Politechnika Wrocławska

# Optymalizacja systemów

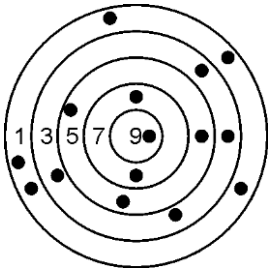
Wykład 7. Algorytmy Inspirowane Naturą

# Agenda

- Wprowadzenie
- Algorytm Wspinaczkowy - Stochastyczny
- Symulowane Wyżarzanie
- Algorytm Mrówkowy
- Algorytm Pszczeli
- Algorytm Roju Cząstek

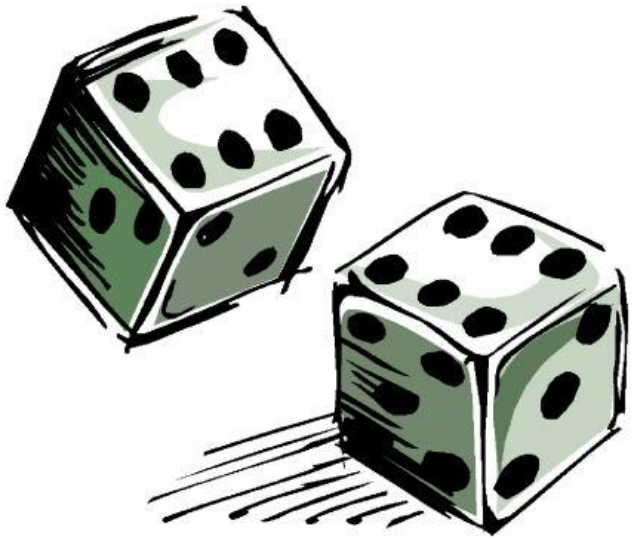
# Wprowadzenie

- Heurystyka to odgadywanie rozwiązania.
- Stosowana tam, gdzie pełne przeszukiwanie rozwiązań jest niemożliwe / nieopłacalne.
- Nie zawsze zwraca optymalne rozwiązanie.
- Może być wejściem dla innego algorytmu.
- Sami stosujemy ją na co dzień.



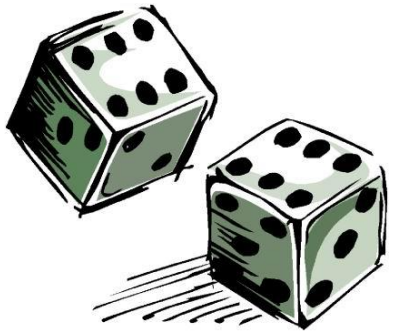
# Wprowadzenie - Algorytm losowy

1. Zaczyna w losowym punkcie.
2. Losuje nowy punkt z rozkładem jednostajnym, na całej przestrzeni.
3. Jeśli nowy jest lepszy od obecnego kandydata – zastępuje.



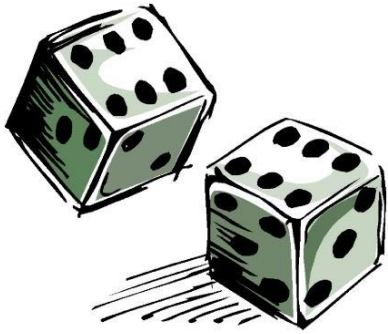
# Wprowadzenie - Algorytm losowy

```
Input: NumIterations, ProblemSize, SearchSpace
Output: Best
Best ← ∅
For (iter_i ∈ NumIterations)
  candidate_i ← RandomSolution(ProblemSize, SearchSpace)
  If (Cost(candidate_i) < Cost(Best))
    Best ← candidate_i
  End
End
Return (Best)
```



# Algorytm losowy
# Zalety i wady

- ✓ Zwraca stosunkowo dobre rozwiązanie w krótkim czasie.
- ✓ Dobry do wstępnego przeszukania przestrzeni rozwiązań.
- ! Losowy wybór może być problematyczny, w zależności od dziedziny.
- ! Nie jest skalowalny (rozmiar/wymiary).



# Algorytm Wspinaczkowy Stochastyczny

1. Zaczyna w losowym punkcie.
2. Losuje sąsiadującą wartość.

(Wariant: prawdopodobieństwo wyboru sąsiada zależy od wartości).

