Wstęp do inteligencji komputerowej – zajęcia nr 4

Jarosław Stańczak
WSISiZ

„Liczby losowe” w zagadnieniach sztucznej inteligencji

„Liczby losowe”

Zamiast wstępu cytat z pracy Johna von Neumanna z 1951 r.
(tłum. R. Wieczorkowski, R. Zieliński) :

„Jawnie grzeszy, kto opowiada o arytmetycznych procedurach generowania liczb losowych. Nie ma bowiem, jak to już nieraz mówiono, czegoś takiego jak liczba losowa. Istnieją metody losowego wytwarzania liczb, ale oczywiście żadna deterministyczna procedura arytmetyczna nie jest taką metodą.”

„Liczby losowe”

Z powyższego tekstu płynie kilka wniosków:

- nie ma czegoś takiego jak „liczby losowe”, a jedynie liczby, a właściwie ciągi liczb, uzyskane w sposób losowy lub nielosowy. Jedna liczba nigdy nie będzie miała charakteru losowego, potrzeba do tego ciągu liczb;
- nie istnieje deterministyczna procedura wygenerowania losowego ciągu liczb. Jest to oczywiste, komputer (a takiego urządzenia najczęściej używamy do generowania „liczb losowych”), jest maszyną deterministyczną, w której każdy kolejny krok (a w związku z tym i wszystkie następne) są określone przez stan jego pamięci, rejestrów i procesora, więc wszystkie generowane wartości można powtórzyć (jeśli się zapamięta odpowiednie stany pamięci, rejestrów, itp). Oczywiście użytkownik lub inne nieoczekiwane zdarzenie zewnętrzne może to zmienić, ale nie można na tym oprzeć generatora liczb losowych, gdyż będzie to generacja „ręczna” (w epoce przedkomputerowej posługiwano się tablicami liczb losowych do generacji ciągów liczb losowych);

„Liczby losowe” a komputery

- istnieją komputery z wbudowanymi sprzętowymi generatorami „liczb losowych”, generatory takie próbują wykorzystywać takie „informacje z zewnątrz”, np. elektryczne szumy cieplne, rozpady promieniotwórcze, które umożliwiają generowanie zdarzeń losowych, które z kolei uzyskują interpretację liczb. Generatory takie rzeczywiście generują ciągi losowe o niepowtarzalnych sekwencjach wartości i bywają stosowane w kryptologii, lecz niestety też mają wady: parametry uzyskiwanych rozkładów mogą nie być stałe i zmieniać się w zależności od czynników zewnętrznych, np. temperatury i wilgotności, a stałość parametrów rozkładu losowego jest często bardzo ważna, generatory sprzętowe najczęściej nie są uniwersalne i nie generują ciągów liczb o dowolnych rozkładach losowych;
- czasem w prostych rozwiązaniach prosi się użytkownika np. o „pomachanie myszą”, aby uzyskać nieco „losowości” do obliczeń;

Liczby losowe i pseudolosowe

- oczywiście termin „liczby losowe” jest już dość mocno zakorzeniony i mimo, że formalnie niepoprawny, jest używany, jednakże należy pamiętać, że oznacza on ciągi liczb wygenerowanych w sposób losowy lub jeszcze częściej w sposób pseudolosowy;
- pojęcie „pseudolosowy” oznacza, że wygenerowany ciąg liczb „wygląda” jak losowy (nawet wiele testów statystycznych to potwierdzi), jednakże będą to liczby wygenerowane przez program, zupełnie deterministycznie i przewidywalnie (o ile ktoś zna algorytm i wartości startowe) i ciąg taki zawsze po jakimś nawet długim okresie zacznie się powtarzać, dlatego też przy odpowiednio długiej sekwencji posiadanych wyników uda się stwierdzić jego nielosowość, czy to testami statystycznymi, czy przez obserwację długości cyklu.

Liczby losowe i generatory liczb losowych

- programowe generatory liczb losowych są jednak chętne używane (może mało chętnie przez kryptologów), gdyż są łatwe w wykonaniu, szybkie, mają dobre (zależie oczywiście od metody) parametry statystyczne - szczególnie stałość i zgodność z parametrami wybranego rozkładu, ale nie tylko;
- dość łatwo przy ich użyciu uzyskać dowolny rozkład, czy to używając jako generatora wejściowego generatora rozkładu jednostajnego i przekształcając go następnie np. metodą odwrotnej dystrybuanty (nie zawsze się da) lub filtrując uzyskiwane liczby w wymagany, bądź też używając algorytmu, generującego ciągi liczb o odpowiednich rozkładach.
- istotne jest, aby używany generator miał odpowiednie parametry do zadania, w którym jest używany, wykorzystywane w wielu algorytmach obliczeniowych liczby losowe powinny być bardzo dobrej jakości, inaczej obliczenia będą dawać słabe wyniki.
- bardzo często do inicjacji programowego generatora liczb losowych wykorzystuje się zegar systemowy, generalnie duże znaczenie ma podanie dobrych parametrów startowych do generatora.

