Czym będziemy zajmować się na najbliższych 2 spotkaniach?

- Optymalizacja – poszukiwanie jak najlepszego rozwiązania (łac. optimus – najlepszy).
- W praktyce często nie chodzi o najlepsze rozwiązanie, ale o lepsze od tego, które już mamy.
- Jak nie mamy żadnego rozwiązania, to zadowolimy się rozwiązaniem akceptowalnym, czyli takim, które spełnia ograniczenia narzucone przez fizykę i/lub eksperta zlecającego wykonanie zadania.
- W praktyce często nie wiemy, czy to co uzyskaliśmy w wyniku optymalizacji jest rzeczywiście najlepsze.

Optymalizacja, uczenie się maszyn, aproksymacja funkcji – wzajemne zależności

- W maszynowym uczeniu się wykorzystuje się metody optymalizacji. Przykłady: Sieci Neuronowe, SVM.
- W przypadku zadań, dla których czasochłonne jest uruchomienie modelu zjawiska stosuje się aproksymatory funkcji.

Od początku

- Dana jest metryczna **przestrzeń przeszukiwań** $\Omega = (U, |\cdot|)$, gdzie U jest zbiorem wartości a $|\cdot|$ to metryka.
- Zwykle mamy do czynienia z optymalizacją z **ograniczeniami**, a zatem pracujemy na zbiorze $D \subseteq U$.
- Dana jest **funkcja celu** $q(\mathbf{x}) : D \to \Re$.
- Zadanie optymalizacji (dla **minimalizacji**) polega na znalezieniu takiego $\mathbf{x}^* \in D$, że $\mathbf{x}^* = \arg\min_{\mathbf{x} \in D} q(\mathbf{x})$. $\mathbf{x}^*$ to **minimum globalne**
- **Minimum lokalne**: $\exists \delta > 0 \quad \forall r < \delta \quad \forall \mathbf{y} \in N_r(\mathbf{x}) \cap D \quad q(\mathbf{x}) \leq q(\mathbf{y})$ – punkt, w którego sąsiedztwie o niezerowym promieniu nie istnieje lepszy punkt.
- Funkcje mające więcej niż jedno optimum nazywa się **funkcjami wielomodalnymi**.
- **Ograniczenia kostkowe** – istnieją wektory $\mathbf{l}$, $\mathbf{u}$. Dla każdego $\mathbf{x}$ musi zachodzić: $l_i \leq x_i \leq u_i$, $i \in 1, \ldots, n$.

Rodzaje metod optymalizacji

Metody optymalizacji można podzielić na:

- Lokalne – skupiają się na pierwszym „zauważonym" ekstremum; są szybkie.
- Globalne – starają się dojść do ekstremum globalnego, są relatywnie wolne.

Ogólnie o ograniczeniach

Ograniczenia można podzielić na:

- Twarde – wynikają z ograniczeń fizycznych, lub z możliwości modelu zjawiska fizycznego. Nie mogą być złamane.
- Miękkie – mogą być chwilowo naruszane, wynik optymalizacji powinien je spełniać.

Podstawowe metody radzenia sobie z ograniczeniami

- Utrzymanie dopuszczalności rozwiązań
- Naprawa rozwiązań
- Stosowanie **funkcji kary**: $q_p(\mathbf{x}) = q(\mathbf{x}) + p(\mathbf{x})$

Rodzaje zadań

Rodzaje zadań optymalizacji:

- Zadania ciągłe – wartości zmiennych x_i należą do zbioru liczb rzeczywistych.
- Zadania dyskretne – wartości zmiennych x_i należą do zbioru dyskretnego.
  - Zadania kombinatoryczne – wartości zmiennych x_i mogą przyjmować wartość logiczną (prawda, fałsz).

Kodowanie rozwiązania

Jak zapisać rozwiązanie zadania optymalizacji w komputerze:

- Kodowanie binarne – x_i mogą przyjmować tylko wartości 0 lub 1.
- Kodowanie wyliczeniowe – x_i mogą przyjmować tylko ściśle określone wartości.
- Kodowanie rzeczywistoliczbowe – $\mathbf{x}$ jest wektorem liczb rzeczywistych.
- Kodowanie specyficzne dla problemu.

