

Symulacja wyżarzania

Metoda Monte Carlo

Maria Budziło, Marcin Kubański, Hiacynta Stypuła

25.01.2022



Politechnika Krakowska
Wydział Inżynierii
Materiałowej i Fizyki



Plan prezentacji

- Wstęp. Opis problemu
- Kryterium Metropolis
- Rozwiązanie problemu skrajnych temperatur
- Algorytm bazowy
- Rozwiązanie problemu. Implementacja kodu
- Porównanie z innymi rozwiązaniami

Wyżarzanie jest procesem chłodzenia stopionej substancji. Celem wyżarzania jest skondensowanie materii w krystaliczne ciało stałe.

Wyżarzanie można potraktować jako proces optymalizacji.

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

$E(r_i)$ - energia konfiguracji

k - stała Boltzmannna

T - temperatura

Zadaniem będzie osiągnięcie procedury symulacji wyżarzania.

Technika iteracyjnego udoskonalania dla optymalizacji kombinatorycznej \sim szybkie hartowanie stopionych metali

Podczas wyżarzania, nowa konfiguracja układu jest akceptowana na podstawie współczynnika prawdopodobieństwa Boltzmana tej konfiguracji. To kryterium akceptacji nowego stanu systemu nazywane jest **kryterium Metropolis**.

Dobrym pomysłem na uniknięcie wielu problemów związanych ze znalezieniem rozwiązania jest zastosowanie zbioru zamiast pojedynczego rozwiązania.

Poprzez uwrażliwienie szybkości zmiany temperatury na jakość rozwiązania, mechanizm wyżarzania może być sprzężony z jakością aktualnego rozwiązania.

Symulacja wyżarzania polega w istocie na powtarzaniu procedury Metropolis dla różnych temperatur.

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
```

Rozwiązanie problemu. Implementacja kodu

```
project Execute Tools AStyle Window Help
IDM-GCC 4.3.2

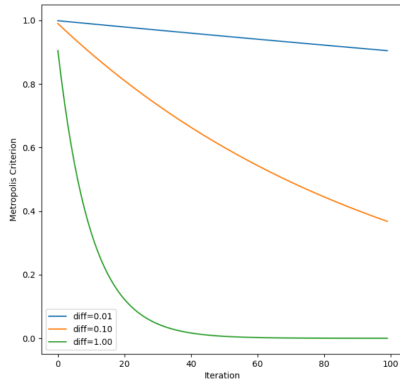
main.cpp
1 // annealing.cpp
2
3 #include <iostream>
4 #include <cmath>
5 #include <cstdlib>
6 using namespace std;
7
8 inline double f(double x)
9 {
10     return sin(x)*x*exp(-x/15.0);
11 }
12
13 inline void randval(double* s)
14 {
15     static const double pi = 3.14159265;
16     *s = fmod((*s+pi)*(*s+pi)*(*s+pi)*(*s+pi),1.0);
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 }
```

```
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 . . .
```



Porównanie z innymi rozwiązaniami

```
wyższy
1
2 # explore metropolis acceptance criterion for simulated annealing
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 iteration
12 temperatures = [initial_temp/float(i + 1) for i in iterations]
13 # metropolis acceptance criterion
14 differences = [0.01, 0.1, 1.0]
15 for d in differences:
16     metropolis = [exp(-d/t) for t in temperatures]
17     # plot iterations vs metropolis
18     label = 'diff=%2f' % d
19     pyplot.plot(iterations, metropolis, label=label)
20 # finalize plot
21 pyplot.xlabel('Iteration')
22 pyplot.ylabel('Metropolis Criterion')
23 pyplot.legend()
24 pyplot.show()
```





Materiały udostępnione w pliku „GeneticAlgorithms.pdf”



[https://machinelearningmastery.com/
simulated-annealing-from-scratch-in-python/](https://machinelearningmastery.com/simulated-annealing-from-scratch-in-python/)

Dziękujemy za uwagę!