PSZT, Projekt 1
1.5 Karty

Grzegorz Rypeść
Wojciech Wolny

1. Problem

Masz 10 kart ponumerowanych od 1 do 10. Znajdz przy użyciu algorytmu ewolucyjnego sposob na podział kart na dwie kupki w taki sposob, że suma kart na pierwszej kupce jest jak najbliższa wartosci A, a suma kart na drugiej kupce jest jak najblizsza wartosci B.

2. Analiza zadania

Przestrzeń rozwiązań tego problemu, to jest możliwych zbiorów A i B, jest dyskretna. Stąd podejście genetyczne, które koduje genotypy za pomocą wektorów binarnych, wydaje się najbardziej rozsądne. Postanowiliśmy jednak rozwiązać zadanie przy użyciu algorytmu zarówno genetycznego i ewolucyjnego, a następnie je ze sobą porównać. Zauważmy też, że na dobrą sprawę liczba możliwych kombinacji zbiorów A i B nie jest duża, wynosi 3 do potęgi 10 czyli 59049 (każdą kartę zaliczamy do A, B lub do żadnego ze zbiorów). Stąd w praktyce algorytm brutalny przeglądający wszystkie rozwiązania wykonałby się zadowalająco dla nas szybko i nie ma potrzeby używania heurystyki. Może warto rozważyć zwiększenie liczbę kart.

3. Funkcja przystosowania

W obydwu podejściach do problemu funkcję przystosowania dla osobnika/fenotypu definujemy tak samo, a celem jest jej minimalizacja. Niech A' to suma kart ze zbioru A, analogicznie B'. Wartość funkcji przystosowania, zwana w kodzie *loss,* jest równa $\sqrt{(A-A')2+(B-B')2}$.

2. Podejście ewolucyjne (μ+λ)

W algorytmie μ+λ mamy do czynienia z pewną populacją P złożoną z μ osobników, z której wybierany jest za pomocą losowania ze zwracaniem zbiór T o wielkości lambda. Osobniki z tego zbioru poddane są krzyżowaniu poprzez uśrednianie lub interpolację, mutacji i są wrzucane do zbioru R. W naszym problemie przestrzeń jest dyskretna, stąd ciężko mówić o uśrednianiu. Zamiast tego bierzemy zbiory A i B, każdy od losowego rodzica, a karty które się powtórzą odrzucamy. Mutowanie następuje poprzez losowo wybranie karty od 1 do 10 i wrzucenie jej do losowego zbioru osobnika-dziecka. Jeśli karta już tam jest to mutacja nie następuje. Następnie zbiór P∪R sortujemy względem funkcji przystosowania i jako nowy zbiór P przypisujemy μ najlepszych osobników. Ten sposób wyboru ma dość dużą presję selekcyjną, przez co szybko powinien eksploatować minima funkcji przystosowania. Kolejną zaletą jest prosta implementacja. Wadą jest kiepska eksploracja.

3. Podejście ewolucyjne – kod

Kod tego rozwiązania zawarliśmy w pliku *miPlusLambda.py* w języku Python 3.x. Są w nim zaimplementowane dwie klasy:

- Individual – Reprezentacja osobnika populacji, składa się ona z dwóch zbiorów A i B, krzyżuje się z partnerem oraz mutuje się w sposób opisany w punkcie 2. Metody nazwaliśmy w sposób intuicyjny do algorytmu, to jest:
  - loss() - zwróć loss osobnika
  - crossover(partner) - zwróć dziecko po krzyżówce z partnerem
  - mutate() - zmutuj tego osobnika

- Environment – Środowisko, w którym ustalona jest populacja osobników, jej rozmiar, a także populacja T nazwana w kodzie *children*.

Skrypt wykonuje się od stworzenia z odpowiednimi argumentami środowiska, następnie ustalana jest liczba epok (*epoch*) i w każdej z nich dokonujemy krzyżowania, mutacji i selekcji nowego zbioru P. Warunkiem stopu jest wykonanie się *epoch* iteracji. Pierwszy osobnik w populacji jest rozwiązaniem najlepszym, tj. jego *loss* jest najmniejszy.

