Optymalizacja (minimalizacja) metodą Monte Carlo

Plan wykładu

- zagadnienie optymalizacji
- metoda symulowanego wyżarzania (SA)
  - algorytm Metropolisa
  - modyfikacja algorytmu SA: tunelowanie stochastyczne
  - przykład 1: najkrótsza trasa w mieście
  - przykład 2: problem Thomsona – klaster z oddziaływaniem Lennarda-Jonesa
  - przykład 3: modelowanie struktury fullerenów
- algorytmy ewolucyjne
  - algorytmy genetyczne (GA)
    - przykład: najkrótsza trasa w mieście

Literatura:

T. Pang, „Introduction to computational physics”
M.H. Kalos, P.A. Whitlock, „Monte Carlo methods”
A.K. Hartmann, H. Rieger, „Optimization algorithms in physics”
R.L. Haupt, D.H. Werner, „Genetic algorithms in electromagnetics”
W. Paszkowicz, „Genetic Algorithms, a Nature-inspired tool: survey of applications in materials science and related fields”
Materials and Manufacturing Processes 24, (2009) 174

Zagadnienie optymalizacji – sformułowanie problemu

- Problem optymalizacji wartości funkcji możemy sformułować jako zagadnienie poszukiwania wartości ekstremalnych, tj. maksimum bądź minimum. Maksimum możemy zamienić w minimum przemnażając funkcję przez czynnik (-1), zatem proces optymalizacji sprowadzany jest zazwyczaj do poszukiwania minimum wartości funkcji.

- minimum może być:
  - globalne (tego zazwyczaj szukamy)
  - lokalne (to zazwyczaj przeszkadza)

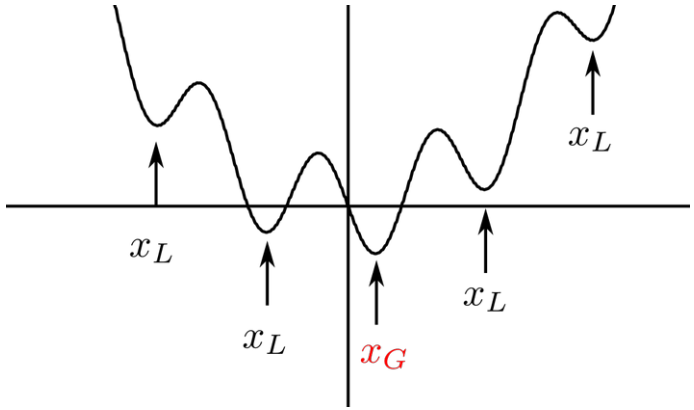
Zdefiniujmy matematycznie punkt w którym funkcja posiada minimum globalne (x_G)

$$f : R^n \to R$$

$$\min f(\vec{x}) = f(\vec{x}_G) \iff \bigwedge_{\vec{x} \in R^n} f(\vec{x}_G) < f(\vec{x})$$

oraz punkt stanowiący minimum lokalne (x_L)

$$\exists \varepsilon : \varepsilon > 0, \varepsilon \in R \quad \bigwedge_{\|\vec{x} - \vec{x}_L\| < \varepsilon} \quad f(\vec{x}_L) < f(\vec{x}) \quad \wedge \quad f(\vec{x}_L) > f(\vec{x}_G)$$



Oprócz minimum możemy też napotkać inny charakterystyczny punkt – punkt siodłowy

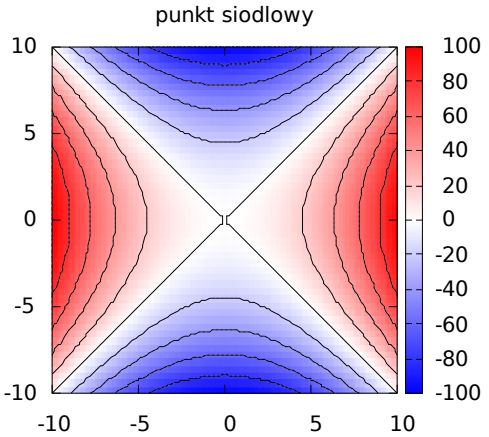
$$\vec{r}_S = \begin{bmatrix} \vec{x}_S \\ \vec{y}_S \end{bmatrix}$$

$$\exists \varepsilon : \varepsilon > 0, \varepsilon \in R \bigwedge_{\substack{\|\vec{x}-\vec{x}_S\|<\varepsilon \\ \|\vec{y}-\vec{y}_S\|<\varepsilon}} f(\vec{x}_S, \vec{y}) \leq f(\vec{x}_S, \vec{y}_S) \leq f(\vec{x}, \vec{y}_S)$$

inaczej: w zależności od kierunku w którym będziemy się poruszać,
wartość funkcji może maleć lub rosnąć

Przykład punktu siodłowego dla funkcji

$$f(x, y) = x^2 - y^2$$



„Klasyczne” metody poszukiwania minimum wartości funkcji to metody iteracyjne

- (1D) metoda złotego podziału, metoda interpolacji kwadratowej Powella
- (>1D) metoda sprzężonego gradientu, największego spadku, Newtona (metody gradientowe)

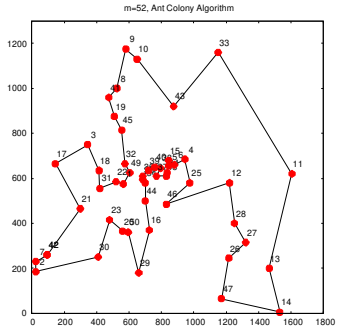
  lub metoda Simplex (bezgradientowa)

- wadą metod standardowych jest to że potrafią znaleźć minimum zazwyczaj, gdy punkt startowy znajduje się w jego pobliżu (tak działają metody gradientowe)
- metody te znajdują pojedyncze minimum, więc nie ma pewności czy jest to minimum lokalne czy globalne (chociaż tej pewności na ogół nie mamy także w przypadku metody MC)
- zastosowanie standardowego podejścia sprawdza się w przypadku gdy liczba wymiarów przestrzeni, w której poszukujemy rozwiązania jest niewielka, rzędu 1-10
- klasyczne metody działają tylko dla problemów zdefiniowanych przy pomocy funkcji ciągłych, w przypadkach dyskretnych (np. problemy kombinatoryczne) stają się bezużyteczne

Przykład - problem komiwojażera

Dany jest zbiór miast które komiwojażer chce odwiedzić tylko raz i w takiej kolejności aby trasa je łącząca była najkrótsza i zamknięta.

położenia miast $\vec{d} = \{\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_n\}$



$$\min S = \sum_i \sum_{j>i} c_{ij} \cdot s_{ij} + s_{fl}, \qquad s_{ij} = |\vec{r}_i - \vec{r}_j|$$

s_{lf} – odcinek pomiędzy ostatnim a pierwszym miastem

$$c_{ij} = \begin{cases} 0, & \text{brak połaczenia} \\ 1, & \text{miasta połaczone} \end{cases}$$

Ponieważ ważna jest kolejność odwiedzanych miast więc moglibyśmy dokonać zmiany kolejności pierwotnej miast a następnie długość trasy liczyć jako odległości między kolejnymi parami – sąsiadującymi na liście miastami

pierwotne położenie miast

$$\vec{d} = \{\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_n\}$$

wektor indeksów miast zgodny z miejscem na liście

$$P_1 = \{1, 2, 3, \ldots, n\}$$

optymalna kolejność miast

$$\vec{d}_\alpha = \{\vec{r}_{\alpha_1}, \vec{r}_{\alpha_2}, \ldots, \vec{r}_{\alpha_n}\}$$

wektor indeksów dla optymalnej trasy

$$P_\alpha = \{\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n\}$$

$$\min S = \sum_{i=1}^{n-1} s_{\alpha_i \alpha_{i+1}} + s_{\alpha_n \alpha_1}$$

- S to **funkcja kosztu**, szukamy takiej kombinacji miast aby znaleźć jej minimum

Zauważmy że wektor P_α jest w zasadzie permutacją ciągu liczb zawartych w P_1.
Ile ścieżek możemy skonstruować?

$$N_\alpha = n!$$

$$n = 10 \rightarrow N_\alpha = 10! = 3.6 \cdot 10^6$$

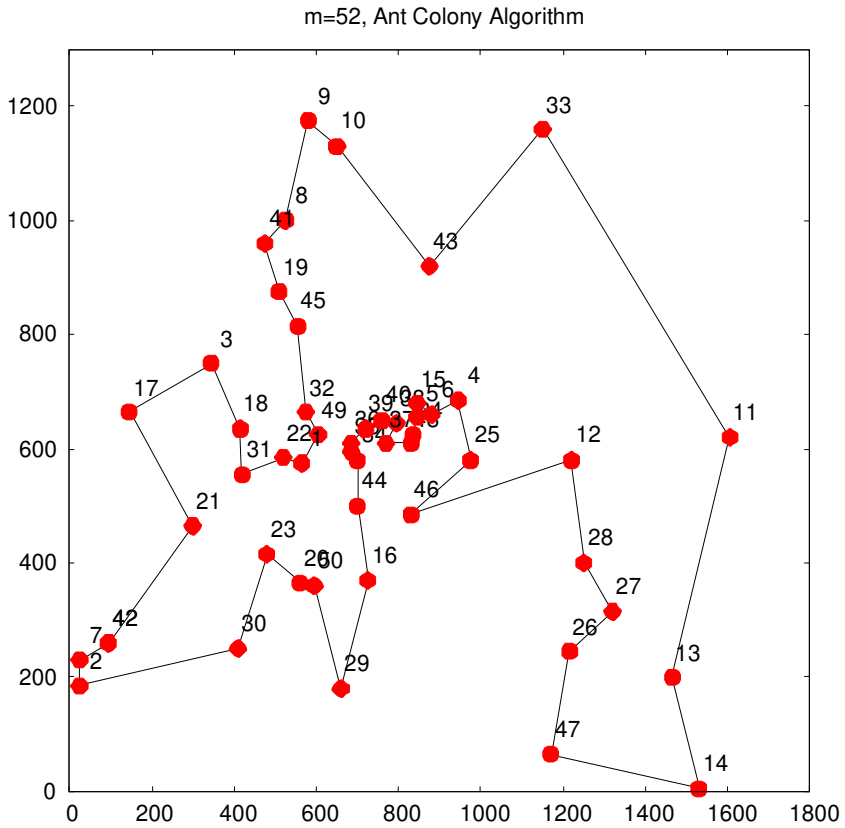
$$n = 20 \rightarrow N_\alpha = 20! = 2.4 \cdot 10^{18}$$

$$n = 50 \rightarrow N_\alpha = 50! = 3 \cdot 10^{64}$$

- metoda „brute force” polegająca na sprawdzaniu kolejnych permutacji jest w tym przypadku bezużyteczna
- jedynie metody stochastyczne (Monte Carlo) są w stanie poradzić sobie z tym problemem

Przykład

- 52 miasta
- najkrótszą trasę znaleziono algorytmem ewolucyjnym (ant colony algorithm)
- czas trwania symulacji: 2 sekundy



- indeksy przy punktach oznaczają położenie miasta na pierwotnej liście (wejściowej)

Przykład. Poszukiwanie konfiguracji układu oddziałujących ciał o najmniejszej energii
np. klastra atomów, wycinka kryształu itp.

$$E_{tot} = V(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_n) = \underbrace{\sum_i V_1(\vec{r}_i)}_{\substack{\text{pola} \\ \text{zewnetrzne}}} + \underbrace{\sum_i \sum_{j>i} V_2(\vec{r}_i, \vec{r}_j)}_{\text{dwucialowe}} + \underbrace{\ldots}_{\substack{\text{trojcialowe} \\ \text{inne}}}$$

Znalezienie minimum energii staje się trudne z dwóch powodów:

- liczba cząstek w układzie określa wymiar przestrzeni, w której pracujemy (d to liczba współrzędnych opsiujących pojedynczą cząstkę) $N = d \cdot n, \quad d = 1, 2, 3$

- w zależności od tego jak skomplikowaną postać ma potencjał oddziaływania zależy liczba lokalnych minimów funkcji kosztu (energii całkowitej), mogą one wręcz uniemożliwić znalezienie minimum globalnego

Rozkład Boltzmanna

W metodzie symulowanego wyżarzania wykorzystujemy dwa czynniki:

- rozkład Boltzmanna
- łańcuch Markowa

Rozkład Boltzmanna został zdefiniowany dla układów opisywanych przy użyciu **zbioru kanonicznego**, ustalone są wówczas parametry: N, V, T
(N -liczba cząstek, V – objętość układu, T – temperatura bezwzględna)

Określa on prawdopodobieństwo realizacji mikrostanu o energii E_i

$$p(E_i) = p_i = \frac{\exp\left(-\frac{E_i}{kT}\right)}{Z} = \frac{\exp\left(-\frac{E_i}{kT}\right)}{\sum_j \exp\left(-\frac{E_j}{kT}\right)}$$

Z jest funkcją rozdziału i pełni rolę czynnika normalizacyjnego.
Zazwyczaj jej nie znamy bo nie wiemy jakie są energie E_i.