Punkt startowy

Optymalizować można coś co istnieje – zatem od czego rozpocząć optymalizację:

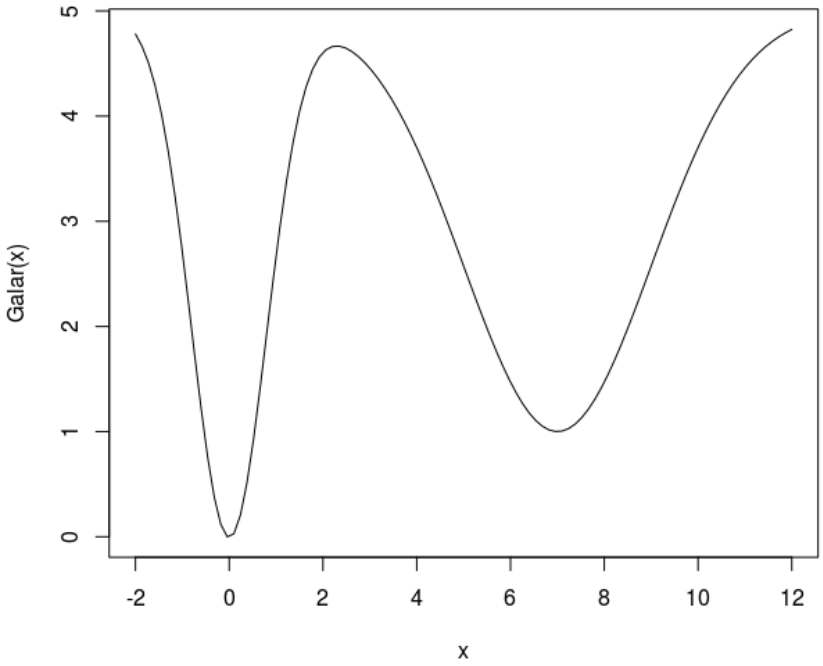
- Rozwiązanie losowe – trudno wygenerować rozwiązanie dopuszczalne przy skomplikowanych ograniczeniach.
- Rozwiązanie wyznaczone dzięki wykorzystaniu wiedzy dziedzinowej.

Punkt startowy

Punkt startowy wyznaczony dzięki wykorzystaniu wiedzy dziedzinowej:

- Może przyspieszyć optymalizację.
- Może być koniecznością w przypadku trudnych wielowymiarowych problemów.
- Może uwięzić optymalizator w ekstremum lokalnym.

Funkcja Galara



- Funkcja posiadająca 1 minimum lokalne i jedno globalne (odpowiednio dla $x = 7$ i $x = 0$).
- Co może się stać jak zaproponujemy punkt startowy $x = 6$?

Algorytmy ewolucyjne

Algorytmy ewolucyjne:

- Inspirowane ewolucją naturalną – zarówno sposób działania i nazewnictwo.
- Początkowo algorytmy z tej rodziny wykorzystywały kodowanie binarne. Algorytmy takie nazywamy **algorytmami genetycznymi**.
- Prosty algorytm genetyczny to już stary pomysł (Holland, 1975 rok; rozwój i zastosowania: De Jong i Goldberg).

Ewolucyjne nazewnictwo

- Osobnik – reprezentuje punkt w przeszukiwanej przestrzeni ($\mathbf{x}$), może zawierać dodatkowe informacje.
- Populacja – zbiór osobników przetwarzanych w każdej iteracji (P).
- Mutacja – losowe zaburzenie zmiennych x_i osobnika (**genów**), $i \in 1 \ldots n$, gdzie n to wymiarowość zadania.
- Krzyżowanie – nowy osobnik powstaje na podstawie genów istniejących osobników (najczęściej dwóch).
- Chromosom – wszystkie geny. Najczęściej osobnik ma jeden chromosom. W niektórych algorytmach osobnik ma dodatkowe, pomocnicze chromosomy.

