WSI21Z

Wojciech Gierulski

Ćwiczenie 2 – Algorytmy ewolucyjne i genetyczne

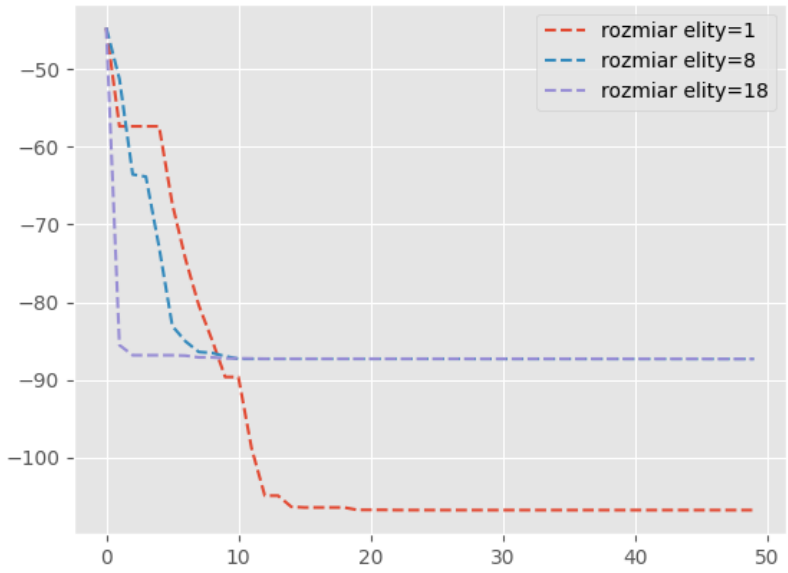
Zaimplementować algorytm ewolucyjny dla problemu minimalizacji funkcji n-zmiennych np. dwuwymiarowe. Wykorzystać selekcję turniejową i sukcesję elitarną. Kryterium stopu: wystarczy na liczbę iteracji, ale można dodać inne. Do krzyżowania użyć pełnej rekombinacji arytmetycznej. Zastosować mutację gaussowską. Wyniki przedstawić na wykresach zależności: - wartości f. celu w kolejnych iteracjach (na jednym wykresie dla różnych - wybranych parametrów np. typu populacji inicjalnej) - czasu/wyniku działania algorytmu od liczby osobników w populacji (średnia z m.in. 3 wywołań dla wybranych parametrów).

Wstęp

Algorytm ewolucyjny charakteryzuje się dużą dowolnością w doborze parametrów i metod mutacji/krzyżowania. Działanie algorytmu w zależności od wartości parametrów omówiono w dalszej części raportu. Metody krzyżowania, mutacji itp. Przyjęto jak w treści zadania. Do testów algorytmu użyto funkcji dwóch zmiennych *Bird function*.

Rozmiar elity

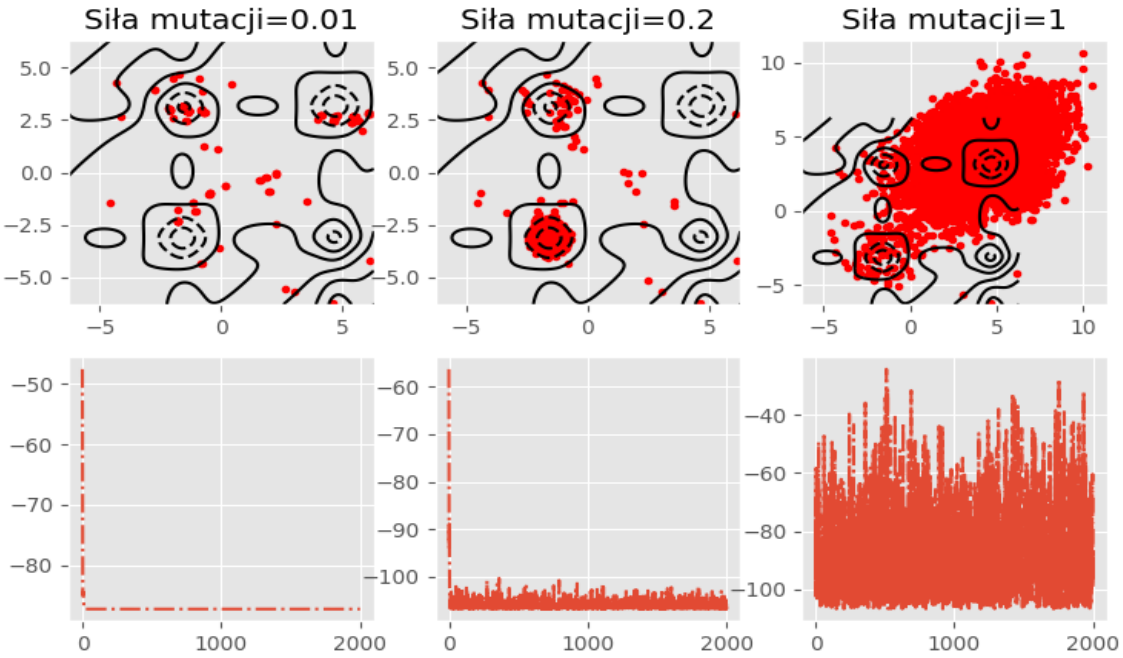
Na rysunku 1.1 przedstawiono wykres wartości funkcji celu dla najlepszego osobnika w kolejnych iteracjach, dla różnych rozmiarów elity. Wyraźnie widać, że dla dużych rozmiarów elity algorytm szybko się stabilizuje i ma tendencję do pozostawania w minimum lokalnym. Dla małego rozmiaru elity algorytm dłużej się stabilizuje i przechodzi przez różne minima lokalne.