Ponieważ zakładamy, że tylko N,V,T są stałe, więc energia układu może się zmieniać w czasie. To oznacza że dla T>0 układ może znajdować się w stanach o różnej energii, ale wyraźnie będą preferowane stany o niższej energii (większe p(E)) niż te o wysokiej.

Dysponując wyrażeniem określającym prawdopodobieństwo obsadzenia stanów, moglibysmy określić wartość oczekiwaną energii układu dla danej temperatury T

$$\langle E \rangle_{NVT} = \sum_j E_j p_j$$

Łańcuch Markowa, algorytm Metropolisa

Pomimo prostoty wyrażenia nie jesteśmy w stanie od razu wyznaczyć wartości oczekiwanej energii, gdyż nie znamy Z.
Zauważmy, że energia układu jest funkcją położeń i pędów cząstek – opisywanego wektorem w przestrzeni fazowej

$$E_i = E_i(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_n, \vec{p}_1, \vec{p}_2, \ldots, \vec{p}_n) = E_i(X)$$

zmiana jednej współrzędnej zmienia położenie w przestrzeni fazowej czyli stan X->Y

- różne realizacje wektora X będą odpowiadały różnym prawdopodobieństwom realizacji tych stanów
- prawdopodobieństwo realizacji stanu w przestrzeni fazowej (fgp) zapiszmy jako f(X)
  z warunkiem normalizacji

$$\int_\Omega f(X)dX = 1, \qquad dX = dr_1 dr_2 \ldots dp_1 dp_2 \ldots$$

- w warunkach równowagi termodynamicznej prawdopodobieństwa przejścia
  pomiędzy stanami X i Y oraz Y i X powinny być identyczne

$$\underbrace{K(X|Y)f(Y)}_{Y\to X} = \underbrace{K(Y|X)f(X)}_{X\to Y}$$

Częstość przejść K zdefiniujmy jako iloczyn częstości T (którą sami określimy)
oraz nieznanego prawdopodobieństwa akceptacji A

$$A(X|Y)T(X|Y)f(Y) = A(Y|X)T(Y|X)f(X)$$

$$A(X|Y) = A(Y|X)\frac{T(Y|X)f(X)}{T(X|Y)f(Y)} = A(Y|X)q(X|Y)$$

Nie znamy A(Y|X), więc chcąc znaleźć A(X|Y) musimy oprzeć się jedynie na znajomości wartości q(X|Y)

Metropolis zaproponował aby prawdopodobieństwo akceptacji nowego stanu określać na podstawie wyrażenia

$$p_{acc} = A(X|Y) = \min\{1, q(X|Y)\}$$

Od nas zależy postać wyrazu określającego cząstość przejść T.
Najprościej jest założyć

$$T(X|Y) = T(Y|X)$$

Ponadto wykorzystując fakt że fgp w przestrzeni fazowej ma postać

$$f(X) = \frac{\exp\left(-\frac{E(X)}{kT}\right)}{Z}$$

możemy w łatwy sposób określić prawdopodobieństwo akceptacji nowego stanu

$$q(X|Y) = \frac{T(X|Y)\exp\left(-\frac{E(X)}{kT}\right)}{T(X|Y)\exp\left(-\frac{E(Y)}{kT}\right)} = \exp\left(-\frac{E(X) - E(Y)}{kT}\right)$$

$$\mathrm{p}_{acc} = \min\left\{1, \exp\left(-\frac{E(X)-E(Y)}{kT}\right)\right\}$$

- wzór Metropolisa

Prawdopodobieństwo akceptacji zaproponowane przez Metropolisa nie jest jedynym wyborem.
Alternatywnie można je określić następująco (**wzór Glaubera**)

$$p_{acc}^{G} = A'(X|Y) = \frac{q(X|Y)}{1+q(X|Y)}$$

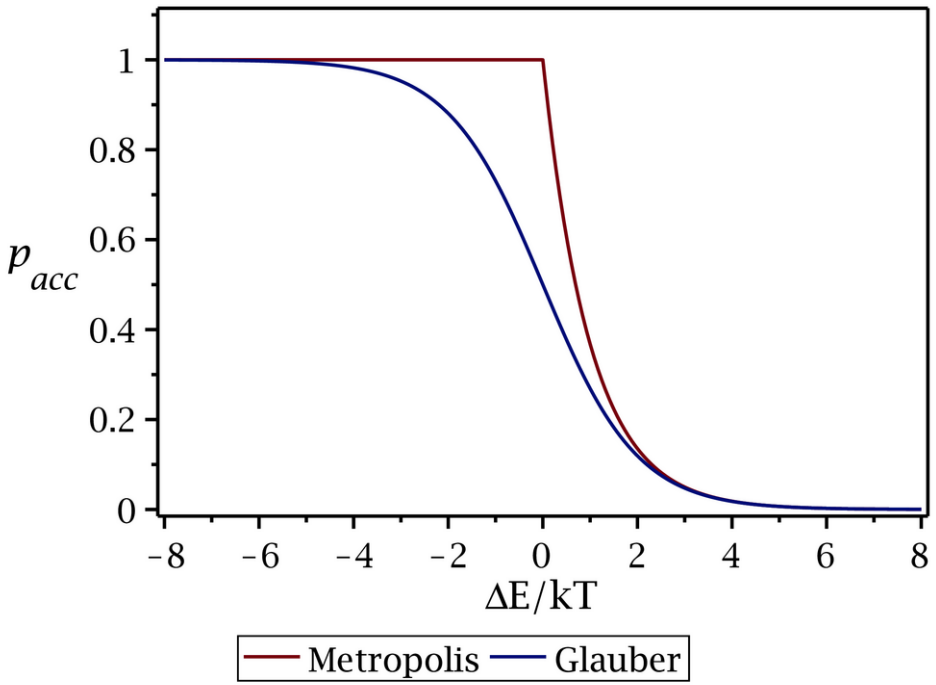
- taka postać p_{acc} również zapewnia spełnienie warunku szczegółowej równowagi

Glauber

$$p_{acc}^{G} = \frac{\exp\left(-\frac{E(X)-E(Y)}{kT}\right)}{1+\exp\left(-\frac{E(X)-E(Y)}{kT}\right)} = \frac{1}{1+\exp\left(+\frac{\Delta E}{kT}\right)}$$

Metropolis

$$\mathrm{p}_{acc} = \min\left\{1, \exp\left(-\frac{E(X)-E(Y)}{kT}\right)\right\}$$



- różnica występuje gdy E(X)~E(Y), wpływa to na proces generowania łańcucha Markowa (mniejsze praw. akceptacji) a dokładniej na szybkość dochodzenia układu do równowagi
- dla większych różnic energii oba wzory dają identyczne wyniki
- po osiągnięciu stanu równowagi nie ma znaczenia, którego wzoru użyjemy

Przy użyciu wzoru $T(X_{n+1}|X_n)$ oraz p_{acc} możemy wygenerować ciąg stanów w przestrzeni fazowej tzw. łańcuch Markowa

$$X_1 \to X_2 \to X_3 \to \ldots \to X_n$$

- prawdopodobieństwo wystąpienia stanu X_n zależeć będzie od f(X) i ustalonej temperatury T układu
- w ten sposób nie znając f(X) (bo nie znamy Z) przy użyciu łańcucha Markowa możemy szacować wartości wielkości fizycznych

$$\langle O \rangle = \int_\Omega O(X) f(X) dX \qquad \longrightarrow \qquad \overline{O} = \sum_{i=1}^{n} O(X_i), \qquad X_i \sim dist\{f(X)\}$$

Algorytm Metropolisa generowania łańcucha Markowa

1) aktualna postać łańcucha $\{X_1, X_2, \ldots, X_n\}$

2) wylosuj nowy stan $T(Y|X_n) \to Y$

3) określ prawdopodobieństwo akceptacji nowego stanu $p_{acc} = \min\left\{1, \exp\left(-\frac{E(Y) - E(X_n)}{kT}\right)\right\}$

4) wylosuj liczbę o rozkładzie jednorodnym $U_1 \sim U(0,1)$

5) sprawdź czy wynik jest akceptowalny
$$\begin{cases} U_1 \leq p_{acc} & \to \quad X_{n+1} = Y \\ U_1 > p_{acc} & \to \quad X_{n+1} = X_n \end{cases}$$

- jeśli nie akceptujemy nowego stanu to nowym elementem ciągu jest X_n

6) łańcuch po modyfikacji $\{X_1, X_2, \ldots, X_n, X_{n+1}\}$

- generując odpowiednio długi łańcuch oczekujemy, że dokonamy poprawnego próbkowania f(X) dla danej T inaczej: ciąg stanów będzie ergodyczny (łańcuch aperiodyczny i homogeniczny)
- elementy łańcucha Markowa są skorelowane, bo gdy nie akceptujemy nowego stanu powielamy poprzedni

Metoda symulowanego wyżarzania

Wiemy już jak osiągnąć stan równowagi termodynamicznej w określonej temperaturze - inaczej: otrzymać optymalny rozkład obsadzeń stanów danej temperaturze.

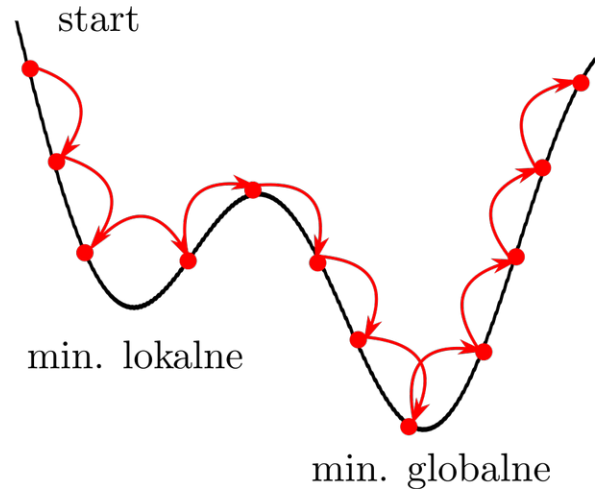
Możemy teraz wykorzystać algorytm Metropolisa w celu znalezienia stanu o najniższej energii, a jeśli energię zamienimy na dowolną inną zmienną lub zestaw zmienych to metoda ta posłuży nam do znalezienia minimum wartości funkcji.