Do czego mogą służyć liczby losowe?

Liczby losowe mają bardzo dużo zastosowań praktycznych:

- od maszynki „chybił-trafił” podpowiadającej liczby graczom w LOTTO
- przez bardziej poważne zastosowania, np. losowanie reprezentatywnych danych do ocen statystycznych, algorytmów uczenia, testów statystycznych, kontroli jakości towarów, ocen preferencji wyborców
- modelowania różnego rodzaju zjawisk fizycznych, przyrodniczych, ekonomicznych, technicznych
- w różnego rodzaju algorytmach obliczeniowych, obecnie już klasycznych, jak metoda Monte Carlo, błądzenie przypadkowe, itp.

Do czego mogą służyć liczby losowe?

- dla nas najważniejsze będą zastosowania generatorów liczb losowych w metodach uznawanych za mniej klasyczne, czyli wszelkiego rodzaju algorytmach zachłannych, heurystykach, algorytmach ewolucyjnych, rojowych, symulowanego wyżarzania i wielu innych, w których ciągi liczb losowych odpowiadają za generację nowych rozwiązań, modyfikację już posiadanych, generowanie nowych kierunków poszukiwań i przeszukiwanie dużych przestrzeni rozwiązań, które nie dadzą się przejrzeć dokładnie i w wielu innych działaniach, w których losowość ma zastąpić nieznane informacje o sposobach uzyskania nowych, lepszych rozwiązań problemu.

Programowe generatory liczb losowych

Jak to już zostało wspomniane wcześniej, obecnie raczej nie używa się sprzętowych generatorów liczb losowych, tak więc praktycznie wszędzie stosuje się już tylko programowe generatory liczb pseudolosowych, które w uproszczeniu nazywa się generatorami liczb losowych, choć należy pamiętać, że ta ich „losowość” jest bardzo dyskusyjna, gdyż w istocie są to metody **deterministyczne**.

Liniowe generatory ciągów liczb pseudolosowych

Najprostsze generatory ciągów liczb pseudolosowych, zostały zaproponowane w przez Lehmera w 1951 r. i bazują na następujących równaniach:

$$X_{n+1}=(a_1*X_n + a_2*X_{n-1} + \ldots + a_k*X_{n-k+1} + c) \bmod m$$

gdzie a_1, a_2,a_k, c i m – są całkowitymi parametrami generatora, mod – oznacza resztę z dzielenia.

lub w uproszczonej wersji:

$$X_{n+1}=(a*X_n + c) \bmod m$$

Analizując szczególnie przypadek drugiego wzoru łatwo zauważyć, że musi on być okresowy, bo możliwych do wygenerowania jest tylko m wartości, a wygenerowanie po raz drugi tej samej wartości zapoczątkuje ten sam ciąg wartości. Generatory o tej postaci nie są najlepsze (rozkład otrzymanych wartości w przestrzeni jest bardzo równomierny i „nielosowy”) i choć bywają stosowane, ich parametry silnie zależą od przyjętych wartości parametrów. Wartości gwarantujące dość dobre parametry liczb losowych można znaleźć w odpowiednich tablicach.

Liniowe generatory ciągów liczb pseudolosowych

ćwiczenie

Dla wzoru:

$$X_{n+1}=(a*X_n + c) \bmod m$$

o parametrach np.: $a=3$, $c=7$, $m=11$, $X_0=1$ policzyć wartości wyrazów X_1 do $X10$, a następnie policzyć wartość oczekiwaną (średnią arytmetyczną) i ewentualnie wariancję.

Uniwersalny generator liczb losowych o rozkładzie równomiernym (MZT)

Generator MZT (od nazwisk twórców) nazwany został uniwersalnym z powodu szczegółów implementacyjnych, które powodują, że ma dobre parametry dla maszyn o nawet dość krótkim słowie maszynowym (obecnie ten parametr ma już raczej tylko znaczenie historyczne).Jest to złożenie dwóch tzw. generatorów Fibonacciego, opisanych jako:

$$V_n = V_{n-97} \circ V_{n-33}$$

gdzie $x \circ y = x-y$ dla $x \geq y$ i $x \circ y = x-y+1$ dla $x<y$ i $V_1, V_2, \ldots, V_{97}$ są generowane przy użyciu ciągów bitów takich, że: $V_1 = 0,b_1b_2...b_{24}$, $V_2 = 0,b_{25}b_{26}...b_{40}$, ... itd., a ciągi bitów są generowane przy użyciu 2 generatorów:

$$y_n = (y_{n-3} * y_{n-2} * y_{n-1}) \bmod 179 \quad \text{i} \quad z_n = (53 * z_{n-1} + 1) \bmod 169$$

$$b_n = \begin{cases} 0 \text{ - dla } y_n * z_n \bmod 64 < 32 \\ 1 \text{ - w przeciwnym przypadku} \end{cases}$$

przy wartościach startowych y_1, y_2, y_3 wybranych z zakresu {1, 2, ... ,179} i $y_1 * y_2 * y_3 \neq 1$ oraz z_1 z zakresu {0, 1, ... , 168}.