3. Porównuje ją z obecną i akceptuje, jeśli jest lepsza.

# Algorytm Wspinaczkowy Stochastyczny

**Input**: $Iter_{max}$, ProblemSize
**Output**: Current
Current $\longleftarrow$ RandomSolution(ProblemSize)
**For** $(iter_i \in Iter_{max})$
  Candidate $\longleftarrow$ RandomNeighbor(Current)
  **If** (Cost(Candidate) $\geq$ Cost(Current))
    Current $\longleftarrow$ Candidate
  **End**
**End**
**Return** (Current)

# Algorytm Wspinaczkowy Stochastyczny – zalety i wady

- ❗ Podatny na lokalne optima.
- ❗ Optymalny jedynie dla funkcji wypukłych.
- ☑ W celu naprawy może być uruchamiany wielokrotnie, w sposób zrównoleglony.

# Symulowane wyżarzanie

1. Losowo wybiera sąsiadującą wartość.

2. Zmniejsza temperaturę (symuluje upływ czasu).

3. Gdy wylosowana wartość jest lepsza, zastępuje aktualną.

4. Jeśli nie jest lepsza oblicza P(e,e',T)= $\exp(\frac{e-e'}{T})$, gdzie:

e - energia systemu w obecnym stanie (*)

e' - energia systemu w stanie wylosowanym

T - temperatura

* energia systemu = wartość funkcji celu

# Symulowane wyżarzanie

…

5. Obecna wartość jest zastępowana kandydującą, jeśli P(e,e’,T) > x, gdzie:

x- liczba losowa z przedziału (0,1)

# Symulowane wyżarzanie

**Input:** ProblemSize, $iterations_{max}$, $temp_{max}$
**Output:** $S_{best}$
$S_{current} \leftarrow$ CreateInitialSolution(ProblemSize)
$S_{best} \leftarrow S_{current}$
**For** ($i = 1$ **To** $iterations_{max}$)
  $S_i \leftarrow$ CreateNeighborSolution($S_{current}$)
  $temp_{curr} \leftarrow$ CalculateTemperature($i, temp_{max}$)
  **If** (Cost($S_i$) $\leq$ Cost($S_{current}$))
    $S_{current} \leftarrow S_i$
    **If** (Cost($S_i$) $\leq$ Cost($S_{best}$))
      $S_{best} \leftarrow S_i$
    **End**
  **ElseIf** (Exp($\frac{CostS_{current} - CostS_i}{temp_{curr}}$) > Rand())
    $S_{current} \leftarrow S_i$
  **End**
**End**
**Return** ($S_{best}$)

# Symulowane wyżarzanie - informacje

- Przy współczynniku T = 0 algorytm staje się zachłannym.
- Początkowo chętniej akceptuje gorsze wartości.
- Wraz z wygaszaniem T->0 algorytm staje się wymagający, częściej akceptuje lepsze wartości.
- Opcjonalnie można resetować algorytm, by cyklicznie powracał do aktualnie najlepszego punktu.
- Zakres sąsiadujących próbek warto zmniejszać z czasem.

# Symulowane wyżarzanie - zalety i wady

- ✓ Istnieje wariant algorytmu dla optymalizacji funkcji ciągłych.
- ✓ Istnieje wiele ulepszeń m.in. zrównoleglone wyżarzanie lub resetowanie temperatury z zapamiętaniem najlepszego rozwiązania.

- ! W najgorszym przypadku algorytm gorszy niż pełne przeszukanie.
- ! Problem z oszacowaniem okresu wyżarzania T.

# Algorytm mrówkowy

1. Mrówki przeszukują przestrzeń losowo.
2. Jeśli natrafiają na „pożywienie”, tj. rozwiązanie – wracają do punktu startu zostawiając ślad feromonu.
3. Natrafiając na ślad feromonu podążają wyznaczoną ścieżką tym samym zwiększając ilość feromonu.
4. Feromon z czasem ulatnia się. Długie dystanse szybciej tracą intensywność feromonu.