Trzeba tylko umiejętnie dobrać temperaturę....

$$E = f(x)$$

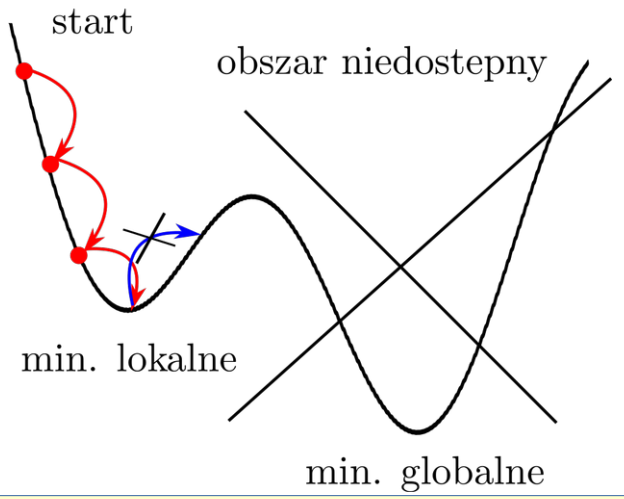
$$\mathrm{p}_{acc} = \min\left\{1, \exp\left(-\frac{f(x_{new})-f(x_{old})}{kT}\right)\right\}$$

$T >> 1 \rightarrow p_{acc}(E_{new} > E_{old}) > 0$



- jeśli T jest duże, to prawdopodobieństwo akceptacji nowego wyniku też jest duże
- „wędrowiec” może w sposób losowy odwiedzić dowolne miejsce w przestrzeni znajdując minimum

$T << 1 \rightarrow p_{acc}(E_{new} > E_{old}) \sim 0$



- małe T oznacza bardzo małe prawdopodobieństwo zaakceptowania wyniku o większej wartości funkcji celu f(x)
- ruch „wędrowca” ograniczony do lokalnej doliny f(x)
- dostęp do minimum globalnego może zostać zablokowany

Analiza przykładu prowadzi do wniosków

- im wyższa temperatura tym większy obszar dostępny dla wędrowca
- dla wysokiej temperatury wędrowiec przebywa w minimum globalnym tylko przez chwilę i w kolejnej iteracji ucieka
- w niskiej temperaturze ruch wędrowca jest silnie ograniczony przez niskie prawdopodobieństwo akceptacji miejsca, w którym funkcja ma większą wartość niż obecnie
- poszukiwanie minimum to proces stochastyczny, aby zwiększyć szansę jego znalezienia można (trzeba?) użyć większej populacji wędrowców
- proces poszukiwania minimum zaczynamy od wysokiej temperatury i stopniowo ją obniżamy, mamy nadzieję że przynajmniej jeden wędrowiec znajdzie się w okolicy minimum globalnego
- konieczne staje się wprowadzenie funkcji opisującej zmiany temperatury, to jeden z kluczowych aspektów algorytmu

Przez analogię do metody powolnego schładzania (wyżarzania) materiałów w celu likwidacji naprężeń i defektów, algorytm ten nosi nazwę metody symulowanego wyżarzania.

Przykład:

Soczewki do największych teleskopów podlegają wyżarzaniu nawet przez 2 lata
– chodzi o zmniejszenie naprężeń które powodują lokalną zmianę współczynnika załamania światła co jest jedną z przyczyn abberacji.

Sposób zmiany temperatury w metodzie SA może być kluczowy dla osiągnięcia minimum.

Najprostszy sposób to zmiana skokowa o zdaną wartość ΔT i wykonywanie IT_{max} iteracji w danej temperaturze w celu osiągnięcia równowagi termodynamicznej z otoczeniem.

inicjalizacja: T_{min}, T_{max}
M - liczba punktow posrednich temperatury
$\Delta T = \frac{T_{max} - T_{min}}{M}$

```
for (m=0; m <= M; m++){

    T = T_max - m * ΔT

    for (it=1; it<=IT_max; it++){
        jeden_krok_SA();
    }

}
```

Można też uzależnić temperaturę od numeru iteracji w sposób bezpośredni bazując na zależności np. jednomianowej

$$y = x^p, \qquad x, y \in (0,1), \quad p \in R$$

$$y = \frac{T - T_{min}}{T_{max} - T_{min}}$$

$$x = \frac{IT_{max} - it}{IT_{max}}$$

$$T = T(it) = T_{min} + (T_{max} - T_{min}) \left(\frac{IT_{max} - it}{IT_{max}} \right)^p$$

- wyrażenie opisuje temperaturę która w każdej iteracji będzie inna niż w pozostałych
- tempo zmian temperatury zależy od wartości parametru p:
  p=1 - tempo liniowe
  p>1 – tempo nieliniowe, przy czym na początku symulacji temperatura obniża się szybko, później wolniej
  p<1 – tempo nieliniowe, na początku temperatura obniża się wolno, później szybko

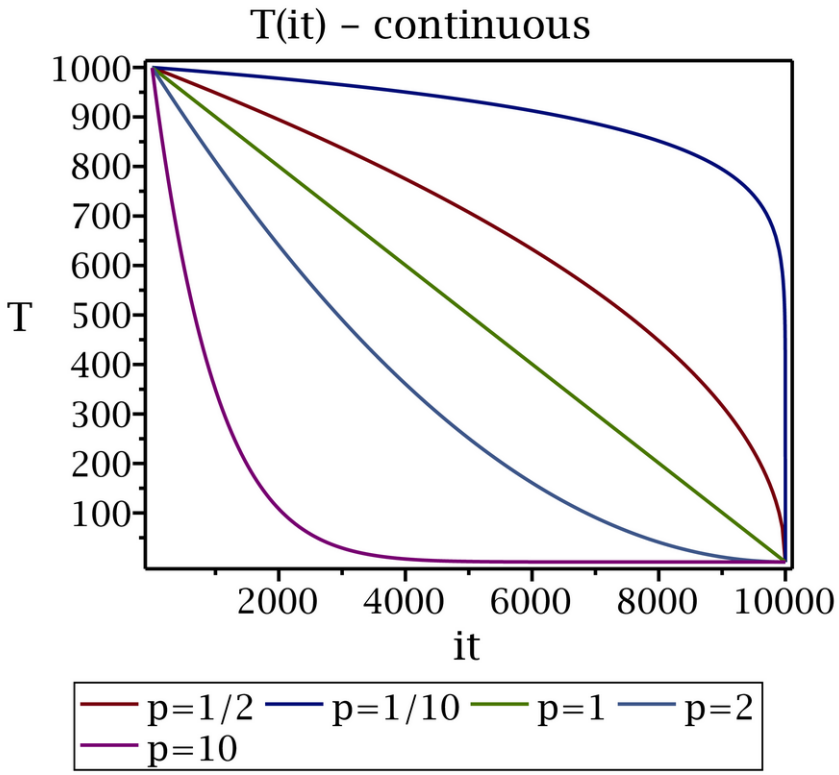
Wzór można dopasować do postaci dyskretnej, co odpowiada wykonywaniu pewnej liczby iteracji w stałej temperaturze

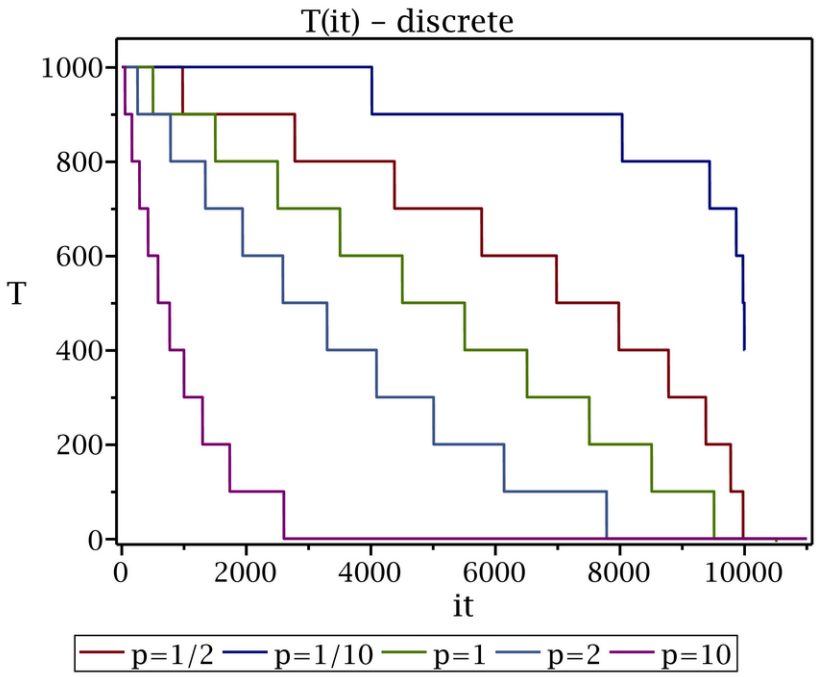
$$\Delta T = \frac{T_{max} - T_{min}}{M}$$

$$T_d = round\left(\frac{T}{\Delta T} \right) \Delta T + T_{min}$$

Przykład – zmiana temperatury w funkcji numeru iteracji

$$T_{min} = 1, \quad T_{max} = 10^3, \quad IT_{max} = 10^4$$





- punktem startowym symulacji może być zmiana liniowa (ciągła lub dyskretna)
- w zależności od przypadku, zmiana liniowa może nie być optymalna,
  czasem warto układ utrzymywać „nieco dłużej” w wyższej temperaturze,
  a później szybko schłodzić – lub odwrotnie,
  przekonać się o tym można wykonując serię symulacji testowych

Algorytm symulowanego wyżarzania

inicjalizacja: T_{min} , T_{max}

M - liczba punktow posrednich temperatury

n – liczba wymiarow

N – liczba wedrowcow

$\vec{\Delta}_{max} = [\Delta_1, \Delta_2, \ldots, \Delta_n]$

$\vec{r}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]$ – wektor startowy i–tego wedrowca

$\Delta T = \frac{T_{max} - T_{min}}{M}$

for (it =1; it <= IT_{max} ; it ++){

$T = T(it)$

for (i =1; i<= N; i++){

$\vec{\Delta} \sim dist\{U^n(\cdot,\cdot)\}$ $\qquad |\vec{\Delta}| \leq |\vec{\Delta}_{max}|$

$\vec{r}_i^{new} = \vec{r}_i + \vec{\Delta}$

$p_{acc} = \min\left\{1, \exp\left(-\frac{f(\vec{r}_i^{new}) - f(\vec{r}_i)}{kT}\right)\right\}$

$U_1 \sim U(0,1)$

if ($U_1 \leq p_{acc}$) {

$\vec{r}_i \leftarrow \vec{r}_i^{new}$

}else{

$\vec{r}_i$ – bez zmian

}

}

}

Modyfikacja SA – metoda stochastycznego tunelowania

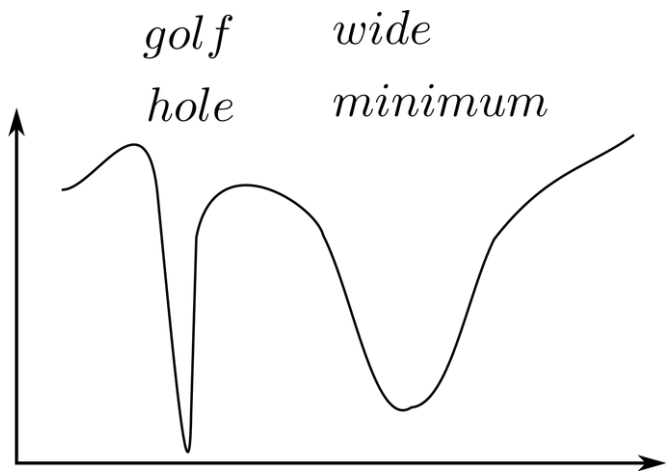
Często zdarza się że algorytm prowadzi nas do lokalnego głębokiego minimum o charakterystyce dołka golfowego (ang. „golf hole”). W takim przypadku najczęściej pojawia się stagnacja rozwiązania - niewielka zmiana położenia prowadzi do dużej zmiany (wzrostu) wartości funkcji celu i znacznie ogranicza prawdopodobieństwo akceptacji nowego położenia (p_{acc} jest ograniczane dodatkowo przez malejącą temperaturę).