Rys 1.1 – funkcja celu w kolejnych iteracjach w zależności od rozmiaru elity

Siła mutacji

Na rysunku 1.2 przedstawiono wykres funkcji celu dla najlepszego osobnika w kolejnych iteracjach w zależności od siły mutacji. Przedstawiono też generowanych osobników na tle optimów funkcji.

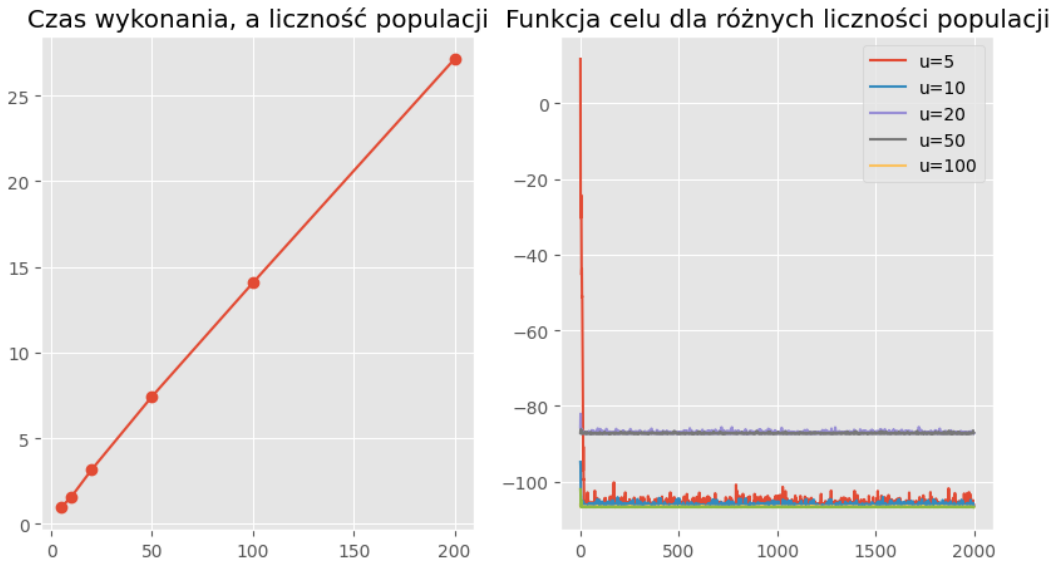


Rys 1.2 – funkcja celu w kolejnych iteracjach i rozkład osobników dla różnych sił mutacji

Z wykresów wynika, że duża siła mutacji zwiększa zdolność do eksploracji, a mała zwiększa zdolność eksploatacji. Mała siła mutacji umożliwia dokładne zbadanie minimum lokalnego, natomiast duża siła bada całą dziedzinę, ale może wystąpić problem w idealne trafienie w minimum globalne/lokalne. W tym przypadku kompromisem wydaje się być siła mutacji = *0,2*.

Czas wykonania, a liczność populacji

Na rysunku 1.3 przedstawiono czas wykonania w zależności od liczności populacji. Pokazano też wartość funkcji celu w kolejnych iteracjach. Czas wykonania programu rośnie w przybliżeniu liniowo ze wzrostem liczności populacji. W prawie wszystkich przypadkach udało się znaleźć minimum globalne, dlatego wydaje się, że przesadne zwiększanie populacji nie jest racjonalne.