$$c_n = c_{n-1} \circ (7654321/16777216)$$

gdzie $n \geq 2$, $a \circ b = a-b$ *dla* $a \geq b$ *i* $a \circ b = a-b+(16777213/16777216)$ *dla* $a<b, c_1 = 362436/16777216$, ostatecznie:

$$MZT_n = V_n \circ c_n$$

Uniwersalny generator liczb losowych o rozkładzie równomiernym

kod w C

```
static double uu[97],cc=362436.0/16777216.0,cd=7654321.0/16777216.0,cm=16777213.0/16777216.0;
static int ip=97,jp=33;
void rstart(int i,int j,int k,int l){
  int ii,jj,m,wi,wj,wk,wl;
  double s,t;
  wi=i;wj=j;wk=k;wl=l;
  for(ii=0;ii<97;ii++){
    s=0;t=0.5;
    for(jj=1;jj<=24;jj++){
      m=(((wi*wj)%179)*wk)%179;
      wi=wj;wj=wk;wk=m;
      wl=(53*wl+1)%169;
      if((wl*m)%64>=32) s+=t;
      t*=0.5;
    }
    uu[ii]=s;
  }
}
double uni(){
  double pom;
  pom=uu[ip-1]-uu[jp-1];
  if(pom<0.0) pom+=1;
  uu[ip-1]=pom;
  ip--;
  if(ip==0) ip=97;
  jp--;
  if(jp==0) jp=97;
  cc-=cd;
  if(cc<0.0) cc+=cm;
  pom-=cc;
  if(pom<0.0) pom+=1;
  return(pom);
}
```

Metody tworzenia generatorów o dowolnych rozkładach

- metoda odwrotnej dystrybuanty
- metoda eliminacji – zastosowanie swoistego „sita” odfiltrowującego niektóre wygenerowane wartości z rozkładu (-ów) jednostajnego tak, że powstaje rozkład wymagany
- metoda superpozycji rozkładów – oparta na wzorach o superpozycji rozkładów
- metoda ROU (ratio of uniforms method) - oparta na twierdzeniu o ilorazie rozkładów równomiernych
- metody mieszane

Generator liczb losowych o rozkładzie normalnym

Rozkład normalny, to jeden z najczęściej używanych rozkładów zmiennych losowych, gęstość rozkładu ma wzór:

$$f_{\mu,\sigma}(x)=\frac{1}{\sigma*\sqrt{2*\pi}}\exp\left(-\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

Najczęściej generuje się wartości rozkładu standaryzowanego o μ=0 i σ=1, jako, że rozkłady o innych wartościach tych parametrów można otrzymać przez proste przekształcenie, polegające na dodaniu μ i pomnożeniu przez σ rozkładu standardowego.
Standardowy rozkład normalny:

$$f_{0,1}(x)=\frac{1}{\sqrt{2*\pi}}\exp\left(-x^2\right)$$

Generator liczb losowych o rozkładzie normalnym

przykładowy algorytm

Pseudokod prostego generatora rozkładu normalnego, opartego na metodzie ROU:

```
Repeat
  Generuj U o rozkładzie równomiernym U(0,1)
  Generuj V o rozkładzie równomiernym U(-√(2/e) ,√(2/e))
     X=V/U
     If X²≤2(3-U(4+U)) then akceptuj=true else
         If X²≤2/U-2U then
             If X²≤-4ln(U) then akceptuj=true
until akceptuj
Return X
```

Dlaczego generator liczb losowych o rozkładzie Cauchy'ego?

W wielu zastosowaniach w algorytmach SI stosuje się w praktyce zamiast rozkładu normalnego rozkład Cauchy'ego.
Z czego to wynika?
Otóż rozkład normalny jest stosowany bardzo często w praktyce, a jeszcze częściej w rozważaniach teoretycznych, gdyż jest łatwy w interpretacji, bardzo dobrze znane są jego właściwości, istnieje wiele twierdzeń, których on jest podstawą, przy odpowiednich założeniach prawie każdy inny rozkład prawdopodobieństwa można zamodelować rozkładem normalnym. Jednakże w rozwiązaniach praktycznych zauważono pewne jego mankamenty. Otóż ze standaryzowanych generatorów rozkładu normalnego otrzymujemy właściwie tylko wartości z przedziału $(-3\sigma, 3\sigma)$. Wynika to częściowo z jego właściwości (te wartości są najbardziej prawdopodobne), ale często też z jego konstrukcji.