Przykładowy problem dyskretny

- Mamy duży magazyn do wychłodzenia – jego zapełnienie zmienia się dynamicznie, ceny energii elektrycznej zmieniają się w ciągu doby.
- Sterujemy ogromnym kompresorem, który możemy włączyć lub wyłączyć.
- Naszym celem jest minimalizacja kosztu energii elektrycznej przy jednoczesnym spełnieniu ograniczeń co do temperatury w magazynie.
- Ze względu na duże stałe czasowe nie ma sensu zmieniać stanu częściej niż co pół godziny.
- Optymalizujemy stan pracy kompresora w przeciągu doby.
- Zatem punkt w przestrzeni przeszukiwań ($\mathbf{x}$), reprezentowany przez osobnika, będzie miał 24*2 bitów, gdzie 1 będzie oznaczać włączenie kompresora w danym kwancie czasu. W Pythonie można to zapisać np. tak: osobnik = np.array([1,0,1,0,0,0,1,1,1,1,1,0,1,...,1,0,0,0,1,1,1,1,1,0])

Przykładowy problem dyskretny, c.d.

- Do pracy potrzebujemy jeszcze zdefiniować funkcję celu $q(\mathbf{x})$, czyli musimy móc przypisać ocenę każdemu rozwiązaniu (główny jej składnik to suma po czasie iloczynu stanu agregatu i kosztu energii).
- Potrzebujemy również modelu magazynu, który to model odpowie nam na pytanie jak będzie zmieniała się temperatura w magazynie.

Funkcja celu w Pythonie

```
price = np.array([1, 2, 1, 3, 3, ...,3, 2, 2, 1])
lower = 1
upper = 3
PEN_MULT = 1e9

def q(x):
  cost = sum(x*price)
  temp = run_simulator( x )
  penalty = ((temp[temp>upper]-upper)**2).sum() +
      ((temp[temp<lower]-lower)**2).sum()
  penalty *= PEN_MULT
  return cost+penalty
```

Pseudokod algorytmu genetycznego

```
   Data: q(x), P_0, μ, p_m, p_c, t_max
   Result: x̂*, ô*
 1 begin
 2 |  t ← 0
 3 |  o ← ocena( q, P_0 )
 4 |  x̂*, ô* ← znajdź najlepszego( P_0, o )
 5 |  while t < t_max do
 6 |  |  R ← reprodukcja( P_t, o, μ )
 7 |  |  M ← krzyżowanie i mutacja( R, p_m, p_c )
 8 |  |  o ← ocena( q, M )
 9 |  |  x*_t, o*_t ← znajdź najlepszego( M, o )
10 |  |  if o*_t < ô* then
11 |  |  |  ô* ← o*_t
12 |  |  |  x̂* ← x*_t
13 |  |  end
14 |  |  P_{t+1} ← M
15 |  |  t ← t + 1
16 |  end
17 end
```

Parametry algorytmu genetycznego

- $q(x)$ – funkcja celu,
- P_0 – populacja początkowa,
- μ – liczba osobników w populacji,
- p_m – prawdopodobieństwo mutacji genu,
- p_c – prawdopodobieństwo krzyżowania. Dla nieużytego jeszcze osobnika z R losujemy mu parę, jak $\mathcal{U}(0,1) < p_c$ to stosujemy krzyżowanie, uzyskując 2 osobniki potomne z 2 rodziców. W przeciwnym razie do następnego kroku przechodzą 2 osobniki wybrane na rodziców. Powtarzamy do wyczerpania R.
- t_{max} – maksymalna liczba iteracji (pokoleń). Czasem mamy ograniczony budżet (FES – liczbę ewaluacji funkcji celu), wówczas $t_{max} = \lfloor FES/\mu \rfloor$.

Składniki algorytmu genetycznego

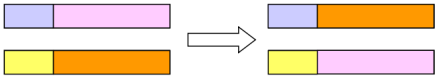
Składniki algorytmu genetycznego:

- Reprodukcja – ma za zadanie wybrać lepsze punkty z P_t z większym prawdopodobieństwem niż gorsze.
- Krzyżowanie – ma za zadanie wygenerować punkt „pośredni” pomiędzy rodzicami.
- Mutacja – ma za zadanie wygenerować punkt z otoczenia punktu mutowanego, realizowana przez negację bitów, dla których $U(0,1) < p_m$.