# Algorytm mrówkowy

$$P_{i,j} \leftarrow \frac{\tau_{i,j}^{\alpha} \times \eta_{i,j}^{\beta}}{\sum_{k=1}^{c} \tau_{i,k}^{\alpha} \times \eta_{i,k}^{\beta}}$$

,gdzie:

$\tau_{i,j}^{\alpha}$ - wartość feromonu dla ścieżki i - j

$\alpha$ - współczynnik wpływu wartości feromonu ~ 1

$\eta_{i,j}^{\beta}$ - np. 1 / długość ścieżki (dla problemu komiwojażera)

$\beta$ - współczynnik heurystyki $\in (2\,,\ 5)$

C - rozmiar zbioru rozwiązań

# Algorytm mrówkowy

Zanikanie ilości feromonu na ścieżkach:

Aktualizacja lokalna:

$\tau_{i,j} \leftarrow (1-\sigma) \times \tau_{i,j} + \sigma \times \tau^0_{i,j}$ , gdzie:

$\sigma$ - współczynnik historyczny ~ 0.1

$\tau^0_{i,j}$ - poprzednia wartość feromonu

Aktualizacja globalna:

$\tau_{i,j} \leftarrow (1-\rho) \times \tau_{i,j} + \rho \times \Delta\tau i,j$

$\rho$ - współczynnik zanikania ~ 0.1

$\Delta\tau i,j$ - nagroda za bycie częścią najlepszej ścieżki globalnej ( =0 dla pozostałych ścieżek)

# Algorytm mrówkowy

**Input**: ProblemSize, $Population_{size}, m, \rho, \beta, \sigma, q0$
**Output**: $P_{best}$
$P_{best} \leftarrow$ CreateHeuristicSolution(ProblemSize)
$Pbest_{cost} \leftarrow$ Cost($S_h$)
$Pheromone_{init} \leftarrow \frac{1.0}{ProblemSize \times Pbest_{cost}}$
Pheromone $\leftarrow$ InitializePheromone($Pheromone_{init}$)
**While** ($\neg$StopCondition())
  **For** ($i = 1$ **To** $m$)
    $S_i \leftarrow$ ConstructSolution(Pheromone, ProblemSize, $\beta, q0$)
    $Si_{cost} \leftarrow$ Cost($S_i$)
    **If** ($Si_{cost} \leq Pbest_{cost}$)
      $Pbest_{cost} \leftarrow Si_{cost}$
      $P_{best} \leftarrow S_i$
    **End**
    LocalUpdateAndDecayPheromone(Pheromone, $S_i, Si_{cost}, \sigma$)
  **End**
  GlobalUpdateAndDecayPheromone(Pheromone, $P_{best}, Pbest_{cost}, \rho$)
**End**
**Return** ($P_{best}$)

# Algorytm mrówkowy - zalety i wady

Stosowany do rozwiązywania problemów plecakowych, grafowych, wykrywania krawędzi na obrazie.

❗ Duża liczba parametrów, które są zależne od problemu.

✅ Dla problemu komiwojażera lepszy niż alg. genetyczny, ewolucyjny czy symulowane wyżarzanie[1].

[1] https://www.assembla.com/spaces/easytsp/documents/duKXyk6n8r379GeJe5cbLr/download/duKXyk6n8r379GeJe5cbLr

# Algorytm pszczeli

1. Zwiadowcy wysyłani są w losowe miejsca.
2. W okolice znalezionych, dobrych wartości wysyłanych jest więcej osobników, w celu zbadania lokalnego optimum.
3. W każdej iteracji wysyłani są zwiadowcy w celu dalszej eksploracji przestrzeni rozwiązań.

# Algorytm pszczeli - I

Wyróżniamy grupy:

- *Zwiadowców*
- *Obserwatorów*
- *Zatrudnionych*

# Algorytm pszczeli - II

***Zwiadowcy*** – przeszukują rozwiązania w sposób losowy

***Zatrudnieni*** – przynoszą informacje o jakości rozwiązań do których są obecnie przypisani.

***Obserwatorzy*** – wybierają następne miejsce na podstawie informacji od *zatrudnionych*.

# Algorytm pszczeli - III

*Wyznaczanie źródeł pożywienia:*

$x_{ij} = x_{\min j} + rand[0,1](x_{\max j} - x_{\min j})$ , gdzie

$X_{ij}$ – źródło pożywienia

i = 1,…,P (P – liczba źródeł pożywienia)

j = 1,…,D (D – rozmiar przestrzeni rozwiązań)