Aby rozwiązać ten problem można użyć modyfikacji polegającej na modyfikacji funkcji celu z powodu podobieństwa do sposobu w jaki cząstka tuneluje przez barierę potencjału modyfikacja nazywana jest **tunelowaniem stochastycznym**

$$f_{stun}(x) = 1 - \exp\{-\gamma[f(x) - f(x_{min})]\}, \qquad \gamma > 0$$

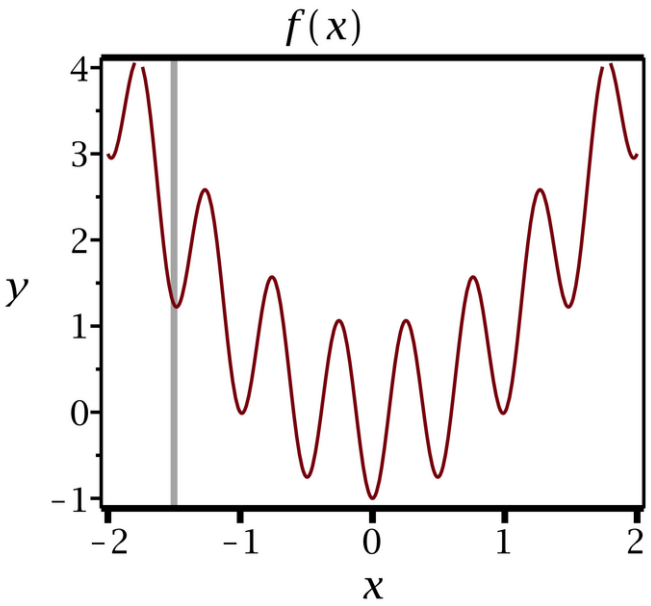
f(x) – pierwotna funkcja celu

x_{min} – znaleziony punkt, w którym funkcja osiąga najmniejszą wartość (lokalne minimum)

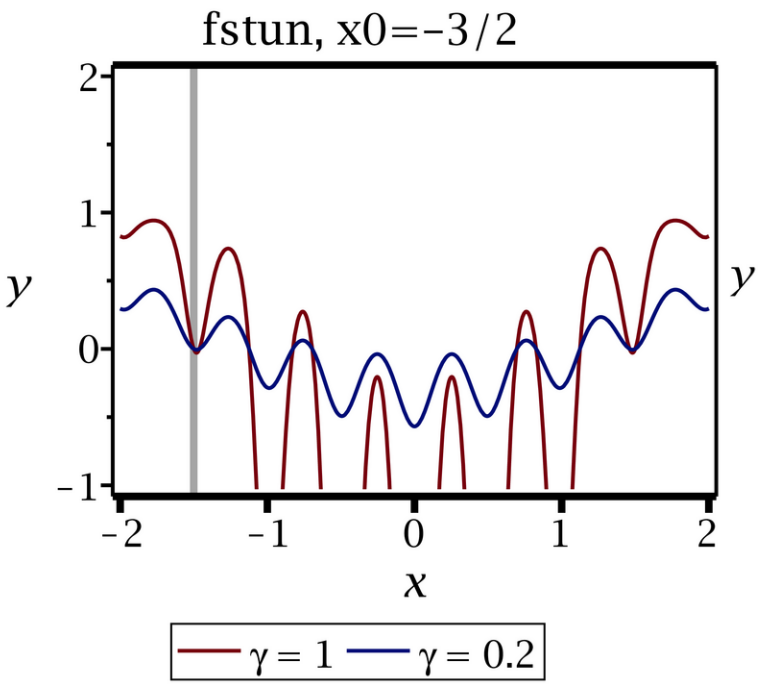


przykład - funkcja celu dana wzorem

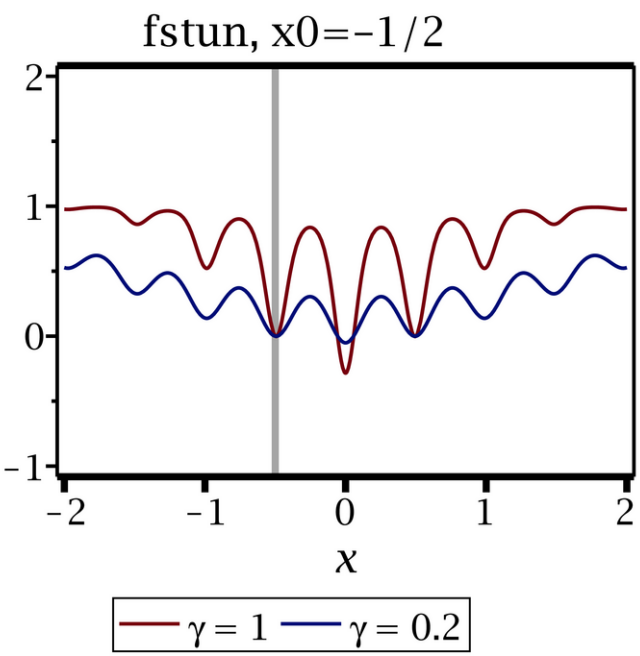
$$f(x) = x^2 - \cos(4\pi x)$$



- pierwsze minimum lokalne w x=-1.5
- dołek głęboki więc możemy utknąć



- stosujemy tunelowanie stochastyczne
- parametr γ skaluje wysokość ścianek dołka



- jeśli wydostaniemy się z pierwszego znalezionego minimum to kontynuujemy poszukiwania stosując modyfikację dla kolejnych minimów

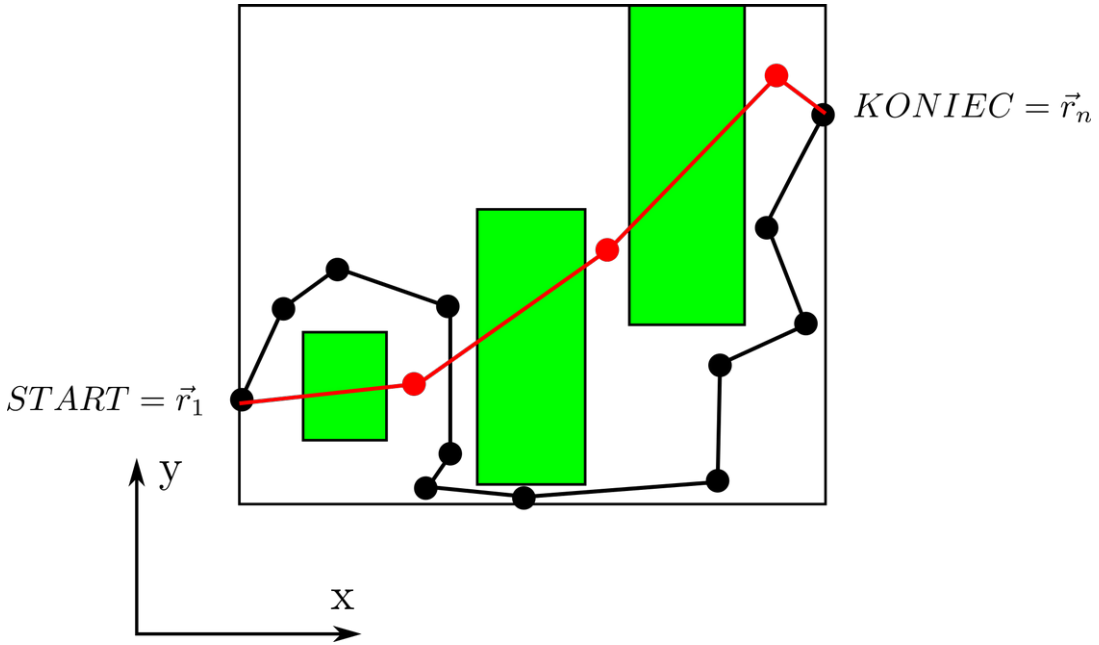
$$f_{stun}(x) = 1 - \exp\{-\gamma[f(x) - f(x_{min})]\}, \qquad \gamma > 0$$

Przykład. Poszukiwanie najkrótszej trasy w mieście.

Wykorzystajmy metodę SA do znalezienia najkrótszej drogi łączącej dwa punkty w mieście

Założenia:

- algorytm powinien uwzględniać konieczność omijania budynków
- punkty: startowy i końcowy są ustalone (wejście/wyjście)
- liczba punktów pośrednich określana jest przed symulacją (możemy ją zmieniać, dopasować do problemu)
- położenie punktów pośrednich będziemy zmieniać przy użyciu algorytmu SA - punkty swobodne
- w symulacji użyjemy populacji N niezależnych wędrowców



Uwagi do rysunku:

- obszary zielone to budynki
- czarna krzywa łamana to trasa bliska optymalnej (taką moglibyśmy zaakceptować)
  zawiera dużą liczbę punktów swobodnych
- czerwona krzywa łamana to trasa nieakceptowalna bo przechodzi przez budynki
  zawiera małą liczbę punktów swobodnych
- im więcej krzywa posiada punktów swobodnych tym łatwiej jest jej ominąć budynki, ale jednocześnie zwiększa to liczbę wymiarów opisujących problem

Konstruujemy funkcję celu/kosztu

- argumentami muszą być punkty stanowiące krańce odcinków łączących wejście z wyjściem

$$f = f(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_{n-1}, \vec{r}_n)$$

$\vec{r}_1, \vec{r}_n$ - parametry modelu

$\vec{r}_2, \vec{r}_3, \ldots, \vec{r}_{n-1}$ - zmienne (stopnie swobody)

ograniczenia przestrzenne $x_i \in [x_{min}, x_{max}]$
$y_i \in [y_{min}, y_{max}]$

wymiarowość problemu $dim\{\vec{r}_2, \vec{r}_3, \ldots, \vec{r}_{n-1}\} = 2 \cdot (n-2)$

Ponieważ trasa składa się z odcinków, więc najłatwiej skonstruować funkcję celu w postaci sumy wkładów wnoszonych przez poszczególne odcinki

$$f = \sum_{i=1}^{n-1} d_{i,i+1}$$

- jeśli trasa przechodzi pomiedzy budynkami to f określa jej długość
- najmniejsza wartość f powinna opisywać najkrótszą trasę
- jeśli odcinek przecina budynek to oprócz jego długości współczynnik $d_{i,i+1}$ powienien zawierać dodatkowy nieujemny czynnik stanowiący karę,
  to spowoduje że wartość funkcji celu bardzo wzrośnie i takie rozwiązanie będzie odrzucane lub akceptowane z małym prawdopodobieństwem

- wkład danego odcinka z możliwą karą za przejście przez budynek np. w postaci potencjału schodkowego

$$d_{i,i+1} = \int_{\vec{r}_i}^{\vec{r}_{i+1}} [1 + V(\vec{r})]\, ds, \qquad ds = \sqrt{dx^2 + dy^2}$$

$$V(\vec{r}) = \begin{cases} 0, & \text{empty field} \\ V_{max} >> 1, & \text{building} \end{cases}$$