4. Podejście genetyczne (1+1)
Następnym algorytmem, który zaimplementowaliśmy jest algorytm 1+1. W algorytmie tym dla każdego elementu z naszej wygenerowanej pseudolosowo populacji P z n osobnikami tworzymy nowy osobnik. Osobnik ten następnie mutujemy i porównujemy jego odległość do celu z odległośćią pierwotnego osobnika. Jeśli zmutowany osobnik jest bliższy wartości docelowej wstawiamy go w miejsce pierwotnego osobnika, który usuwamy. W naszym problemie o dziedzinie dykretnej reprezentujemy osobnik przez dwie tabilce binarne o długości liczby kart – 10. Pierwsza tablica reprezentuje zbiór kart, które znajdują się w stosie A a druga w stosie B. Mutacja następuje poprzez wybranie karty przy pomocy równiania *„(sigma*N(0,10))%10”*. Jeśli ta karta należy do zbioru A, wtedy przesuwamy ją do zbioru B. Jeśli znajduje się w zbiorze B to usuwamy ją ze zbioru. Jesli karta nie znajduje się w żadnym zbiorze wstawiamy ją do zbioru A. Zaletą tego algorytmu jest fakt, że kazda kolejna populacja jest lepsza lub równa poprzedniej.

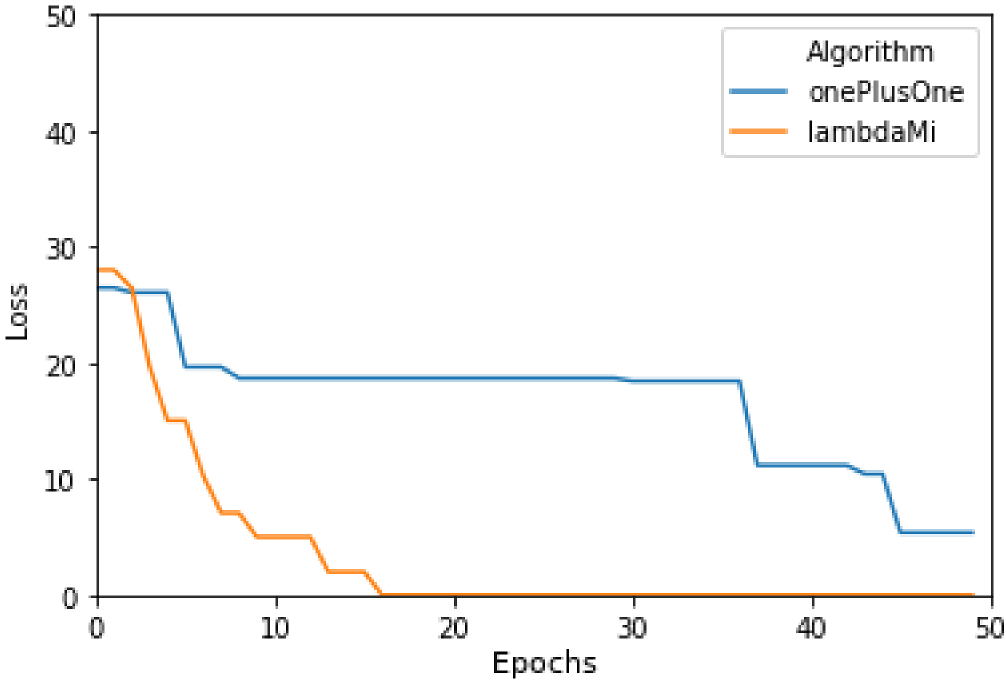
5. Podejście genetyczne – kod
Kod tego rozwiązania zawarliśmy w pliku onePlusOne.*py* w języku Python 3.x. Są w nim zaimplementowane dwie klasy:

- OnePlusOne – Reprezentacja osobnika populacji, który składa się z dwóch tablic binarnych A i B, mutuje się w sposób opisany w punkcie 4. Metody nazwaliśmy w sposób intuicyjny do algorytmu:
    - loss() - zwróć loss osobnika
    - mutate() - zmutuj tego osobnika
- EnvironmentOnePlusOne – Środowisko, w którym ustalona jest populacja osobników, jej rozmiar. Metodami tej klasy są:
    - los() - podająca sumę strat dla wszystkich osobników w srodowisku.
    - lossTop() - podająca stratę najlepszego osobnika.
    - sort() - sortująca osobników od najlepszego do najgorszego.
    - mutation() - mutująca wszystkie osobniki.

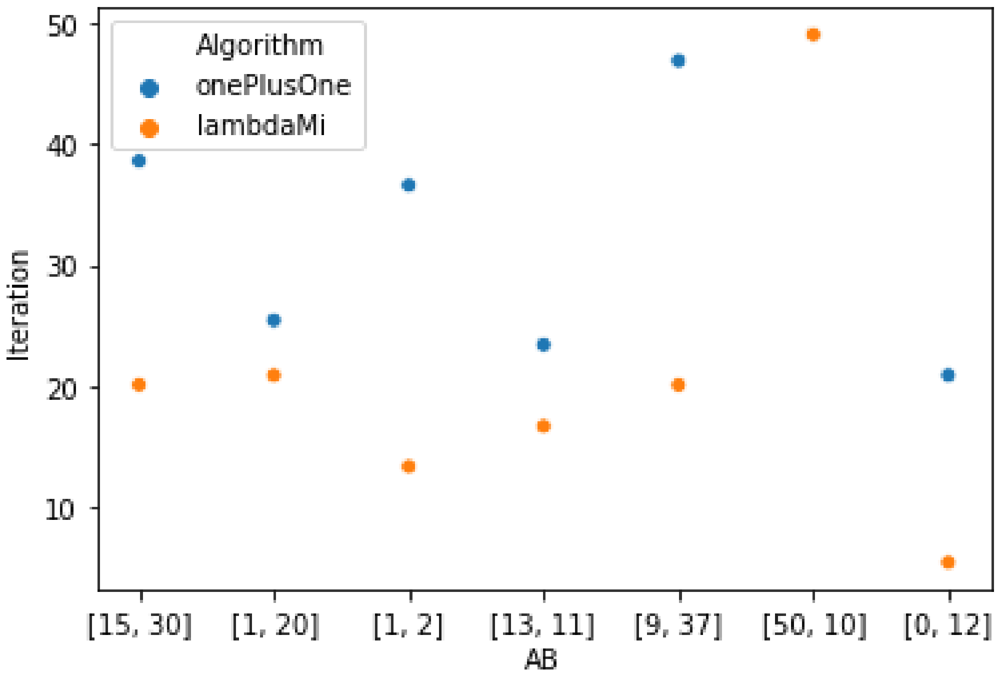
6. Porównanie algorytmów
Dla obu algorytmów przeprowadziliśmy 2 porównania. Pierwsze jest widoczne na wykresie numer 1. Jest to test dla epochs=50, A=5 oraz B =50. Zrównoleglony algorytm 1+1 miał 15 osobników, natomiat w drugim algorytmie μ=10, λ=15. Algorytm μ+λ pomimo mniejszej populacji dużo szybciej odnalazł optymalne rozwiązanie, natomiast 1+1 nie dał rady zrobić tego w 50 epokach. Wykres jest przedstawiony na Fig. 1.

W drugim porównaniu szukamy średniej liczby iteracji z 1000 uruchomień algorytmu danego algorytmu potrzebnej do znalezienia optymalnego rozwiązania. Jak widzimy na wykresie numer 2 algorytm μ+λ okazuje się znów lepszy.

Te wykresy można uzyskać korzystając z pliku *Porównanie wyników.ipynb* otworzonego w środowisku Jupyter Notebook.



Wykres 1



Wykres 2