# Algorytm pszczeli - IV

*Poszukiwanie pożywienia (pszczoły pracujące):*

$x'_{ij} = x_{ij} + rand[-1,1](x_{ij} - x_{kj}), i=1,\ldots,P,\ k \neq i$

, gdzie:

$k \in \{1, 2, \ldots, P\}$ oraz $j \in \{1, 2, \ldots, D\}$

są dwoma losowo wyznaczonymi indeksami

Jeśli $F(x_i') > F(x_i)$ to i-ta pszczoła zmienia źródło pożywienia

# Algorytm pszczeli - V

Poszukiwanie pożywienia (obserwatorzy):

$$p_i = \frac{F(x_i)}{\sum_{i=1}^{P} F(x_i)}, \qquad i = 1, \ldots, P$$

Obserwatorzy wybierają swój cel na podstawie informacji od pszczół pracujących, a następnie przeprowadzają lokalne przeszukiwanie w jego sąsiedztwie.

Analogicznie do symulowanego wyżarzania- sąsiedztwo jest zawężane wraz z czasem działania algorytmu.

# Algorytm pszczeli

**Input:** ProblemSize, $iterations_{max}$, $temp_{max}$
**Output:** $S_{best}$
$S_{current} \leftarrow$ CreateInitialSolution(ProblemSize)
$S_{best} \leftarrow S_{current}$
**For** ($i = 1$ **To** $iterations_{max}$)
  $S_i \leftarrow$ CreateNeighborSolution($S_{current}$)
  $temp_{curr} \leftarrow$ CalculateTemperature($i$, $temp_{max}$)
  **If** (Cost($S_i$) $\leq$ Cost($S_{current}$))
    $S_{current} \leftarrow S_i$
    **If** (Cost($S_i$) $\leq$ Cost($S_{best}$))
      $S_{best} \leftarrow S_i$
    **End**
  **ElseIf** (Exp($\frac{CostS_{current} - CostS_i}{temp_{curr}}$) > Rand())
    $S_{current} \leftarrow S_i$
  **End**
**End**
**Return** ($S_{best}$)

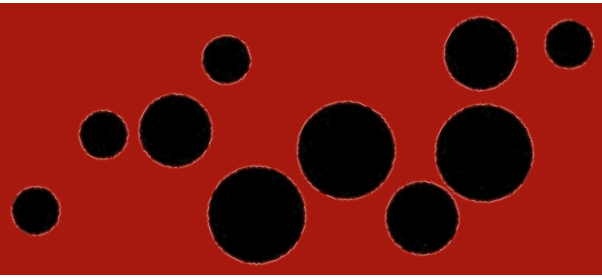
# Algorytm pszczeli

Stosowany w Data Miningu - Analiza Skupień (Clustering Methods), optymalizacji sieci neuronowych oraz kwantyzacji wektorowej.

- ✓ Szybszy niż algorytmy genetyczny czy mrówkowy dla wielu wymiarów.

- ! Stworzony dla funkcji ciągłych
- ! Podobieństwo do zrównoleglonego algorytmu losowego – ryzyko wielokrotnego sprawdzania tego samego rozwiązania.

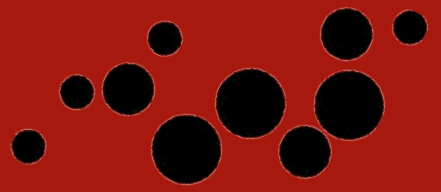
# Algorytm Roju Cząstek



Celem algorytmu jest zgromadzenie wszystkich cząstek wokół optimum wielowymiarowej hiperprzestrzeni.

- Cząsteczkom przypisuje się losowe położenie i niską prędkość początkową.
- Symulowany jest ruch cząsteczek na podstawie jej prędkości, położenia, najlepszego znanego rozwiązania oraz optimum globalnego.
- Cząsteczki zbiegają do optimum/optimów.

