



**Politechnika Krakowska**  
Wydział Inżynierii  
Materiałowej i Fizyki



MODELOWANIE KOMPUTEROWE

---

# **Symulacja wyżarzania – Metoda Monte Carlo**

---

*Autorzy:*  
BUDZIŁO MARIA,  
KUBAŃSKI MARCIN,  
STYPUŁA HIACYNTA

25.01.2022

## 1. Wstęp. Opis problemu

**Wyżarzanie** to proces chłodzenia stopionej substancji, którego celem jest skondensowanie materii w krystaliczne ciało stałe. Wyżarzanie może być traktowane jako proces optymalizacji. Konfiguracja układu podczas wyżarzania jest określona przez zbiór położeń atomów  $r_i$ . Konfiguracja układu jest ważona przez jej współczynnik prawdopodobieństwa Boltzmann'a:

$$\exp(-E(r_i)/kT),$$

gdzie  $E(r_i)$  jest energią konfiguracji,  $k$  jest stałą Boltzmann'a, a  $T$  jest temperaturą. Kiedy substancja jest poddawana wyżarzaniu, jest ona utrzymywana w każdej temperaturze przez czas wystarczająco długi, aby osiągnąć równowagę termiczną. Problemem, z którym się zetknięto było osiągnięcie procedury symulacji wyżarzania.

## 2. Kryterium Metropolis

Technika iteracyjnego udoskonalania dla optymalizacji kombinatorycznej może być porównana do szybkiego hartowania stopionych metali. Podczas szybkiego hartowania stopionej substancji energia jest gwałtownie odbierana z układu poprzez kontakt z potężnym zimnym substratem. W wyniku gwałtownego schładzania powstają stany metastabilne układu; w metalurgii w wyniku gwałtownego schładzania otrzymuje się substancję szklistą, a nie krystaliczne ciało stałe. Analogia pomiędzy iteracyjnym udoskonalaniem i szybkim chłodzeniem metali wynika z faktu, że iteracyjne usprawnianie i szybkie chłodzenie metali akceptuje tylko te konfiguracje systemu, które zmniejszają funkcję kondycji. W procesie wyżarzania (powol-

nego chłodzenia) nowa konfiguracja układu, która nie poprawia funkcji kosztu, jest akceptowana na podstawie współczynnika prawdopodobieństwa Boltzmanna tej konfiguracji. To kryterium akceptacji nowego stanu systemu nazywane jest **kryterium Metropolis**.

### **3. Rozwiązanie problemu zbyt skrajnych temperatur**

Jeśli temperatura początkowa jest zbyt niska, proces bardzo szybko ulega zahamowaniu i znaleźć można tylko lokalne optimum. Jeśli temperatura początkowa jest zbyt wysoka, proces przebiega bardzo wolno. Do przeszukiwania używane jest tylko jedno rozwiązanie, co zwiększa prawdopodobieństwo utknięcia rozwiązania w lokalnym optimum. Zmiana temperatury oparta jest na zewnętrznej procedurze, która nie jest związana z aktualną jakością rozwiązania, czyli tempo zmiany temperatury jest niezależne od jakości rozwiązania. Problemy te mogą być rozwiązane poprzez zastosowanie zbioru zamiast pojedynczego rozwiązania. Mechanizm wyżarzania może być również sprzężony z jakością bieżącego rozwiązania poprzez uwrażliwienie szybkości zmiany temperatury na jakość rozwiązania.