Klasyczny algorytm genetyczny (Holland, 1975)

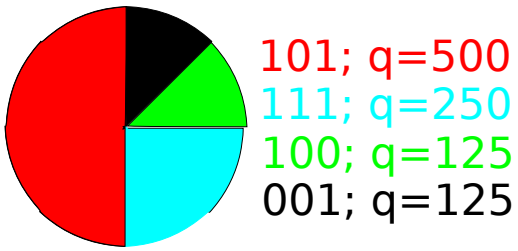
- Reprodukcja proporcjonalna (ruletkowa) – prawdopodobieństwo wyboru osobnika jest wprost proporcjonalne do wartości funkcji przystosowania: $p_s(P(t,j)) = \dfrac{q(P(t,j))}{\sum_k (q(P(t,k))}$. Wzór dotyczy maksymalizacji – zastosowanie go wprost do dowolnej funkcji q może prowadzić do problemów (np. zerowe i ujemne prawdopodobieństwa).

- Krzyżowanie jednopunktowe – wybieramy losowo punkt przecięcia chromosomu, z dwóch osobników rodzicielskich powstają dwa osobniki potomne przez prostą wymianę części chromosomów rodziców.



- Sukcesja generacyjna.
- Z klasycznym podejściem związana jest również teoria schematów uzasadniająca oczekiwania co do jego skuteczności. Opinie co do jej słuszności są podzielone.

Przykład reprodukcji



przykładowo po reprodukcji:

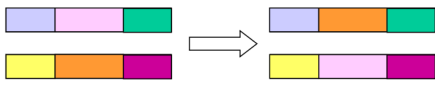
101; q=500
100; q=125
101; q=500
111; q=250

Przykładowe osobniki, z przykładową oceną, przykładowy wynik reprodukcji:

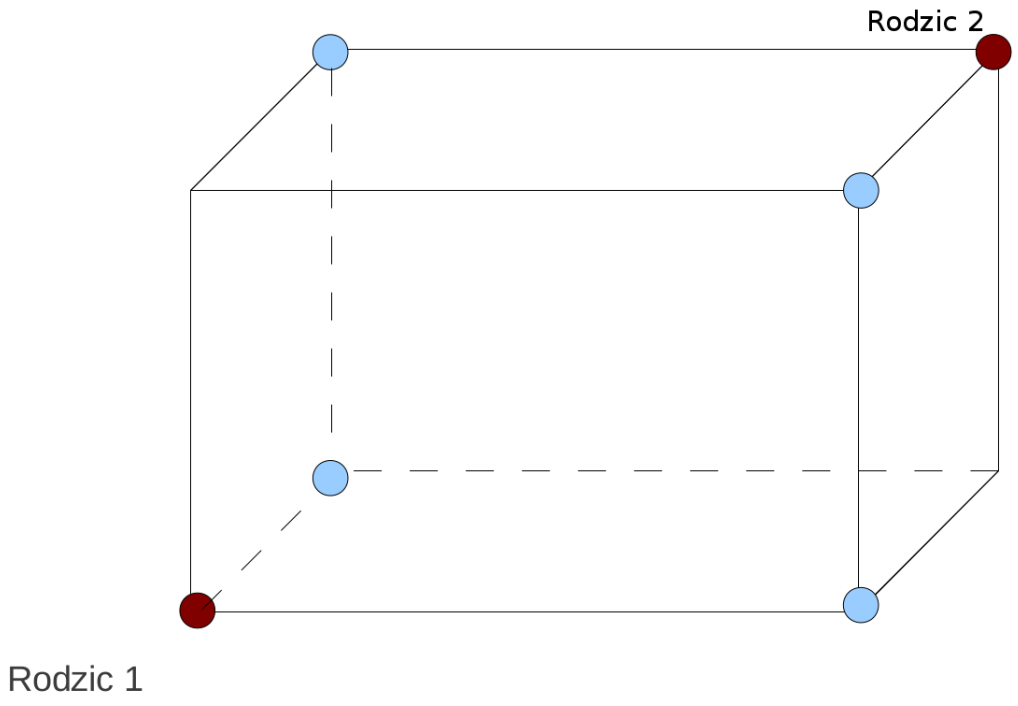
Klasyczny algorytm genetyczny – ustawienia