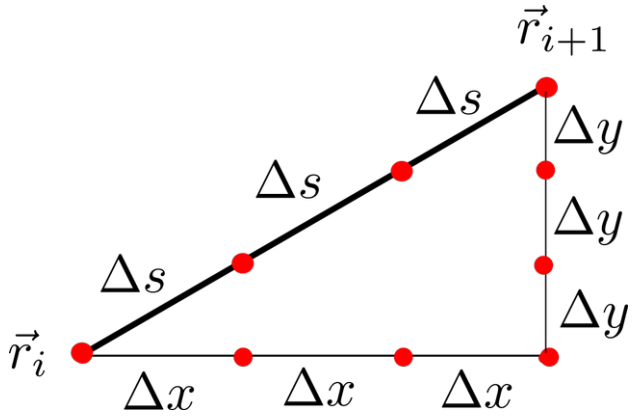
Całkowanie wykonajmy numerycznie przy użyciu metody trapezów dla (K+1) węzłów
- takie podejście jest dość ogólne i pozwala na łatwą zmianę potencjału kary

$$\Delta x = \frac{x_{i+1} - x_i}{K} \qquad \Delta y = \frac{y_{i+1} - y_i}{K}$$

$$x_\mu = x_i + \mu \cdot \Delta x, \qquad \mu = 0, 1, \ldots, K$$

$$y_\mu = y_i + k \cdot \Delta y, \qquad \mu = 0, 1, \ldots, K$$

$$\vec{r}_\mu = [x_\mu, y_\mu]$$



$$d_{i,i+1} = \int_{\vec{r}_i}^{\vec{r}_{i+1}} [1 + V(\vec{r})]\, ds \approx \sum_{\mu=0}^{K} c_\mu \, [1 + V(\vec{r}_\mu)] \cdot \Delta s$$

$$\Delta s = \sqrt{\Delta x^2 + \Delta y^2}$$

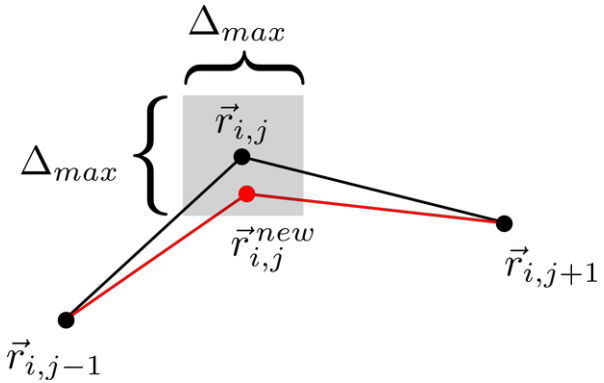
$$c_\mu = \begin{cases} \frac{1}{2} & \Longleftrightarrow \quad \mu = 0, K \\ 1 & \Longleftrightarrow \quad \mu = 2, 3, \ldots, K-1 \end{cases}$$

Uwaga: spodziewamy się że tylko część odcinka Δs może przechodzić przez budynek, co prowadzi do powstawiania błędu całkowania, ale jeśli K jest duże (np. K=100) to błąd ten staje się mały, pamietajmy także że naszym celem jest trasa omijająca budynki – wówczas błąd całkowania będzie znikał (bo potencjał kary znika)

- konstruujemy tablicę danych zawierającą informacje o trasach N wędrowców k - w niej będziemy dokonywać zmian

$$\vec{f} = \begin{array}{|c|c|c|c|c|c|} \hline \vec{r}_1 & \vec{r}_{1,2} & \vec{r}_{1,3} & \dots & \vec{r}_{1,n-1} & \vec{r}_n \\ \hline \vec{r}_1 & \vec{r}_{2,2} & \vec{r}_{2,3} & \dots & \vec{r}_{2,n-1} & \vec{r}_n \\ \hline \vec{r}_1 & \vdots & \vdots & \vdots & \vdots & \vec{r}_n \\ \hline \vec{r}_1 & \vec{r}_{N,2} & \vec{r}_{N,3} & \dots & \vec{r}_{N,n-1} & \vec{r}_n \\ \hline \end{array}$$

- zmiany położeń punktów swobodnych wykonujemy losowo w obu kierunkach nie przekraczając wyznaczonych granic obszaru Ω



Δ_{max} – ustalone
$U_1, U_2 \sim U(0,1)$
$\delta x = (2U_1 - 1)\Delta_{max}$
$\delta y = (2U_2 - 1)\Delta_{max}$
$\vec{r}^{new}_{i,j} = \vec{r}_{i,j} + [\delta x, \delta y]$
$\Omega = [x_{min}, x_{max}] \times [y_{min}, y_{max}]$
if $(\vec{r}^{new}_{i,j} \in \Omega)\{$
$p_{acc} = \min\left\{1, \exp\left(-\beta[f(\vec{r}^{new}_{i,j}) - f(\vec{r}_{i,j})]\right)\right\} = \min\left\{1, \exp\left(-\beta[d^{new}_{j-1,j} + d^{new}_{j,j+1} - d_{j-1,j} - d_{j,j+1}]\right)\right\}$
$U_3 \sim U(0,1)$
if $(U_3 \leq p_{acc})\{$
$\vec{r}_{i,j} \leftarrow \vec{r}^{new}_{i,j}$
$\}$ else $\{$
$\vec{r}_{i,j}$ – bez zmian
$\}$
$\}$

$$d_{j,j+1} = \int_{\vec{r}_j}^{\vec{r}_{j+1}} [1 + V(\vec{r})]\, ds$$

Optymalizacja Monte Carlo

Parametry symulacji:

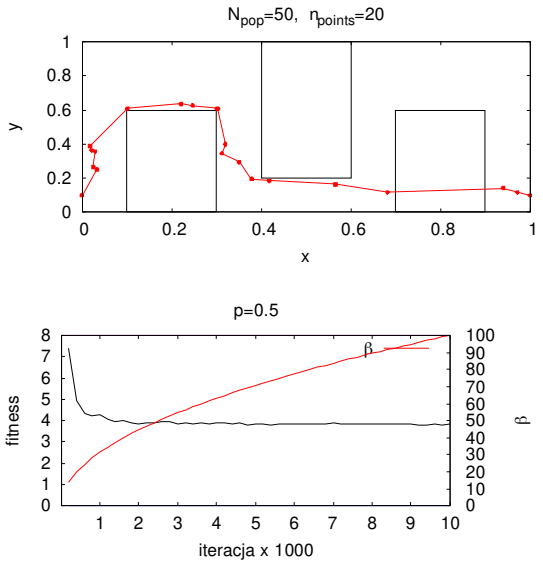
$$\beta_{min} = 10^{-6}, \quad \beta_{max} = 10^2$$

$$N = 50, \quad n = 20, \quad IT_{max} = 10^4$$

$$\Omega = [0,1] \times [0,1]$$

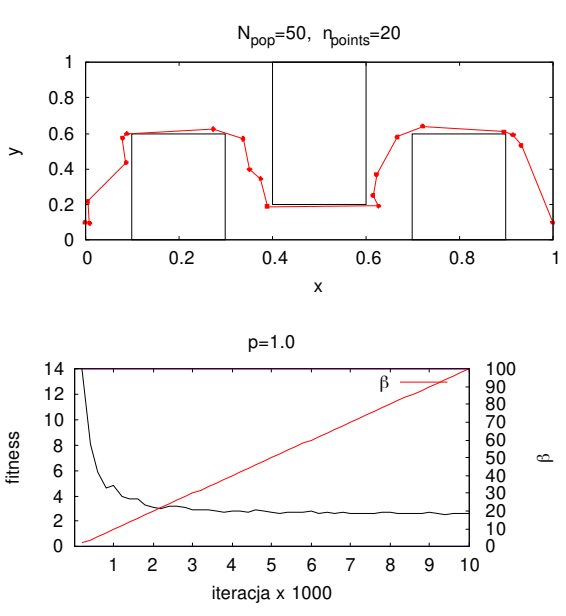
$$\Delta_{max} = 0.03$$

$$p = \frac{1}{2}, 1, 2, 6$$

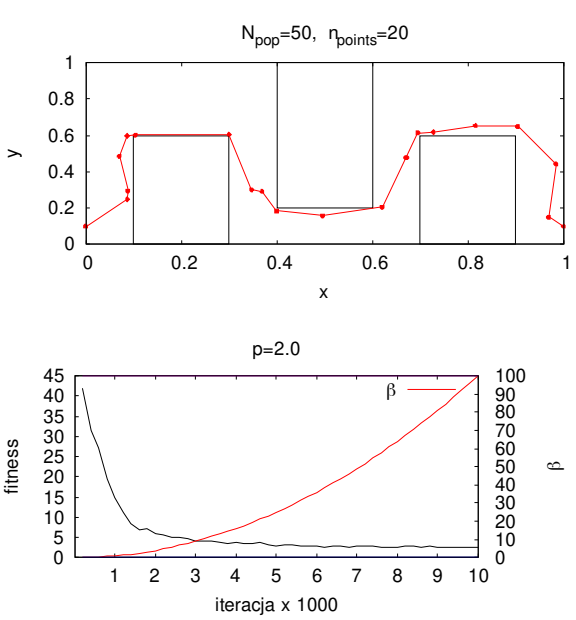


tu optymalizacja ścieżki nie powiodła się, ponieważ wartość β rośnie zbyt szybko co obniża p_{acc}, a trzeba przesunąć odcinek (2 punkty)

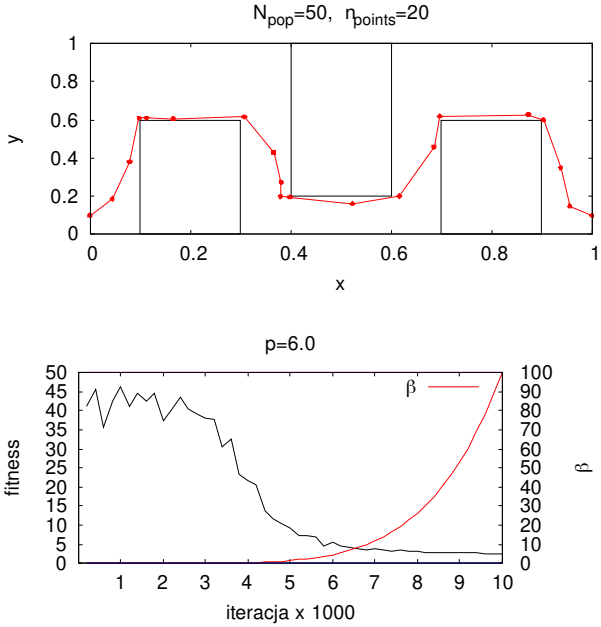
$$f_{p=1/2} = 3.82$$



$$f_{p=1} = 2.601$$



$$f_{p=2} = 2.617$$



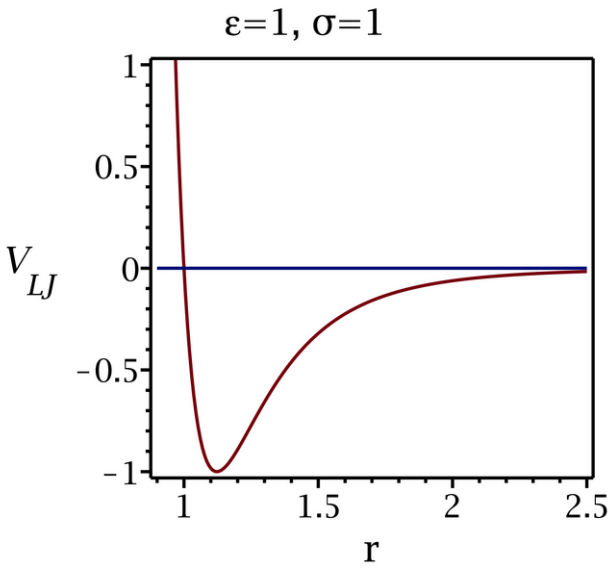
$$f_{p=6} = 2.575$$

Przykład. Problem Thompsona – należy znaleźć konfigurację przestrzenną N atomów rozłożonych na sferze.
Atomy oddziałują ze sobą potencjałem Lennarda-Jonesa.
Szukamy najstabilniejszych konfiguracji – te mają najmniejszą energię całkowitą (największą energię wiązania)
- do znalezienia minimalnej energii użyjemy metoda SA.