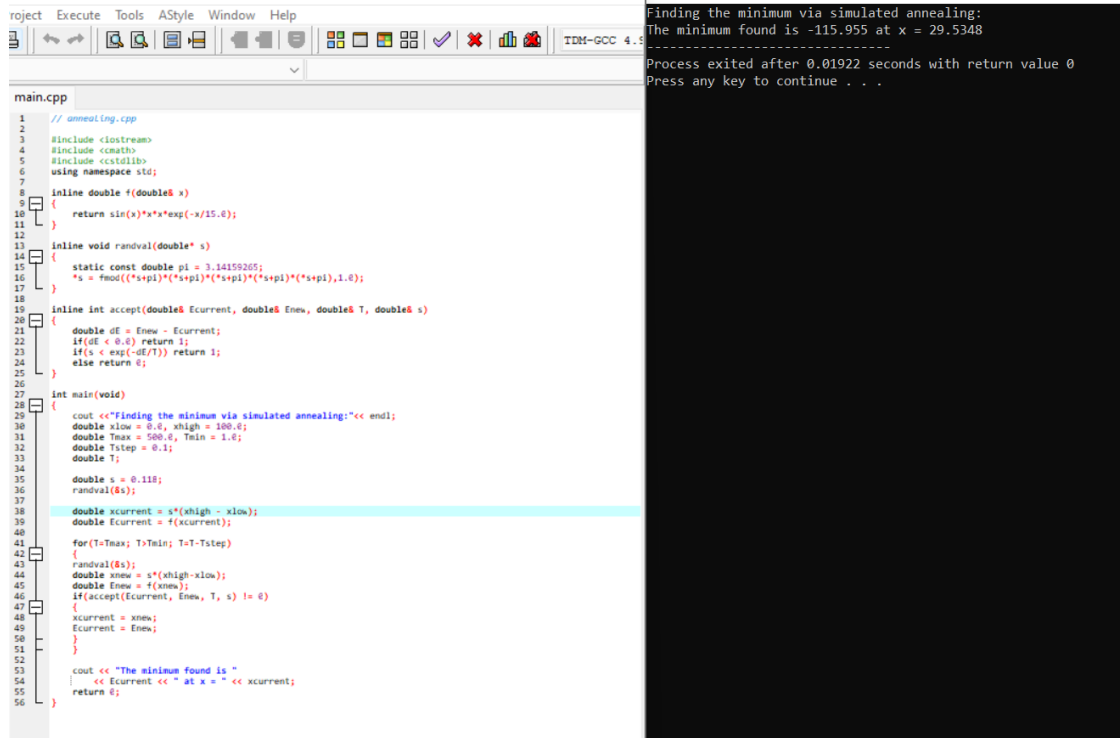
### **4. Algorytm bazowy**

Poniżej przedstawiona jest procedura symulowanego wyżarzania. Symulowane wyżarzanie polega w istocie na powtarzaniu procedury Metropolis dla różnych temperatur. Temperatura jest stopniowo obniżana w każdej iteracji algorytmu symulowanego wyżarzania.

### **Procedura symulowanego wyżarzania:**

```
początek  
t ← 0  
ustawiamy temperaturę T  
wybieramy losowo aktualny ciąg  $v_c$   
oceniamy sprawność  $v_c$   
powtarzamy  
powtarzamy  
wybieramy nowy ciąg  $v_n$   
w pobliżu  $v_c$   
poprzez odwrócenie pojedynczego bitu  $v_c$   
jeśli  $f(v_c) < f(v_n)$   
wtedy  $v_c \leftarrow v_n$   
inaczej, jeśli losowo[0,1] <  $\exp((f(v_n) - f(v_c))/T)$   
wtedy  $v_c \leftarrow v_n$   
dopóki (warunek zakończenia)  
T ← g(T,t)  
t ← t+ 1  
aż do (kryterium zatrzymania)  
koniec
```

## 5. Rozwiązanie problemu. Implementacja kodu



The image shows a screenshot of a C++ IDE with a code editor on the left and a terminal window on the right. The code editor displays the implementation of a simulated annealing algorithm in a file named `main.cpp`. The code includes headers for `iostream`, `cmath`, and `cstdlib`, and uses the `std` namespace. It defines a function `f` for the objective function, a `randval` function for generating random values, and an `accept` function for the Metropolis-Hastings acceptance criterion. The `main` function sets up the algorithm parameters and runs the simulated annealing process.