- Rozmiar populacji powinien być duży (znacznie większy niż rozmiar problemu).
- Stosuje się duże prawdopodobieństwo krzyżowania i bardzo małe prawdopodobieństwo mutacji.
- Uważano, że głównym czynnikiem napędowym ewolucji jest krzyżowanie, przy czym nie powinno ono niszczyć znalezionych schematów. W związku z tym krzyżowanie jednopunktowe było uważane za dobre.
- W podejściu tym mutacja jest traktowana tylko jako mechanizm pomocniczy, ułatwiający opuszczenie ekstremum lokalnego, zwiększający różnorodność populacji.

Alternatywne schematy krzyżowania



- Krzyżowanie dwupunktowe – wybieramy losowo 2 punkty przecięcia chromosomu.
- Krzyżowanie równomierne (jednorodne) – dla każdego genu losujemy, z którego rodzica będzie pochodził.
- Krzyżowanie dwupunktowe jest zwykle lepsze od jednopunktowego.
- Krzyżowanie równomierne było uważane za złe, ponieważ gubi znalezione schematy.



Rysunek 1: Krzyżowanie jednopunktowe. Nie wszystkie kombinacje genów są osiągalne.

Kodowanie rozwiązania

- Kodowanie: genotyp-fenotyp.
- Problem, którego obecnie się nie spotyka ale można spotkać analogiczne: liczba rzeczywista kodowana binarnie a mutacja, np. 7 vs 8, czyli B111 vs B1000.

Algorytm genetyczny – podsumowanie

- +/− – bogata literatura.
- +/− – dostępne wiele implementacji w wielu językach.
- +/− – możliwość konfiguracji, wiele komponentów.
- Można używać nowszych wersji jeśli problem jest naturalnie binarnie zakodowany.

Algorytm ewolucyjny

Zostawiamy **algorytm genetyczny**, ponieważ przechodzimy do problemów ciągłych. Do ich rozwiązania na początek zastosujemy **algorytm ewolucyjny**. Algorytm ewolucyjny pracuje na liczbach rzeczywistych, czyli gen jest reprezentowany przez liczbę rzeczywistą.

Przykładowy problem ciągły

Przykładowy problem, kodowanie rzeczywistoliczbowe.

- Mamy duży magazyn do wychłodzenia – ten sam, który był przykładem przy algorytmie genetycznym.
- Każda chłodziarka, włącznie z lodówką ma termostat, który próbuje utrzymać temperaturę zadaną.

- Niech tym razem temperatura zadana podlega optymalizacji.
- Ze względu na duże stałe czasowe nie ma sensu zmieniać stanu częściej niż co pół godziny.
- Optymalizujemy stanu pracy kompresora w przeciągu doby.
- Zatem osobnik będzie miał 24*2 liczb rzeczywistych.

Inne przykładowe problemy ciągłe

Inne przykładowe problemy

- Chcemy znaleźć wymiary belki stalowej aby wytrzymała obciążenie a jednocześnie była jak najtańsza.
- Chcemy wykryć czy stalowa belka pęknie, mamy pomiary jej aktualnych własności i model teoretyczny – chcemy znaleźć parametry tego modelu, tak aby jego wynik pasował do pomiarów.

