowa, krzyżowanie jednopunkowe



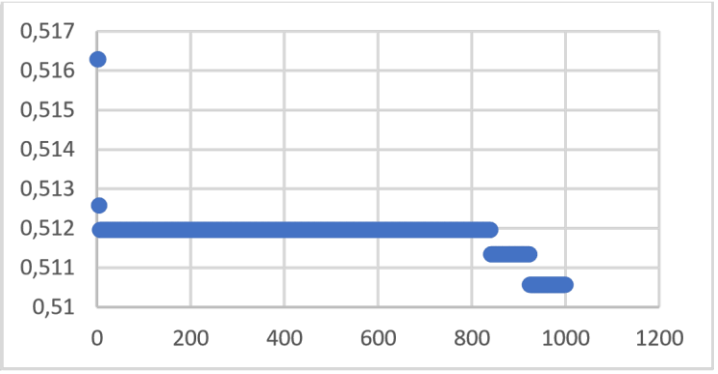
Selekcja turniejowa, krzyżowanie jednopunkowe



Selekcja turniejowa, krzyżowanie dwupunktowe (80, 150)



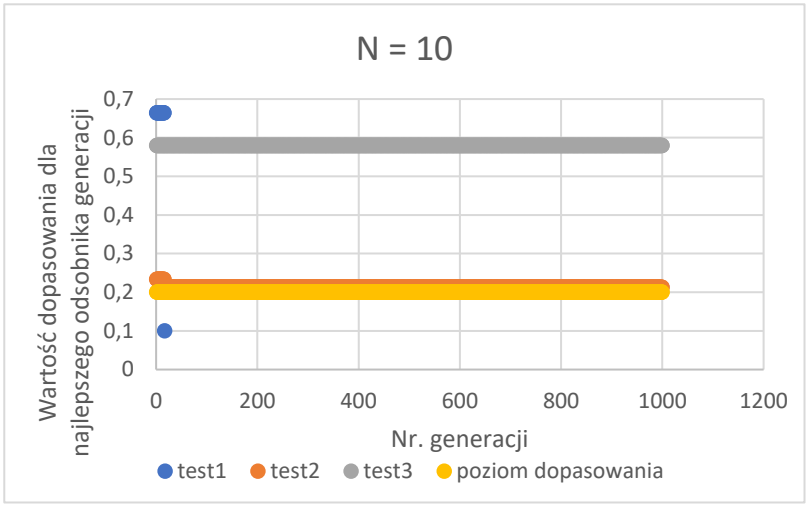
Selekcja turniejowa, krzyżowanie dwupunktowe losowe

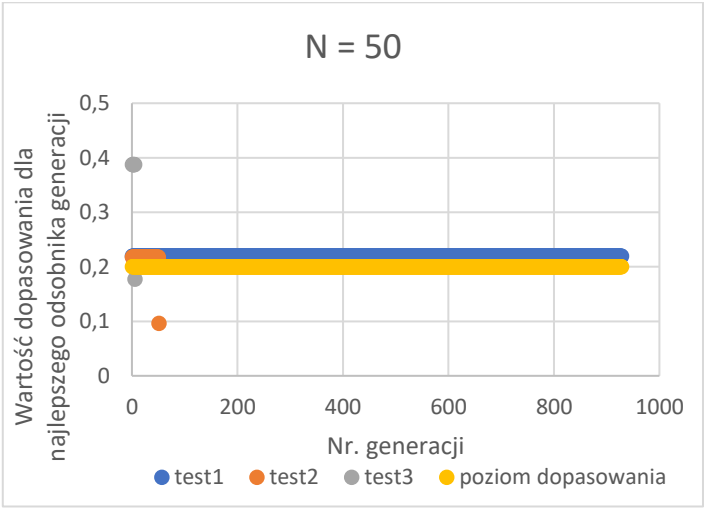


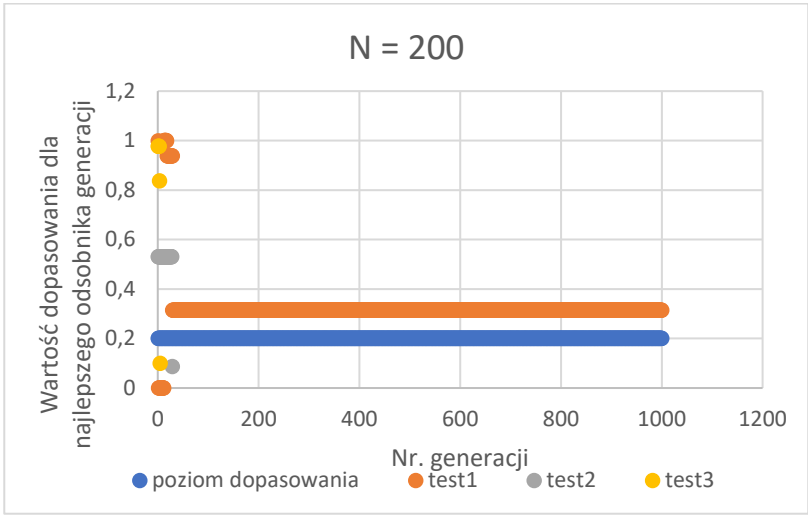
Czas pracy programu w zależności od N kart i K osobników (3 wyniki):