Dlaczego generator liczb losowych o rozkładzie Cauchy'ego?

Otóż w wielu generatorach wygenerowanie wartości nie z przedziału $(-3\sigma, 3\sigma)$, lecz szerszego jest niemożliwe z uwagi na jego metodę działania. Takiego typu generatory nie będą odpowiednie do wielu zastosowań (szczególnie w SI). Zresztą nawet i w generatorze dokładnie odwzorowującym rozkład normalny wartości takie pojawiają się rzadko (lub nigdy!), dla wielu algorytmów SI za rzadko. Znacznie lepsze właściwości w tej kwestii ma rozkład Cauchy'ego i dlatego jest stosowany w praktyce, choć w rozważaniach teoretycznych raczej rozkładu normalnego nie zastąpi.

Rozkład Cauchy'ego

Rozkład Cauchy'ego $C(\theta, \lambda)$ opisuje wzór:

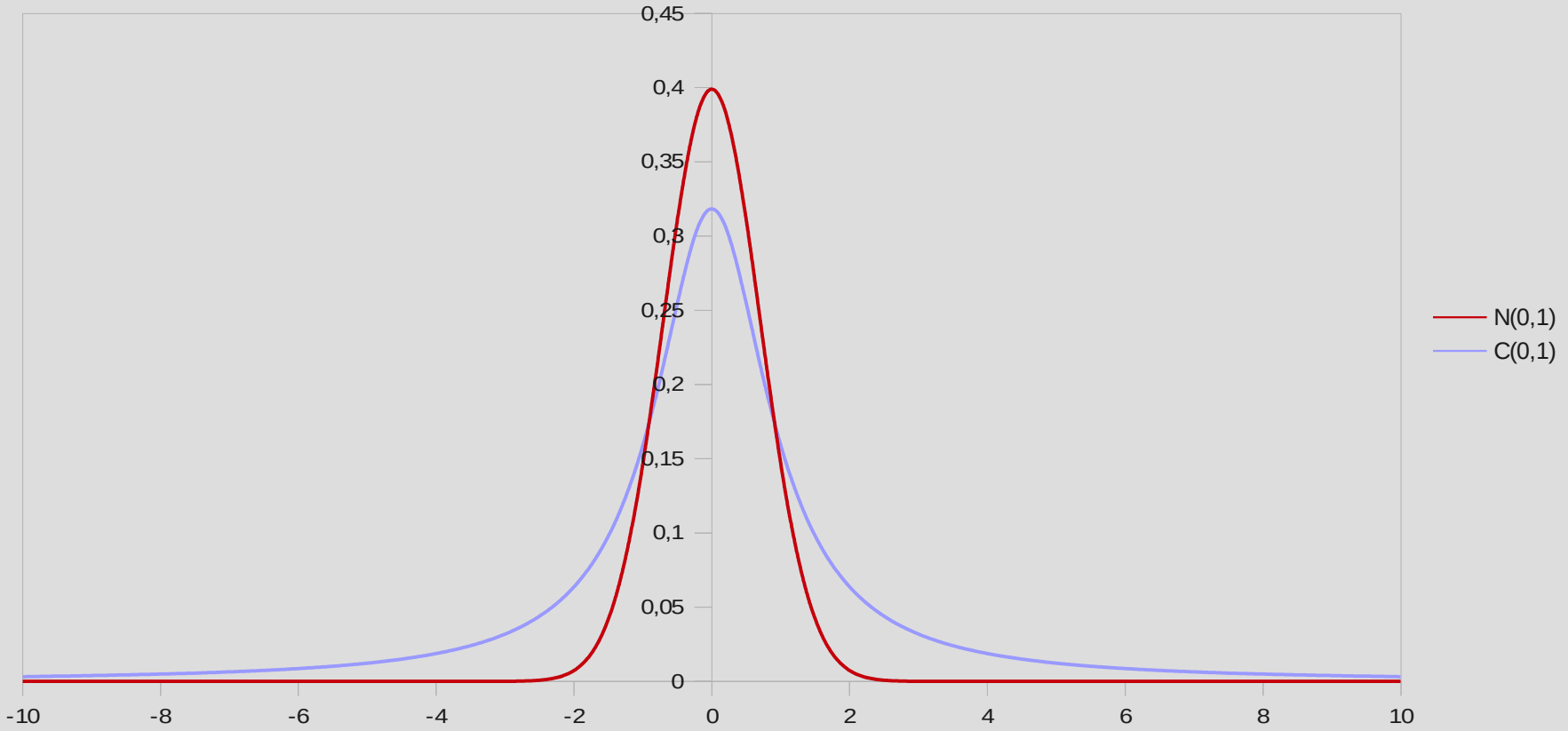
$$f_{\theta,\lambda}(x) = \frac{\lambda}{\pi} * \frac{1}{\lambda^2 + (x-\theta)^2} \quad -\infty < x < \infty$$