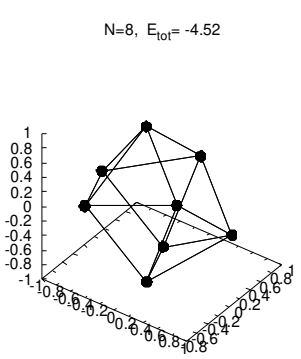
Potencjał Lennarda-Jonesa opisuje oddziaływanie dwóch cząsteczek wynikające z deformacji ich rozkładów gęstości elektronowej.

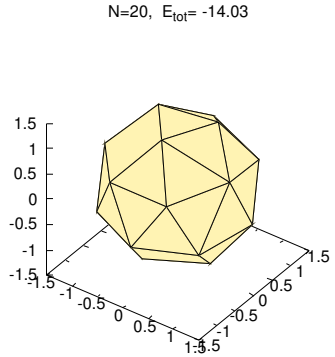
$$V_{LJ}(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

$$\min V_{LJ}(r) = -\varepsilon \quad \Longleftrightarrow \quad r_{min} = 2^{\frac{1}{6}}\sigma = 1.12\sigma$$



- potencjał jest izotropowy
- krótkozasięgowy wyraz odpychający nie pozwala zbliżyć się cząsteczkom bo prowadzi to do gwałtownego wzrostu energii potencjalnej
- wyraz przyciągający też jest krótkozasięgowy (w porównaniu do oddziaływania kulombowskiego)





- przykładowe konfiguracje klasterów atomowych

Konstrukcja algorytmu SA dla problemu Thomsona

1) wybór układu współrzędnych

- cząstki położone są na sferze, więc wybór współrzędnych sferycznych jest naturalny
- jednak oddziaływanie wygodniej liczy się we współrzędnych kartezjańskich
- będziemy elastyczni i użyjemy obu typów współrzędnych

$$
\begin{aligned}
x &= r \sin\theta \cos\phi \\
y &= r \sin\theta \sin\phi \\
z &= r \cos\theta
\end{aligned}
$$

2) konfiguracja początkowa/startowa

- początkowe położenia cząstek możemy zadać zgodnie z jakąś regułą
  - wówczas może się okazać, że liczba iteracji potrzebna do znalezienia konfiguracji optymalnej będzie krótka

- najłatwiej konfigurację wylosować

$$
\begin{cases}
i = 1, 2, \ldots, N \\
r_i = r_{init} \\
\theta_i = \pi \cdot U_1, & U_1 \sim U(0,1) \\
\phi_i = 2\pi \cdot U_2, & U_2 \sim U(0,1)
\end{cases}
$$

3) zmiana konfiguracji przestrzennej

- wystarczy zmiana położenia jednego atomu aby zmieniła się energia całkowita (i położenie w przestrzeni fazowej)

- w jednej iteracji będziemy próbować zmieniać położenia wszystkich atomów
  - indywidualnie (r,θ, ϕ)
  - całego zbioru cząstek (tylko: r)

- **Indywidualne przesunięcia cząstek**

$$w_r, w_\phi, w_\theta << 1$$
$$U_1, U_2, \ldots \sim U(0,1)$$
$$\delta r = r \cdot (2U_1 - 1) w_r$$
$$\delta_\phi = 2\pi \cdot (2U_2 - 1) w_\phi$$
$$\delta_\theta = \pi \cdot (2U_3 - 1) w_\theta$$
$$\phi^{new} = (2\pi + \phi + \delta_\phi) \mod 2\pi$$
$$\theta^{new} = \begin{cases} \theta + \delta_\theta \in [0, \pi] \\ \theta \iff \theta + \delta_\theta \notin [0, \pi] \end{cases}$$

- przesunięcia generujemy sekwencyjnie dla każdego z kierunków z rozkładem jednorodnym i symetrycznym

- współczynniki **w** określają maksymalne względne przesunięcia

  - małe wartości w spowodują, że prawdopodobieństwo akceptacji będzie duże (p_{acc}**~1**), ale konfiguracja będzie zmieniać się bardzo wolno

  - duże wartości w, spowodują że prawdopodobieństwo akceptacji będzie małe (p_{acc}**«1**), co też doprowadzi do bardzo powolnych zmian konfiguracji

  - zasadą jest aby dla ustalonej temperatury: p_{acc}**~0.5** tj. należy dążyć do tej wartości, bo daje to kompromis pomiędzy dwoma powyższymi skrajnymi przypadkami

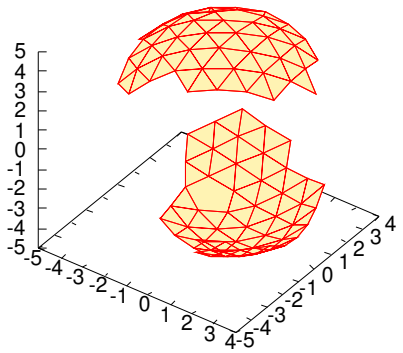
- zakres kąta azymutalnego ϕ ograniczamy do zakresu [0,2π], mimo iż x i y zależą od niego periodycznie, to zabezpieczamy się przed (mało prawdopodobną) sytuacją że jego amplituda stale narasta co może doprowadzić do przekroczenia zakresu reprezentującej go zmiennej

- kąt biegunowy θ nie jest periodyczny, więc jeśli przekroczy dolny lub górny zakres to zgodnie z algorytmm Metropolisa zostawimy starą wartość (prawdopodobnie r i ϕ zostaną zmienione więc przypadek braku zmiany konfiguracji będzie rzadki)

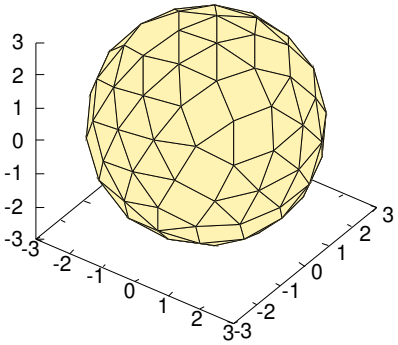
- **globalna zmiana konfiguracji - PO CO?**

  - jeśli nic nie wiemy o docelowej konfiguracji przestrzennej, to nie wiemy jaki promień końcowy będzie miała sfera
  - zazwyczaj startujemy od małego promienia i pozwalamy go zwiększać po to aby cząstki oddaliły się od siebie obniżając energię całkowitą
  - pozwalając cząstkom samodzielnie zmieniać swój promień, możemy dopuścić do sytuacji, w której część cząstek szybko oddali się od centrum pozostawiając resztę – takie postępowanie spowoduje deformację klastra tj. kształt będzie znacznie różnił się od sferycznego
  - globalna zmiana promienia zwiększa zbieżność metody (wymaga mniej iteracji), ponadto cząstki będą oddalały się od siebie w sposób bardziej równomierny

N=100, r_{init}=5, r_{final}=4.94, E_{tot}= -66.32



N=100, r_{init}=1, r_{final}=2.99, E_{tot}= -76.23



- za duży promień początkowy „rozerwał” klaster
- atomy w dwóch powstałych połówkach przyciągają się więc mogą to być konfiguracje stabilne

- start od małego promienia pozwolił sukcesywnie zwiększać objętość bez „rozrywania” klastera
- energia końcowa niższa niż w przypadku rozdzielenia na dwie części

initialization: r_{init}, p, β_{min}, β_{max}, IT_{max}, w_r, w_ϕ, w_θ, W_{all}

generate positions: $\vec{r} = [r_{init}, \phi_i, \theta_i], \qquad i = 1, 2, \ldots, N$

for (it=1; it <= IT_{max}; it++){

$\beta = \beta(it)$

********** individual changes **********

for (i=1; i <= N; i++){

$$E_i^{old} = \frac{1}{2} \sum_{j \neq i}^{N} V_{ij}(|\vec{r}_i - \vec{r}_j|)$$

$$\vec{r}_i^{new} \leftarrow \vec{r} + \delta\vec{r}$$

$$E_i^{new} = \frac{1}{2} \sum_{j \neq i}^{N} V_{ij}(|\vec{r}_i^{new} - \vec{r}_j|)$$

$$p_{acc} = \min\left\{1, \exp\left(-\beta(E_i^{new} - E_i^{old})\right)\right\}$$

if ($p_{acc} \geq U_1$){ $\qquad (U_1 \sim U(0,1))$

$\vec{r}_i \leftarrow \vec{r}_i^{new}$

}else{

$\vec{r}_i \leftarrow \vec{r}_i$

}

}

********** global change **********

$$E^{old} = \frac{1}{2} \sum_{i=1}^{N} \sum_{j \neq i}^{N} V_{ij}(|\vec{r}_i - \vec{r}_j|)$$

$$i = 1, 2, \ldots, N: \qquad r_i^{new} = r_i \cdot [1 + (2U_2 - 1)W_{all}] \qquad (U_2 \sim U(0,1))$$

$$E^{new} = \frac{1}{2} \sum_{i=1}^{N} \sum_{j \neq i}^{N} V_{ij}(|\vec{r}_i^{new} - \vec{r}_j^{new}|)$$

$$p_{acc} = \min\left\{1, \exp\left(-\beta(E_i^{new} - E_i^{old})\right)\right\}$$

if ($p_{acc} \geq U_3$){ $\quad (U_3 \sim U(0,1))$

$r_i \leftarrow r_i^{new}, \qquad i = 1, 2, \ldots, N$

}else{

$r_i \leftarrow r_i, \qquad i = 1, 2, \ldots, N$

}

}

Jak będziemy analizować wyniki? a dokładniej jakie wielkości będą istotne?

- **energia całkowita** powinna być ujemna jeśli klaster ma być stabilny
- **energia wiązania na atom** pozowli porównywać stabilność klasterów o różnej liczbie atomów

$$E_B = \frac{|E_{tot}|}{N}$$

- **średni promień klastera** – co prawda dopuszczamy niezależne zmiany położeń atomów kierunku radialnym, ale po ich uśrednieniu dostaniemy parametr charakteryzujący wielkość klastera
- **średnia odległość do najbliższego sąsiada** – potencjał Lennarda-Jonesa posiada dobrze zdefiniowane minimum, możemy sobie zadać pytanie: jaka będzie relacja odległości do najbliższego sąsiada i r_{min} potencjału V_{LJ} ?

Trzy pierwsze parametry: E_{tot}, E_B i r_{sr} łatwo policzymy.

Ale jak określić, który atom jest najbliższym sąsiadem? Jakie przyjąć kryterium?

W tym celu wykorzystamy **funkcję korelacji par** – określa ona gęstość prawdopodobieństwa znalezienia atomu j-tego w odległości r od atomu i-tego.

Funkcja korelacji par (ang. pair correlation function – PCF, radial distribution function - RDF)

Funkcję tę wykorzystuje się najczęściej w modelowaniu płynów (gazów i cieczy), ponieważ pozwala w prosty sposób zobrazować korelacje przestrzenne w układzie.