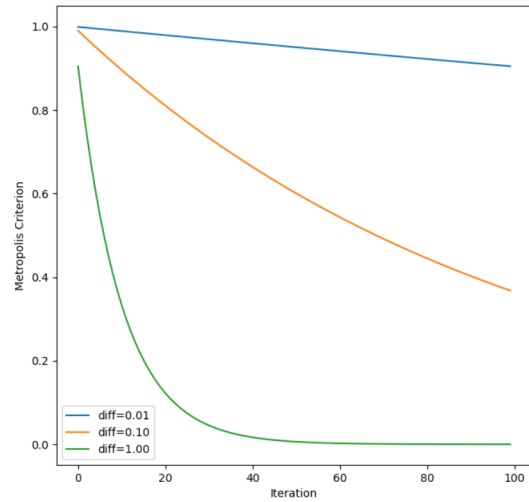
```
1 // annealing.cpp
2
3 #include <iostream>
4 #include <cmath>
5 #include <cstdlib>
6 using namespace std;
7
8 inline double f(double& x)
9 {
10     return sin(x)*x*exp(-x/15.0);
11 }
12
13 inline void randval(double* s)
14 {
15     static const double pi = 3.14159265;
16     *s = fmod((*(s+pi)*(*(s+pi)*(*(s+pi)*(*(s+pi),1.0);
17 }
18
19 inline int accept(double& Ecurrent, double& Enew, double& T, double& s)
20 {
21     double dE = Enew - Ecurrent;
22     if(dE < 0.0) return 1;
23     if(s < exp(-dE/T)) return 1;
24     else return 0;
25 }
26
27 int main(void)
28 {
29     cout << "Finding the minimum via simulated annealing:" << endl;
30     double xlow = 0.0, xhigh = 100.0;
31     double Tmax = 500.0, Tmin = 1.0;
32     double Tstep = 0.1;
33     double T;
34
35     double s = 0.118;
36     randval(&s);
37
38     double xcurrent = s*(xhigh - xlow);
39     double Ecurrent = f(xcurrent);
40
41     for(T=Tmax; T>Tmin; T=T-Tstep)
42     {
43         randval(&s);
44         double xnew = s*(xhigh-xlow);
45         double Enew = f(xnew);
46         if(accept(Ecurrent, Enew, T, s) != 0)
47         {
48             xcurrent = xnew;
49             Ecurrent = Enew;
50         }
51     }
52
53     cout << "The minimum found is "
54          << Ecurrent << " at x = " << xcurrent;
55     return 0;
56 }
```

The terminal window on the right shows the output of the program:

```
Finding the minimum via simulated annealing:
The minimum found is -115.955 at x = 29.5348
-----
Process exited after 0.01922 seconds with return value 0
Press any key to continue . . .
```

## 6. Porównanie z innymi rozwiązaniami

```
1 # explore metropolis acceptance criterion for simulated annealing
2
3 from math import exp
4 from matplotlib import pyplot
5 # total iterations of algorithm
6 iterations = 100
7 # initial temperature
8 initial_temp = 10
9 # array of iterations from 0 to iterations - 1
10 iterations = [i for i in range(iterations)]
11 # temperatures for each iterations
12 temperatures = [initial_temp/float(i + 1) for i in iterations]
13 # metropolis acceptance criterion
14 differences = [0.01, 0.1, 1.0]
15
16 for d in differences:
17     metropolis = [exp(-d/t) for t in temperatures]
18     # plot iterations vs metropolis
19     label = 'diff=%2F' % d
20     pyplot.plot(iterations, metropolis, label=label)
21
22 # finalize plot
23 pyplot.xlabel('Iteration')
24 pyplot.ylabel('Metropolis Criterion')
25 pyplot.legend()
26 pyplot.show()
```



## 7. Bibliografia

Materiały udostępnione w pliku „GeneticAlgorithms.pdf”