Algorytm genetyczny w problemie optymalizacji

Daria Bartkowiak

16 października 2025

Spis treści

1 Opis algorytmu **2**

2 Testowanie algorytmu **2**

2.1 Opis eksperymentu 3

2.2 Wyniki 4

2.3 Wnioski 5

1 Opis algorytmu

Mój algorytm zaimplementowałam jako **algorytm genetyczny**, zgodnie z szablonem prof. Biedrzyckiego prezentowanym na wykładzie (Rys. 1). Celem jego działania jest znalezienie rozwiązania zapewniającego osiągnięcie wartości optymalnej funkcji celu.

Data: $q(x), P_0, \mu, p_m, p_c, t_{max}$
Result: $\hat{x}^*, \hat{o}^*$
begin
  $t \leftarrow 0$
  $o \leftarrow$ ocena(q, P_0)
  $\hat{x}^*, \hat{o}^* \leftarrow$ znajdź najlepszego(P_0, o)
  while $t < t_{max}$ **do**
    $R \leftarrow$ reprodukcja(P_t, o, μ)
    $M \leftarrow$ krzyżowanie i mutacja(R, p_m, p_c)
    $o \leftarrow$ ocena(q, M)
    $x_t^*, o_t^* \leftarrow$ znajdź najlepszego(M, o)
    if $o_t^* < \hat{o}^*$ **then**
      $\hat{o}^* \leftarrow o_t^*$
      $\hat{x}^* \leftarrow x_t^*$
    end
    $P_{t+1} \leftarrow M$
    $t \leftarrow t + 1$
  end
end

Rysunek 1: Schemat algorytmu genetycznego zgodny ze wzorcem przedstawionym na wykładzie prof. Biedrzyckiego.

Na początku generowana jest losowa populacja chromosomów binarnych reprezentujących możliwe rozwiązania. Każdy osobnik oceniany jest za pomocą funkcji celu, która określa jego jakość.

Następnie przeprowadzana jest **selekcja metodą ruletki**, **krzyżowanie jednopunktowe** oraz **mutacja bitowa**, co umożliwia tworzenie nowych, coraz lepszych osobników. Proces ewolucji powtarzany jest przez określoną liczbę generacji (`t_max`), aż do uzyskania satysfakcjonującego wyniku. W ten sposób algorytm stopniowo doskonali populację, dążąc do znalezienia rozwiązania optymalnego.

2 Testowanie algorytmu

Mój algorytm wykorzystałam przy szukaniu rozwiązania problemu opisanego w zadaniu.

- Oczywistym rozwiązaniem trywialnym byłoby pozostawienie obu silników wyłączonych, co odpowiada osobnikowi reprezentowanemu przez wektor samych zer. W takim przypadku obiekt pozostałby w miejscu startu, czyli w odległości 350 metrów od celu. Podstawiając taki wektor do funkcji celu otrzymujemy wynik równy:

$$-350^2 = -122500 \tag{1}$$

- Górne ograniczenie wartości funkcji celu osiągane jest w sytuacji idealnej, gdy obiekt ląduje dokładnie w odległości 350 metrów od punktu startu. Wówczas różnica $d = 0$, a wartość funkcji celu wynosi:

$$-d^2 = -0^2 = 0.$$

  Zatem maksymalny możliwy wynik (wartość optymalna) funkcji celu wynosi 0.

- Przetestowałam także rozwiązanie losowe, w którym wartości sterowań zostały wygenerowane przypadkowo. Uzyskany wynik wyniósł -121996, co jest tylko nieznacznie lepsze od wartości dla rozwiązania trywialnego (-122500). Pokazuje to, że losowe sterowanie praktycznie nie przybliża obiektu do celu i stanowi bardzo słaby punkt odniesienia w porównaniu z wynikami uzyskiwanymi przez nasz algorytm.

- Po sprawdzeniu różnych konfiguracji uznałam, że dobre wyniki dają parametry: $\mu = 200$, $p_m = 0.005$, $p_c = 0.8$ oraz $t_{\max} = 100$. Testy przeprowadzone dla 10 różnych losowych ziaren (*random seed*) potwierdziły stabilność algorytmu — znaczna większość wyników była bardzo dobra.