Pseudokod algorytmu ewolucyjnego

Data: $q(x), P_0, \mu, \sigma, p_c, t_{max}$
Result: $\hat{x}^*, \hat{o}^*$

```
begin
    t ← 0
    o ← ocena( q, P_0 )
    x̂*, ô* ← znajdź najlepszego( P_0, o )
    while nie spełnione kryterium stopu( t, t_max, P_t, o ) do
        R ← reprodukcja( P_t, o, μ )
        M ← operacje genetyczne( R, σ, p_c )
        o_m ← ocena( q, M )
        x*_t, o*_t ← znajdź najlepszego( M, o_m )
        if o*_t < ô* then
            ô* ← o*_t
            x̂* ← x*_t
        end
        P_{t+1}, o ← sukcesja( P_t, M, o, o_m )
        t ← t + 1
    end
end
```

Składniki algorytmu ewolucyjnego

- Reprodukcja – ma za zadanie wybrać lepsze punkty z P_t z większym prawdopodobieństwem niż gorsze.
- Operatory genetyczne – to typowo krzyżowanie i mutacja, zachowujemy rozmiar populacji (tyle mutantów ile rodziców).
- Krzyżowanie – ma za zadanie wygenerować punkt „pośredni” pomiędzy rodzicami, należy używać z umiarem, tylko jak problem pozwala na sensowne zdefiniowanie tego operatora.
- Mutacja – ma za zadanie wygenerować punkt z otoczenia punktu mutowanego.
- Sukcesja (zastępowanie) – ma zdecydować które osobniki przeżyją do następnej generacji.

Rozmiar populacji

- Większość środowiska uważa, że im więcej tym lepiej.
- Zbyt dużo osobników spowalnia proces optymalizacji.
- Część uważa, że małe populacje są najlepsze.

- Klątwa wymiarowości (*curse of dimensionality*) – w miarę wzrostu liczby wymiarów (rozmiar zadania) liczba obserwacji (rozmiar populacji) potrzebnych do odpowiednio gęstego spróbowania przestrzeni przeszukiwań rośnie wykładniczo.
- Jeżeli przy ograniczeniach $< 0,1 >^n$ chcemy mieć punkty co 0,01 to w przestrzeni jednowymiarowej potrzebujemy 100 punktów (10^2). Jeżeli chcemy taką samą odległość pomiędzy punktami zachować w przestrzeni 10 wymiarowej to potrzebujemy 10^{20} punktów $(10^2)^{10}$.

Inicjacja

- W idealnym algorytmie inicjalizacja populacji ma znaczenie ze względu na szybkość dojścia do zadowalających rozwiązań.
- W praktyce część populacji początkowych może prowadzić do zadowalającego rozwiązania, a pozostałe mogą prowadzić do rozwiązań zbyt słabych (słabe optima lokalne).
- Gdy populacja zawiera podobne do siebie chromosomy, to przeszukiwanie ma charakter bardziej lokalny.
- Zwykle różnorodność populacji zmniejsza się w trakcie optymalizacji.
- Trzeba zapewnić dużą różnorodność na starcie algorytmu.

Reprodukcja

- Proporcjonalna (ruletkowa) – prawdopodobieństwo wyboru osobnika jest wprost proporcjonalne do wartości funkcji przystosowania: $p_s(P(t,j)) = \frac{q(P(t,j))}{\Sigma_k(q(P(t,k))}$.
- Rangowa – populację sortuje się ze względu na funkcję oceny. Pozycja osobnika to jego ranga (najlepszy $r = 1$). Prawdopodobieństwo wyboru osobnika: $p_s(P(t,j)) = a + k\left(1 - \frac{j}{\mu}\right)$. Istnieją też inne pomysły na p_s

Reprodukcja

- Progowa – $\mu(1-\rho)$ najgorszych osobników jest wyrzucane. Pozostałe osobniki mają jednakową szansę na wybór: $p_s(P(t,j)) = \frac{1}{\rho\mu}$. ρ nazywany jest wskaźnikiem nacisku selektywnego, jest parametrem algorytmu.
- Turniejowa – z jednakowym prawdopodobieństwem, z powtórzeniami wybieramy grupę osobników (typowo stosuje się turnieje 2 osobnikowe), następnie najlepszy z tej grupy jest zwracany jako zwycięzca turnieju. Turniej powtarza się μ krotnie, ponieważ potrzebujemy μ osobników po reprodukcji.

O reprodukcji turniejowej

- Dobre zdolności eksploracyjne.
- Łatwa w implementacji.