a wariant znormalizowany $C(0,1)$:

$$f_{0,1}(x) = \frac{1}{\pi} * \frac{1}{1+x^2} \quad -\infty < x < \infty$$

Rozkład normalny a rozkład Cauchy'ego

wersje znormalizowane



Rozkład Cauchy'ego w algorytmach SI

Z przedstawionych wykresów widać, że rozkład Cauchy'ego ma niższe wartości prawdopodobieństwa w pobliżu „środka” (rozkład nie posiada wartości oczekiwanej i wariancji, co oczywiście jest dużym problemem w rozważaniach teoretycznych) niż rozkład normalny, a za to wyższe w tzw. „ogonach”, dzięki czemu częściej generuje wartości oddalone od środka. Ma to znaczenie w wielu algorytmach, w których takie „makromutacje/makroperturbacje” mogą przenieść poszukiwania rozwiązania w zupełnie nowe rejony, a co prawie się nie zdarza dla rozkładu normalnego. Oczywiście rozkład ten jest przydatny raczej w przypadku problemów nie będących dyskretnymi.

Generatory rozkładu Cauchy'ego

Ponieważ dystrybuanta rozkładu Cauchy'ego wyraża się wzorem:

$$F(x)=\frac{1}{\pi}*arc\ tg(x)+\frac{1}{2}$$

to można łatwo zbudować generator metodą dystrybuanty odwrotnej w oparciu o wzór:

$$X=tg(U*\pi),$$

gdzie U jest rozkładem jednostajnym na przedziale (-1/2,1/2).

Generatory rozkładu Cauchy'ego

Inny sposób, to użycie generatora programowego, o następującym algorytmie:

```
Generuj X o rozkładzie równomiernym U(-1, 1)
Generuj Y o rozkładzie równomiernym U(0, 1)
If (1/2*Y>f_u(X)-(2/π-1/2)
   then Generuj X o rozkładzie równomiernym U(-1,1)
Return X
```

gdzie:

$$f_u(x)=\begin{cases}\frac{2}{\pi}*\frac{1}{1+x^2} \text{ - gdy } -1\leqslant x\leqslant 1\\ 0 \text{ - w przeciwnym przypadku}\end{cases}$$

Praktyczne użycie generatora rozkładu jednostajnego

Bardzo często w standardowej bibliotece dołączanej do pakietu oprogramowania (np. kompilatora jakiegoś języka) jest generator o rozkładzie jednostajnym i dyskretnym o generowanych wartościach np. z przedziału 0...RAND_MAX. Często zależy nam na zamianie go na generator dyskretny o wartościach z pewnego przedziału np. {0,...K-1} lub ciągły z przedziału ⟨0, 1⟩. Jeśli chodzi o pierwszy punkt,to często robi się to tak:

~~x=rand()%K;~~ - (notacja z C/C++) jest to sposób niepoprawny, niszczący „losowość" wygenerowanej wartości!

wariant poprawny to:

x=**(int)(K*(double)rand()/(RAND_MAX+1.0))**; - ten sposób umożliwia pełne wykorzystanie wszystkich bitów wylosowanej liczby

Dla przypadku ⟨0, 1⟩ jest to:

y=**(double)rand()/((double)RAND_MAX)**;

Generator permutacji losowych

W wielu zagadnieniach (np. problem komiwojażera) potrzebne są losowe permutacje pewnego zbioru liczb. Można w łatwy sposób zbudować taki generator w oparciu o generator dyskretny jednostajny z zakresu 0...*N*-1 i tablicę zawierającą *N* permutowanych elementów:

- Wypełniamy tablicę permutowanymi *N* elementami
- ustawiamy zakres losowania *K* indeksów z tablicy na *N*-1 (numeracja od 0)
- powtarzamy dopóki $K \geq 0$
  - losujemy element tablicy o indeksie z zakresu 0..*K* – jest to element generowanej permutacji
  - zamieniamy go z elementem w tablicy o indeksie *K*
  - zmniejszamy *K* o 1
- wylosowane elementy tworzą permutację elementów zbioru

Działanie to można powtarzać wielokrotnie, permutacje mogą się powtórzyć, ale generator wszystkich różnych permutacji nie jest już generatorem losowym,