2.1 Opis eksperymentu

Postanowiłam zbadać wpływ wartości parametru mutacji p_m na wyniki algorytmu. Na początku, przy ustalonym ziarnie losowym równym 13, przeprowadziłam testy dla zestawu wartości [0.00005, 0.0001, 0.001, 0.005, 0.01, 0.02, 0.05, 0.1].

p_m	**best_score**
0.00005	-0.000002
0.00010	-0.000009
0.00100	-0.000235
0.00500	-7.272213
0.01000	-0.011614
0.02000	-1.774386
0.05000	-426.7087
0.10000	-773.4878

Jednak rezultat działania algorytmu w dużym stopniu zależy od punktu startowego (czyli też ziarna generatora liczb pseudolosowych). Dlatego dla każdej wartości p_m uruchomiłam algorytm dziesięciokrotnie z różnymi losowymi ziarnami i następnie uśredniłam otrzymane wyniki.

2.2 Wyniki

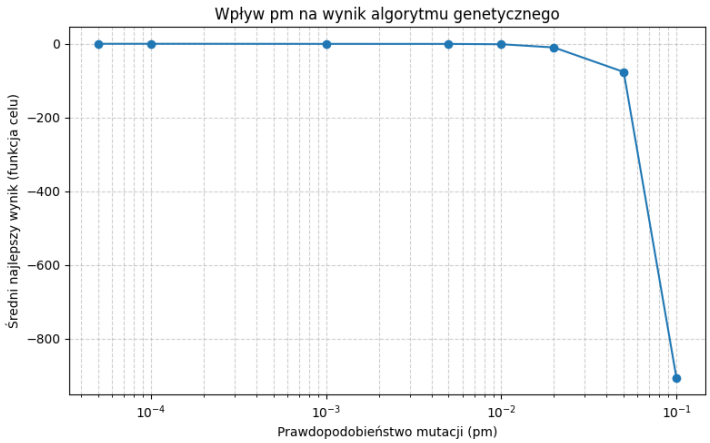
Wyniki eksperymentu dla ustawionego ziarna generatora losowego

p_m	**best_score**
0.00005	-0.000002
0.00010	-0.000009
0.00100	-0.000020
0.00500	-0.001995
0.01000	-2.740780
0.02000	-0.214603
0.05000	-947.265382
0.10000	-1903.151471

Wyniki eksperymentu dla 10 losowych wartości ziaren generatora

Tabela 1: Wyniki dla ziaren generatora: [678, 833, 852, 43, 368, 568, 902, 128, 95, 746]

p_m	**best_score**
0.00005	-0.000002
0.00010	-0.000009
0.00100	-0.000020
0.00500	-0.001995
0.01000	-2.740780
0.02000	-0.214603
0.05000	-947.265382
0.10000	-1903.151471



2.3 Wnioski

- **Małe p_m wygrywa:** Najlepsze (niemal zerowe) wartości funkcji celu uzyskano dla bardzo małych mutacji $p_m \in [10^{-5}, 10^{-4}]$.
- **Lekki wzrost = lekka strata:** Dla $p_m \approx 10^{-3}$–$5 \cdot 10^{-3}$ wyniki wciąż są dobre, ale zauważalnie gorsze niż dla 10^{-5}–10^{-4}.
- **Próg pogorszenia:** Około $p_m \approx 10^{-2}$ pojawia się wyraźny spadek jakości (średnie ujemne rzędu jednostek i dziesiątek).
- **Głowny wniosek:** Zbyt duża mutacja szkodzi! Dla $p_m \geq 0.05$ następował gwałtowny spadek (do -10^2–-10^3), co wskazuje na rozpraszanie populacji i brak zbieżności.
- **Dlaczego duże p_m szkodzi:** Wysokie prawdopodobieństwo mutacji powoduje, że potomkowie znacząco różnią się od rodziców, przez co zanika dziedziczenie dobrych cech. Selekcja nie ma wtedy czego „utrwalać", bo każda generacja zostaje silnie losowo zaburzona — algorytm de facto błądzi losowo a nie idzie w kierunku doskonalenia.