Definicja

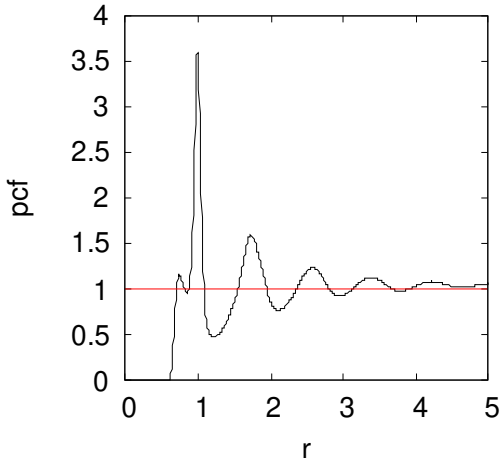
$$g(r) = \frac{2\Omega}{N^2} \left\langle \sum_{i=1}^{N} \sum_{j>i}^{N} \delta\left(r - |\vec{r_{ij}}|\right) \right\rangle_t$$

gdzie: Ω – n-wymiarowa objętość układu, N – liczba cząstek w układzie, δ – delta Diraca

Ponieważ wyraża ona fgp, można ją wykorzystać do wyznaczania wartości oczekiwanych, np. dla 3 wymiarów w przypadku izotropowym (brak zależności kątowych)

$$3D \quad \longrightarrow \quad \langle A \rangle = \left\langle \sum_{i} \sum_{j>i} A(r_{ij}) \right\rangle = \frac{N^2}{2\Omega} \int_0^\infty A(r) g(r) 4\pi r^2 dr$$

PCF wody:
główny pik – wiązanie wodorowe
pik na zboczu – oddziaływanie LJ



Zgodnie z definicją uśrednianie odbywa się w pewnym przedziale czasu, co wynika z faktu że w układzie dynamicznym położenia cząstek podlegają ciągłym zmianom (**fluktuacje**). Obserwacja prowadzona w odpowiednio długim przedziale czasu pozwala zredukować fluktuacje - czyli sytuacja typowa dla metody Monte Carlo, fluktuacje te w naturalny sposób pojawiają się jednak także w **dynamice molekularnej**, ich przyczyną jest stochastyczny charakter oddziaływań w układzie zawierającym dużą liczbę cząstek.

Definicję musimy zmienić – wzór jest dobry tylko do rozważań teoretycznych, deltę Diraca musimy zastąpić funkcją rozmytą przestrzennie. Zamiast niej użyjemy histogram.

Histogram dla funkcji korelacji par

- przyjmujemy maksymalny promień do liczenia odległości
  (większy niż r_{sr} klastera bo może być wydłużony/zdeformowany)

$$r_{sr} = \frac{1}{N}\sum_{i=1}^{N} r_i \qquad r_{max} = 2.5 \cdot r_{sr}$$

$$\begin{array}{lccccccccc} m = & 0 & & 1 & & 2 & & 3 & & M \\ r = & 0 & & \Delta & & 2\Delta & & 3\Delta & & r_{max} \end{array}$$

- dzielimy r_{max} na M przedziałów

$$\Delta r = \frac{r_{max}}{M}$$

środek przedziału

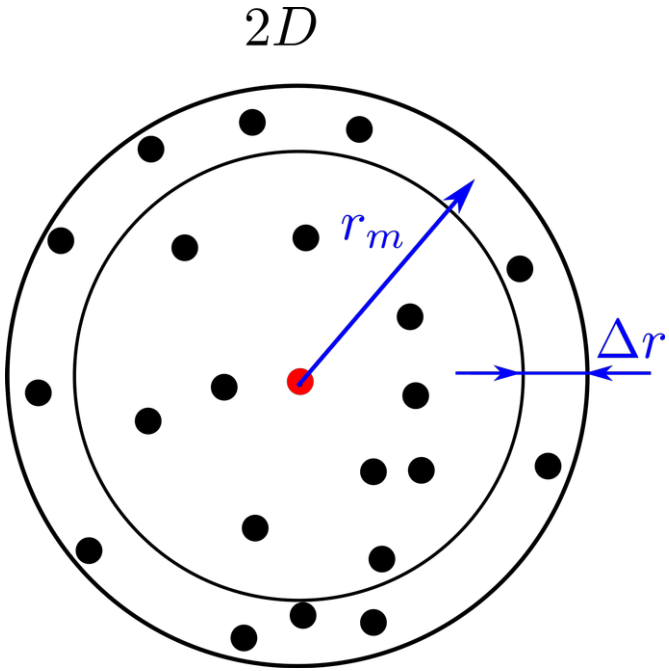
$$r_m = \left(m + \frac{1}{2}\right) \cdot \Delta r, \quad m = 0, 1, 2, \ldots, M$$

- funkcję pcf zapiszemy w postaci unormowanego histogramu dla 2 wymiarów
  (interesuje nas powierzchnia klastera/sfery)

$$pcf[m] = \frac{2\Omega}{n^2}\sum_{i=1}^{n}\sum_{j>i}^{n}\frac{\delta_{m,k}}{\Delta\Omega}$$

$$\delta_{m,k} = \begin{cases} 1, & gdy \quad k = m \\ 0, & gdy \quad m \neq k \end{cases}$$

$$k = floor\left(\frac{r_{ij}}{\Delta r}\right)$$



powierzchnia sfery: $\Omega = 4\pi r_{sr}^2$

powierzchnia pierścienia: $\Delta\Omega = 2\pi r_m \Delta r$

pseudokod do generowania histogramu

definiujemy: M, pcf[M]

$r_{sr} = \frac{1}{n}\sum_i r_i$

$r_{max} = 2.5 \cdot r_{sr}$

$\Delta r = \frac{r_{max}}{M}$

for (i=1; i <= N; i++){

  for (j=i+1; j <=N; j++){

    $r = r_{ij}$

    $m = floor\left(\frac{r}{\Delta r}\right)$

    $r_m = \left(m + \frac{1}{2}\right)\Delta r$

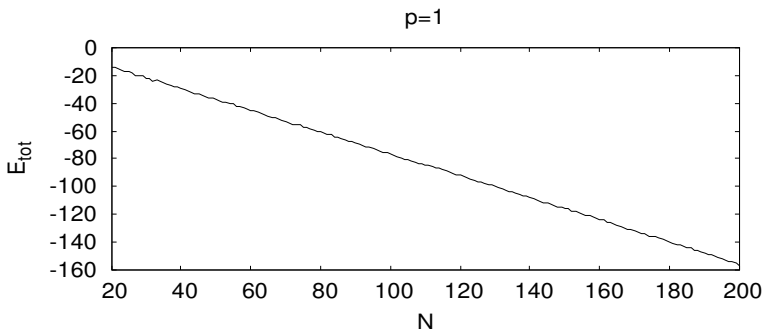
    if (m < M) $pcf[m] = pcf[m] + \frac{2 \cdot 4\pi r_{sr}^2}{N^2 2\pi r_m \Delta r}$

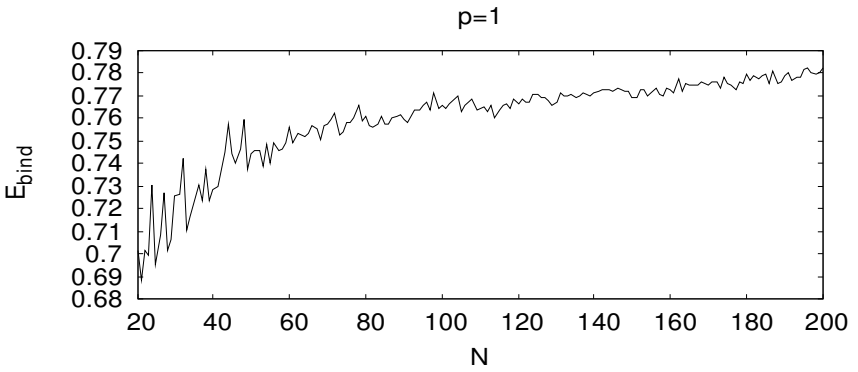
  }

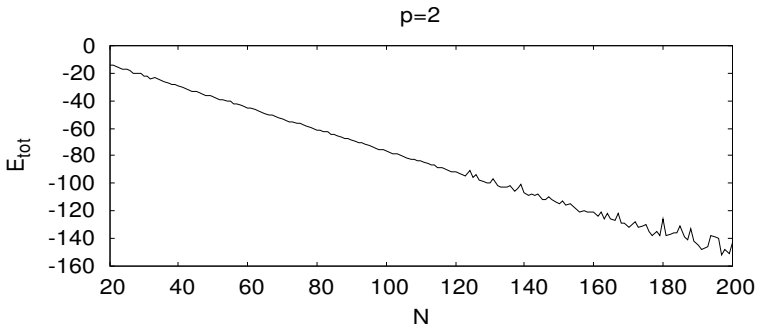
}

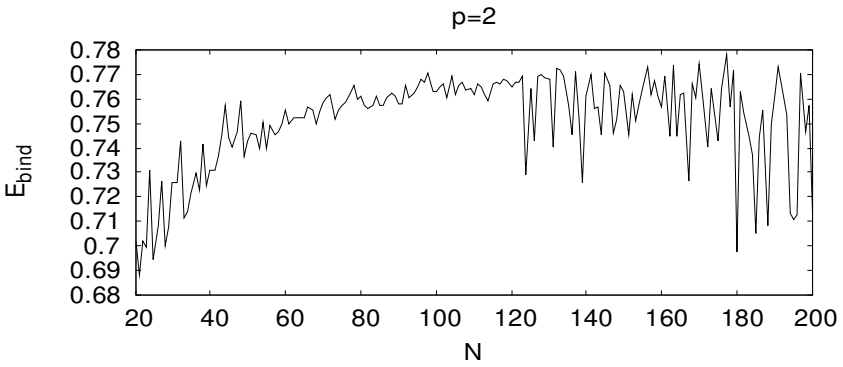
Wyniki symulacji dla parametrów

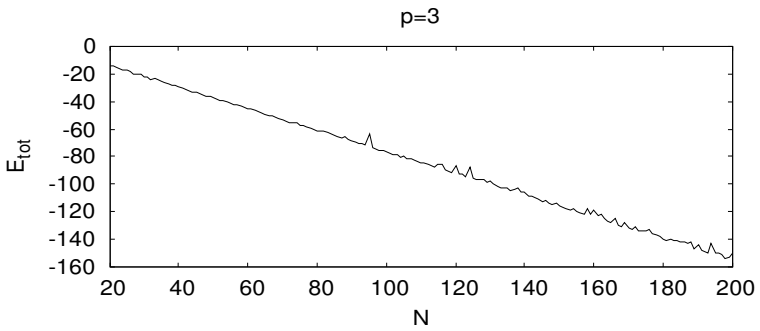
$$
\begin{aligned}
N &= 20 \div 200 \\
\varepsilon &= 1 \\
\sigma &= 1 \\
r_{init} &= 1 \\
\beta_{min} &= 10^{-4} \\
\beta_{max} &= 10^{+4} \\
p &= 1 \\
w_r &= 10^{-2} \\
w_\phi &= 10^{-2} \\
w_\theta &= 10^{-2} \\
W_{all} &= 10^{-4}
\end{aligned}
$$

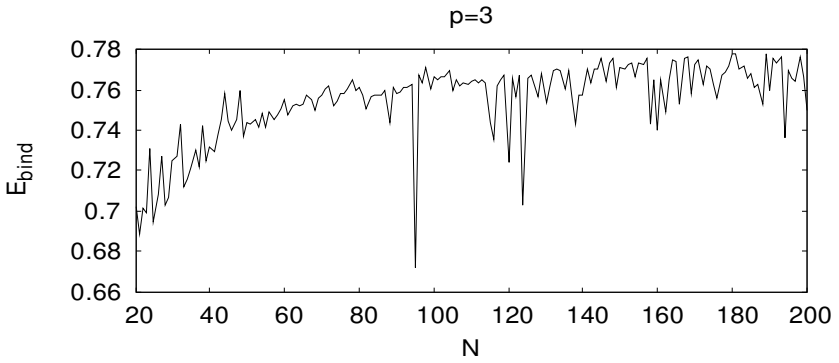

p=1
E_{tot}
0
-20
-40
-60
-80
-100
-120
-140
-160
20
40
60
80
100
120
140
160
180
200
N