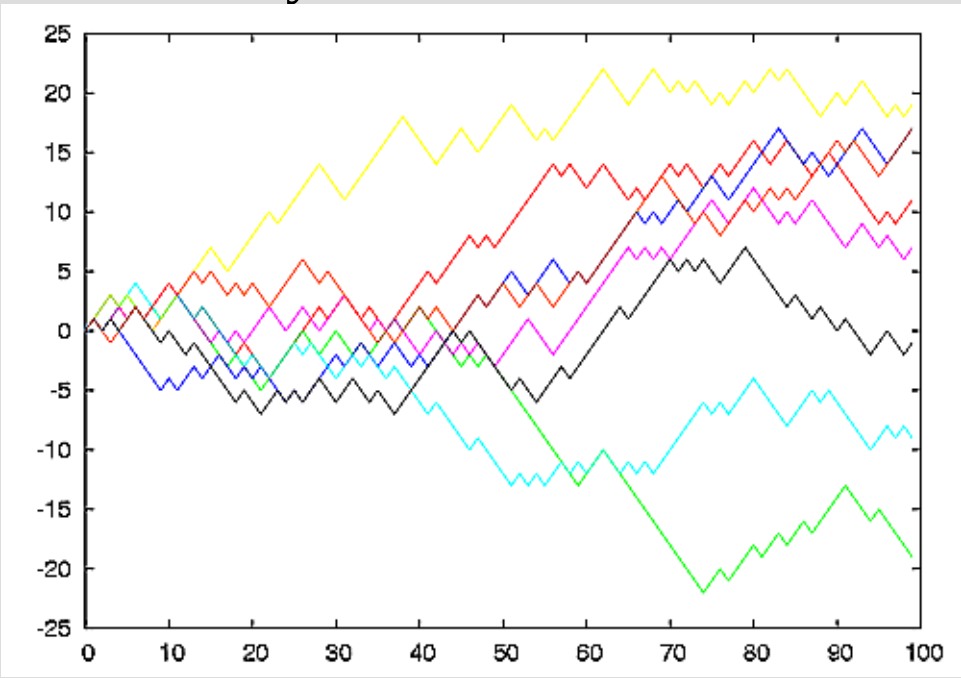
Liczby losowe a Metoda Monte Carlo

Metoda ta służy do modelowania zjawisk zbyt złożonych, aby możliwe było rozpatrzenie wszystkich możliwości. Dość popularnym przykładem ilustrującym działanie metody jest obliczenie całki (czyli np. powierzchni) jakiejś skomplikowanej krzywej.
W prezentowanym przykładzie tą „skomplikowaną” krzywą będzie koło. Koło ma środek w punkcie (0,0), promień r=1 i jest wpisane w kwadrat o wierzchołkach w (-1, 1), (1, 1), (1, -1), (-1, -1) i polu P_k=4.

Losujemy z **rozkładu jednostajnego** U(-1, 1) pary liczb, reprezentujące współrzędne punktów kwadratu. Wartość $P_o=P_k*k/n$, będzie oszacowaniem pola koła w zależności od liczby punktów, które zostały wylosowane w obrębie koła (k) (należy to sprawdzić z tzw. równania okręgu $x^2+y^2\leq r^2$) do liczby wszystkich punktów (n). W podobny sposób możemy metodą MC oszacować wartość π.
Metoda Monte Carlo jest jednym z pierwszych algorytmów z pogranicza heurystycznych i klasycznych, w których do szacowania wyniku używane są ciągi liczb losowych.

Liczby losowe i błądzenie przypadkowe

Metoda ta służy do modelowania zjawisk fizycznych, ekonomicznych, przyrodniczych i innych. Jest to przykład **procesu stochastycznego** w którym badamy poruszanie się obiektu „napędzanego” losowością. Przy okazji można też analizować wartość jakiejś funkcji oceny położenia obiektu, co umożliwia rozwiązywane tą metodą również zdań obliczeniowych.



Przykład błądzenia przypadkowego.
Źródło: Wikipedia

Liczby losowe w algorytmie przeszukiwania lokalnego - algorytm (największego) wzrostu z wielostartem

begin
    t:=0
    repeat
        local:=FALSE
        wylosuj rozwiązanie początkowe x_s
        oceń x_s
        x_0:=x_s
        repeat
            wybierz n nowych rozwiązań x_1...x_n przez **losową modyfikację** x_0
            oceń rozwiązania x_1...x_n
            wybierz najlepsze rozwiązanie x* z x_1...x_n
            jeśli x* lepsze od x_0 to x_0:=x* jeśli nie to local:=TRUE
        until local
        zapamiętaj najlepsze x_0
        t:=t+1
    until t=MAX
end