# Algorytm Roju Cząstek



Aktualizacja prędkości cząsteczki:

$$v_i(t+1) = v_i(t) + \left(c_1 \times rand() \times (p_i^{best} - p_i(t))\right) + \left(c_2 \times rand() \times (p_{gbest} - p_i(t))\right)$$

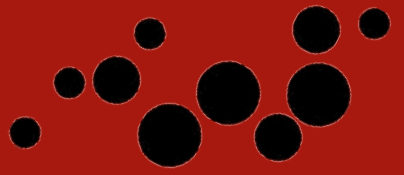
Aktualizacja położenia cząsteczki:

$$p_i(t+1) = p_i(t) + v_i(t)$$

, gdzie:

v(t) – aktualna prędkość
$p_i(t)$ – aktualne położenie i-tej cząsteczki
$P_i^{best}$ – najlepsze rozwiązanie znane i-tej cząsteczce
$P_{gbest}$ – najlepsze rozwiązane globalne
rand() – wartość losowa z przedziału (0,1)
$c_1$, $c_2$ – współczynniki wag odpowiednio dla lokalnego i globalnego optimum (wartości z przedziału (0,4), zwykle = 2).

# Algorytm Roju Cząstek



**Input:** ProblemSize, $Population_{size}$
**Output:** $P_{g\_best}$
Population $\leftarrow \emptyset$
$P_{g\_best} \leftarrow \emptyset$
**For** ($i = 1$ **To** $Population_{size}$)
  $P_{velocity} \leftarrow$ RandomVelocity()
  $P_{position} \leftarrow$ RandomPosition($Population_{size}$)
  $P_{p\_best} \leftarrow P_{position}$
  **If** (Cost($P_{p\_best}$) $\leq$ Cost($P_{g\_best}$))
    $P_{g\_best} \leftarrow P_{p\_best}$
  **End**
**End**
**While** ($\neg$StopCondition())
  **For** ($P \in$ Population)
    $P_{velocity} \leftarrow$ UpdateVelocity($P_{velocity}, P_{g\_best}, P_{p\_best}$)
    $P_{position} \leftarrow$ UpdatePosition($P_{position}, P_{velocity}$)
    **If** (Cost($P_{position}$) $\leq$ Cost($P_{p\_best}$))
      $P_{p\_best} \leftarrow P_{position}$
      **If** (Cost($P_{p\_best}$) $\leq$ Cost($P_{g\_best}$))
        $P_{g\_best} \leftarrow P_{p\_best}$
      **End**
    **End**
  **End**
**End**
**Return** ($P_{g\_best}$)

# Algorytm Roju Cząstek - informacje

- Liczba cząstek zazwyczaj wynosi 20-40
- Należy ustawić limit prędkości cząsteczki (% względem rozmiaru przestrzeni)
- Należy ustalić czy cząsteczki mogą wybiegać poza przestrzeń rozwiązań oraz ustalić kary, sposób powrotu (nowe położenie, kierunek, prędkość), ewentualnie zastosować pętle.
- Można uwzględnić zasadę pędu (niechęci do zmiany wektora prędkości).

# Algorytm Roju Cząstek

Stosowany z powodzeniem w sieciach rozproszonych, przetwarzaniu obrazu i wideo, sterowaniu czy dziedzinie optymalizacji wielokryterialnej.

- ✓ Algorytm łatwo adaptuje się do różnych dziedzin
- ✓ Mała liczba parametrów
- ✓ Może być uruchamiany w sposób zrównoleglony (Multi-Swarm Optimization)

- ⚠ Aplikacja dla dyskretnej dziedziny wymaga mapowania na liczby rzeczywiste

- ❗ Możliwe przedwczesne zbieganie cząstek do lokalnych optimów

# Bibliografia


- *Clever Algorithms: Nature-Inspired Programming Recipes*, Jason Brownlee
- *Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning , Shumeet Baluja,* School of Computer Science Carnegie Mellon University Pittsburgh,1994
- The Bees Algorithm – A Novel Tool for Complex Optimisation Problems, D.T. Pham, A. Ghanbarzadeh, E. Koç et. al, Cardiff University, 2006
- Zastosowanie Algorytmów Rojowych do Optymalizacji Parametrów w Modelach Układów Regulacji, Mirosław Tomera, Zeszyty Naukowe Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej Nr 46, 2015
- Automatic Tuning of a Retina Model for a Cortical Visual Neuroprosthesis Using a Multi-Objective Optimization Genetic Algorithm, Antonio Martínez-Álvarez, Rubén Crespo-Cano, Ariadna Díaz-Tahoces et. al., International Journal of Neural Systems 26/7, 2016