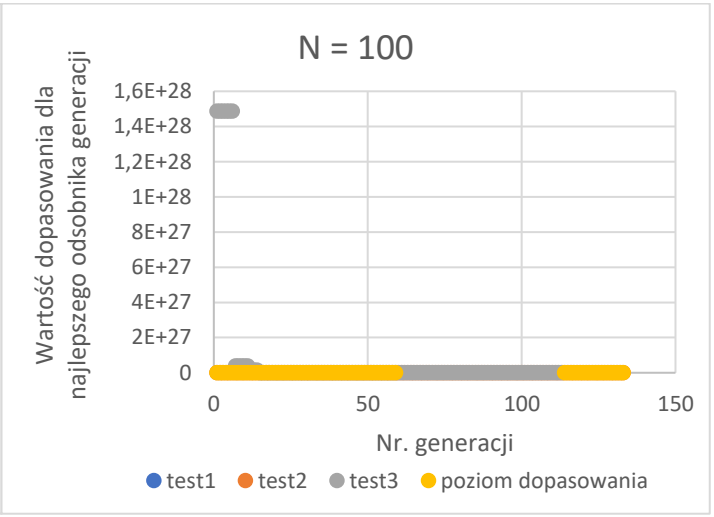
K	10	100	200	400
N(aspA, aspB, dopas.)	max_iter	0.000625	0.000992	0.002235
	0.000326	0.000705	0.002115	0.002257
10(30, 4000, 0.2)	0.000307	0.000605	0.002442	0.002275
50(834, 8.97724e+22, 0.2)	max_iter	0.032618	0.432129	0.005429
	0.039116	0.03202	0.387544	0.004954
	0.041692	0.033124	0.414664	0.005536
100(4280, 2.36489e+26, 0.2)	max_iter	max_iter	0.883481	1.25562
	max_iter	max_iter	0.852622	1.47967
	max_iter	max_iter	1.02716	1.4681
200(11043, 6.17403e+155, 0.2)	max_iter	0.020559	0.351749	0.156594
	max_iter	2.76964	0.06127	0.155597
	max_iter	0.849996	max_iter	0.155645

W celu wyznaczenia optymalnych aspiracji A oraz aspiracji B uruchomiłem wstępnie program i zapisałem poziom dopasowań A i B dla najlepszego osobnika.
Wykresy dla różnych wartości N(parametry aspiracji i poziom dopasowania są tu takie same jak w tabelce powyżej dla danego N):









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 dwupunktowe pozwala lepiej znajdować bardziej zróżnicowane wyniki. Wynika to z tego, że selekcja turniejowa wybiera najlepsze osobniki do rozmnażania, a więc każda kolejna populacja powinna być na ogół lepsza od poprzedniej. Przy użyciu krzyżowania dwupunktowego można otrzymać większą liczbę różnych osobników potomnych, dzięki czemu możliwe jest lepsze rozpatrzenie większej przestrzeni poszukiwań.

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 zrobić inną sukcesję, np. 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.

Liczba osobników w populacji ma znaczenie. Im jest ich więcej, tym większa szansa na znalezienie lepszego rozwiązania, ponieważ jest zapewniona większa różnorodność. Minusem jest to, że zwiększa się również czas wykonywania poszczególnych iteracji, co może prowadzić w niektórych przypadkach do zwiększenia czasu działania algorytmu. Jednak według nas większa zdolność do ,,przebijania się" przez minima lokalne, jaką daje większa liczba osobników w populacji, jest ważniejsza, szczególnie jeśli chce się otrzymać jak najlepsze rozwiązanie.

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.

Po przeprowadzeniu testów uznaliśmy, że nasz algorytm genetyczny jest gorszy od algorytmu typu *brute force* dla małych N, ale lepszy dla dużych N. Algorytm genetyczny charakteryzuje się tym, że szybciej znajduje lepsze wartości, ale nie rozpatruje on całej przestrzeni poszukiwań i po jakimś czasie spowalnia i się zatrzymuje. Natomiast algorytm typu *brute force* próbuje zawsze znaleźć najlepsze rozwiązanie, więc jeśli starczy mu czasu na przeanalizowanie całej przestrzeni poszukiwań, to je znajdzie. W związku z tym dla odpowiednio dużych N algorytm typu *brute force* nie zdoła znaleźć odpowiednio dobrego rozwiązania, ponieważ zajęłoby mu to zbyt dużo czas, a nasz algorytm ma na to szansę. Dla małych N, algorytm typu *brute force* zdoła znaleźć to rozwiązanie stosunkowo szybko i będzie to rozwiązanie na ogół lepsze niż otrzymane przez nasz algorytm.

Instrukcja dla użytkownika programu

Program jest uruchamiany z konsoli za pomocą komendy: ./bin/program [liczba kart N] [suma A] [iloczyn B] [poziom dopasowania G] [liczba osobników]. Poziom dopasowania to w rzeczywistości wartość funkcji oceny, do której dążymy. 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 najlepszego znalezionego osobnika. Na końcu wyświetla czas wykonania programu. Wyświetlany wynik jest podzielony na 2 części: najpierw wyświetlane są wszystkie atrybuty rozwiązania (obiektu Specimen), a następnie wyświetlana informacja, na której kupce jest karta o danej wartości. Maksymalną liczbę wykonań pętli głównej algorytmu można zmienić w kodzie, podobnie jak rodzaje używanych krzyżowań i selekcji. Domyślnie ustawione jest max 1000 przebiegów pętli, selekcja turniejowa oraz krzyżowanie dwupunktowe z losowymi punktami krzyżowania.