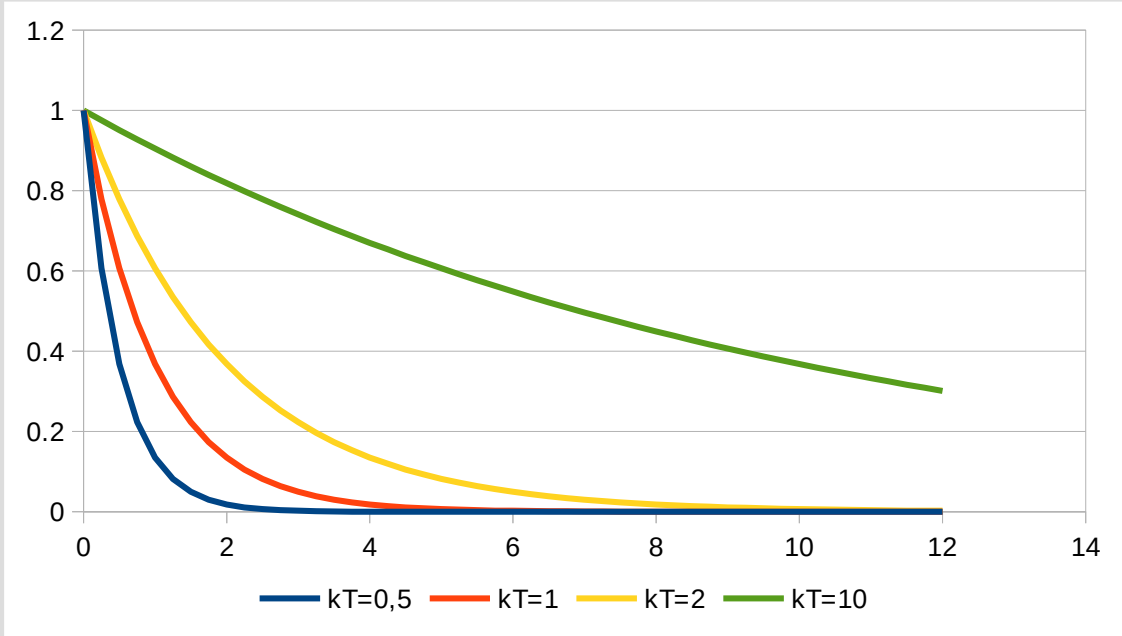
Losowość w algorytmie Metropolisa

wygeneruj losowo rozwiązanie startowe x i oceń jego jakość.
$x^*=x$ //początkowe rozwiązanie jest także najlepszym rozwiązaniem
ustal $T(0)$ i schemat chłodzenia $g(T(n))$ //$T(0)$ – temperatura początkowa; $g(T(n))$ – funkcja zmian temperatury w trakcie obliczeń
$n=0$
do {
  $licznik=0$
  do {
    wygeneruj nowe rozwiązanie x' operatorem **perturbacji/mutacji**
    if ($F(x') < F(x)$) { //minimalizacja, przy maksymalizacji porównanie odwrotne
      $x=x'$ //jeśli nowe rozwiązanie jest lepsze, to jest zawsze akceptowane
      $x^*=x$
    else {
      z przedziału (0,1) wylosuj liczbę y
      if (y<exp(-($F(x')- F(x)$)/kT)) { //tu wykorzystywany jest **rozkład Boltzmanna** do akceptacji rozwiązań
        $x=x'$
        if ($F(x)<F(x^*)$) $x^*=x$
      }
    $licznik=licznik+1$
  }
  while ($licznik$<MAX) //MAX – liczba powtórzeń przy stałej temperaturze
  $T(n+1)=g(T(n))$ //zmiana temperatury
  $n=n+1$
}
while (warunek końca)

Rozkład Boltzmanna

Istotnym elementem, różniącym tę metodę od innych metod przeszukiwania lokalnego, jest reguła akceptacji rozwiązań wykorzystująca rozkład Boltzmanna.
Rozkład Boltzmanna opisuje między innymi rozkład energii (stanów energetycznych) cząstek w pewnej temperaturze, wyraża się wzorem:

$$P(\delta E) = \alpha * \exp\left(\frac{-\delta E}{kT}\right)$$



Algorytmy ewolucyjne i losowość

ogólny schemat działania

t=0
Losowa inicjacja populacji startowej P(0)
Ocena rozwiązań w P(0)
do {
  T(t)=**Reprodukcja** P(t) //powielenie i **krzyżowanie** osobników
  O(t)=**Modyfikacja** T(t) //**mutacja** osobników
  Ocena rozwiązań w O(t)
  P(t+1)=**Selekcja**/**Sukcesja**(O(t) lub O(t)∪P(t))
  t=t+1
}
while(warunek stopu)

Algorytmy ewolucyjne i losowość

selekcja osobników do populacji potomnej
selekcja proporcjonalna (ruletkowa)

- Każdy osobnik ma przyporządkowaną wartość funkcji dopasowania *f(j)*.
- **Prawdopodobieństwo wyboru** i **wartość oczekiwana** liczby potomków to:

$$p(j)=\frac{f(j)}{\sum_{l=1}^{k} f(l)} \qquad E(p(j))=\frac{k*f(j)}{\sum_{l=1}^{k} f(l)}=\frac{f(j)}{\bar{f}}$$

gdzie: *k* – liczność populacji, *f(j)* – wartość funkcji dopasowania osobnika *j*, $\bar{f}$ - średnia wartość funkcji dopasowania populacji.