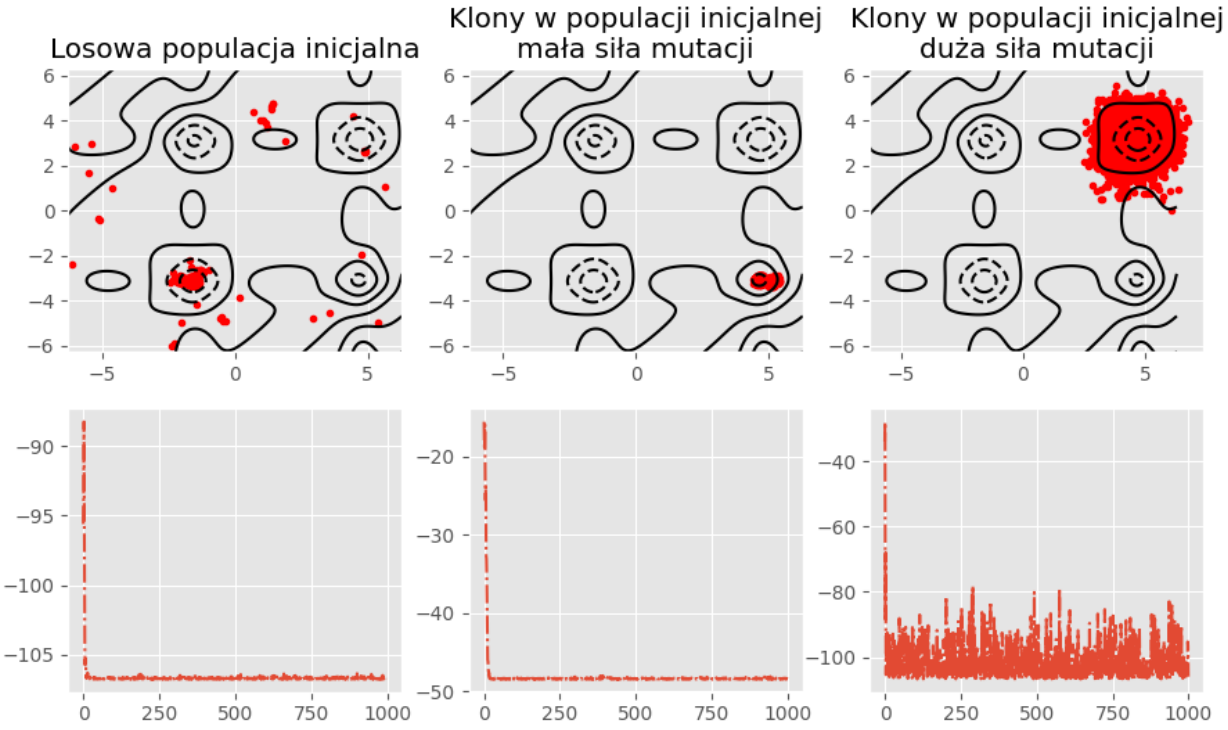


Rys 1.3 – czas wykonania w zależności od liczności populacji i wykres funkcji celu w kolejnych iteracjach

Populacja inicjalna

Na rysunku 1.4 przedstawiono rozkład generowanych osobników i wartość funkcji celu w kolejnych iteracjach dla różnych wariantów:

- Losowa populacja początkowa, mała siła mutacji
- Populacja początkowa składająca się z klonów, mała siła mutacji
- Populacja początkowa składająca się z klonów, duża siła mutacji



Rys 1.4 – rozkład generowanych osobników i wartości funkcji celu w kolejnych iteracjach dla różnych populacji inicjalnych

Dla losowej populacji inicjalnej algorytm działa normalnie. Gdy populacja inicjalna składa się z klonów jednego osobnika, to na początku dziedzina funkcji jest bardzo słabo zbadana i algorytm działa słabo, ale działa ze względu na mechanizm mutacji. Wynika z tego, że być może warto ustawić większy rozmiar populacji inicjalnej niż rozmiar populacji w dalszych iteracjach. Wtedy dziedzina funkcji zostanie dobrze zbadana. Niestety dla wielowymiarowych funkcji celu nawet bardzo duża populacja początkowa będzie miała problem, żeby równomiernie pokryć całą dziedzinę.