- Procedurę **losowego wyboru** przeprowadzamy *k* (dokładniej tyle razy, ile osobników ma mieć populacja potomna, nie zawsze musi to być taka sama liczba jak dla populacji rodzicielskiej) razy.

Wniosek!
Osobniki o większym prawdopodobieństwie selekcji powinny mieć więcej potomków niż osobniki o mniejszym, a tym samym populacja powinna ewoluować w kierunku lepszego przystosowania – lepszych rozwiązań.

Algorytm mrówkowy i losowość

l. Ustalenie początkowego poziomu feromonu w miejscach (krawędziach grafu) odpowiadających przejściom między kolejnymi elementami rozwiązywanego problemu

do {

2. Odparowanie feromonu (czyli zmniejszenie wartości wag feromonowych krawędzi).

3. **Wylosowanie** punktu startu agenta-mrówki.

4. **Wylosowanie** ścieżek na podstawie poziomu feromonu (wag) na przejściach między kolejnymi elementami rozwiązania.

5. Naniesienie feromonu (zwiększenie wag) krawędzi wykorzystanych w rozwiązaniu.

6. Operacje dodatkowe – np. zbalansowanie feromonu ścieżek (próba wyjścia z optimum lokalnego), dodatkowe wzmocnienie pewnych krawędzi, itp.

}

while (warunek stopu)

Algorytm mrówkowy i losowość

Algorytm mrówkowy odwzorowuje zwyczaje mrówek poszukujących pożywienia. Zazwyczaj chodzą one po śladach zapachowych (feromonowych), pozostawionych przez inne mrówki, czasem jednak zmieniają droge, jeśli poczują pożywienie. W algorytmie to modelującym, każda mrówka wybiera następny ruch jako wypadkową z 2 przypadków: iść za zapachem pożywienie lub śladem feromonowym, co opisuje odpowiednie prawdopodobieństwo:

$$p_{i,j}^{k}(t) = \frac{[\tau_{i,j}(t)]^{\alpha} * [\eta_{i,j}]^{\beta}}{\sum_{l=1}^{n} ([\tau_{i,l}(t)]^{\alpha} * [\eta_{i,l}]^{\beta})}$$

$p^{k}_{i,j}(t)$ – prawdopodobieństwo wyboru drogi; $\tau_{i,j}(t)$ – natężenie feromonu, $\eta_{i,j}$ – waga krawędzi wynikająca z optymalizowanego zadania, np. odwrotność odległości między miastami w przypadku TSP; α, β - parametry określające wpływ feromonu i wagi krawędzi wynikającej z zadania na p. wyboru drogi; i, j - indeksy elementów rozwiązania (np. miast); k – numer mrówki; n – liczba punktów dopuszczalnych, które można włączyć do rozwiązania w danym momencie.

Sieci neuronowe i losowość

Sieci neuronowe modelują działanie neuronów i ich grup, w miarę możliwości także całych fragmentów mózgu. Jedną z najważniejszych sieci tego typu jest sieć typu backpropagation (BP). Sieć tam ma umiejętność uczenia się dowolnych funkcji,ale aby mogła się czegoś nauczyć konieczne jest zainicjowanie jej strojonych wag (połączeń między neuronami) **wartościami losowymi**.
Innym rodzajem sieci neuronowej, wykorzystującej losowość, to tzw. maszyna Boltzmanna, jest to sieć, której działanie opisuje tzw. funkcja energetyczna, a przejścia między stanami są losowe, realizują pewien **proces stochastyczny**. Sieć taka może rozpoznawać wzorce, działać jako pamięć asocjacyjna, a nawet rozwiązywać problemy optymalizacyjne.

Algorytmy heurystyczne i losowość

Zaprezentowane tu przykłady algorytmów heurystycznych, wykorzystujących losowość nie wyczerpują wszystkich sposobów, metod i zastosowań liczb losowych. Są to jedynie reprezentatywne przykłady, które pokazują znaczenie losowości w metodach sztucznej inteligencji. Przedstawione algorytmy zostaną omówione dokładniej na kolejnych zajęciach.