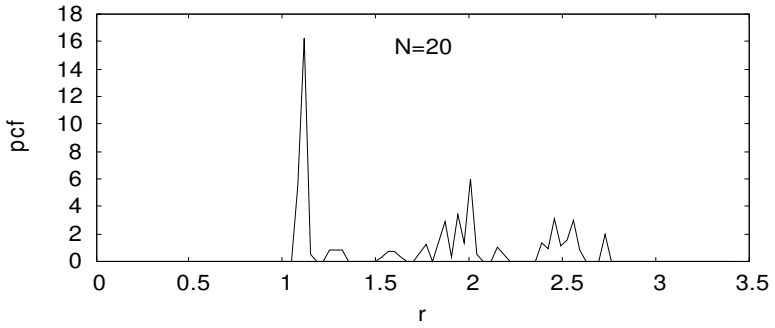

p=1
E_{bind}
0.79
0.78
0.77
0.76
0.75
0.74
0.73
0.72
0.71
0.7
0.69
0.68
20
40
60
80
100
120
140
160
180
200
N

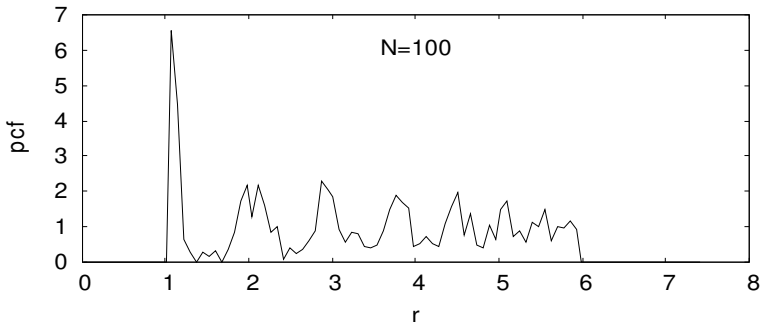

p=2
E_{tot}
0
-20
-40
-60
-80
-100
-120
-140
-160
20
40
60
80
100
120
140
160
180
200
N

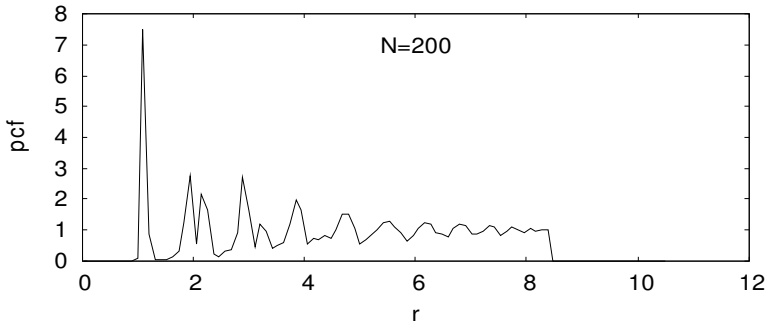

p=2
E_{bind}
0.78
0.77
0.76
0.75
0.74
0.73
0.72
0.71
0.7
0.69
0.68
20
40
60
80
100
120
140
160
180
200
N


p=3
E_{tot}
0
-20
-40
-60
-80
-100
-120
-140
-160
20
40
60
80
100
120
140
160
180
200
N


p=3
E_{bind}
0.78
0.76
0.74
0.72
0.7
0.68
0.66
20
40
60
80
100
120
140
160
180
200
N





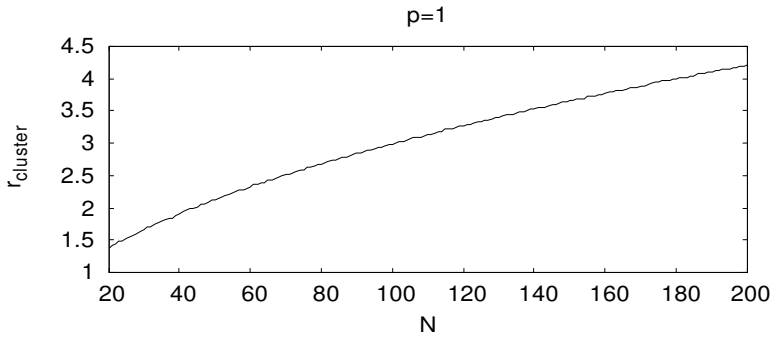


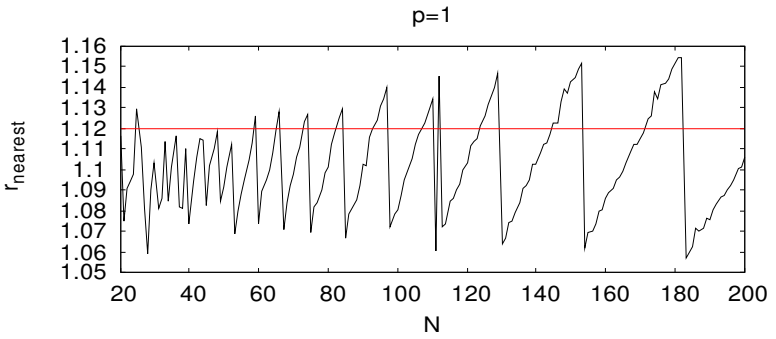
- uwaga: skale osi r na wykresach są różne ze wzgędu na różne wielkości klasterów
- na wykresach PCF łatwo odnaleźć najwyższy pik, jego maksimum jest bliskie minimum potencjału Lennarda-Jonesa

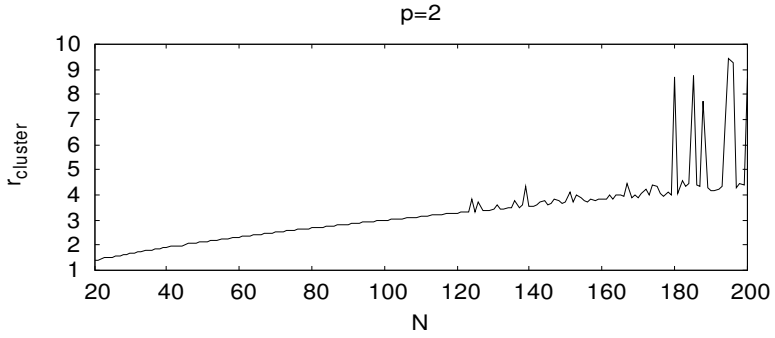
$$r_{max} \sim 1.12$$

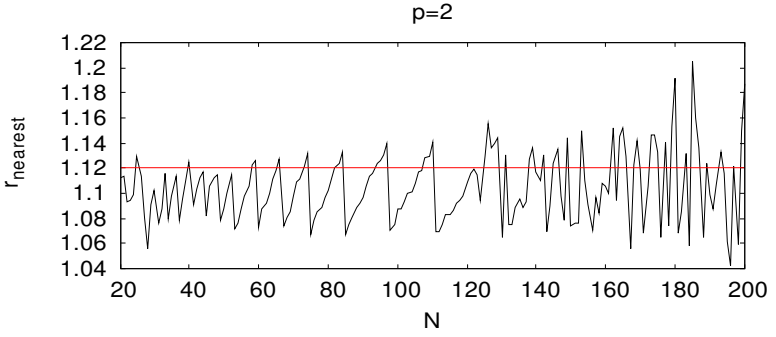
- pik ten odpowiada najbardziej prawdopodobnej odległości dwóch najbliższych sąsiadów
- histogram posiada też kilka niższych pików, te pokazują w jakiej odległości możemy spodziewać się kolejnych sąsiadów (następny najbliższy sąsiad)
- dla małej liczby N pcf maleje do zera po wystąpieniu pierwszego piku, natomiast dla N=200 i dużych r pcf dąży do wartości 1 jak dla układu jednorodnego

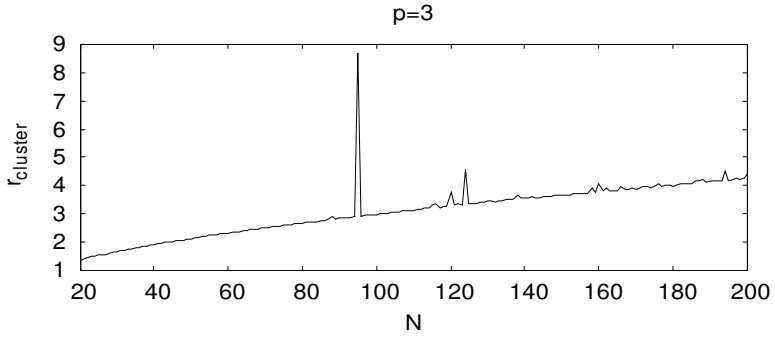
Mimo, iż pcf zastosowaliśmy do układu o niewielkiej liczbie cząstek to pozwoliło to określić najbardziej pradwopodobną odległość do najbliższego sąsiada.

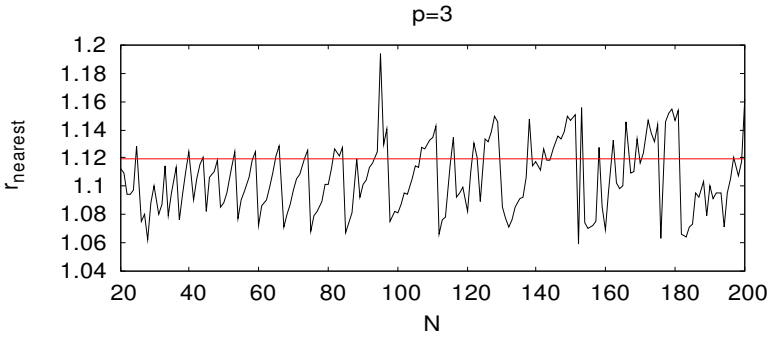

p=1
r_cluster
N
4.5
4
3.5
3
2.5
2
1.5
1
20 40 60 80 100 120 140 160 180 200


p=1
r_nearest
N
1.16
1.15
1.14
1.13
1.12
1.11
1.1
1.09
1.08
1.07
1.06
1.05
20 40 60 80 100 120 140 160 180 200

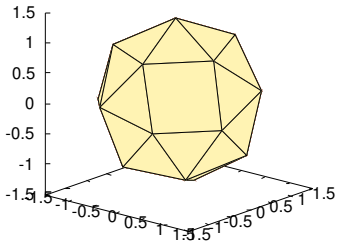

p=2
r_cluster
N
10
9
8
7
6
5
4
3
2
1
20 40 60 80 100 120 140 160 180 200


p=2
r_nearest
N
1.22
1.2
1.18
1.16
1.14
1.12
1.1
1.08
1.06
1.04
20 40 60 80 100 120 140 160 180 200

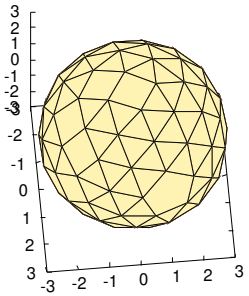

p=3
r_cluster
N
9
8
7
6
5
4
3
2
1
20 40 60 80 100 120 140 160 180 200


p=3
r_nearest
N
1.2
1.18
1.16
1.14
1.12
1.1
1.08
1.06
1.04
20 40 60 80 100 120 140 160 180 200

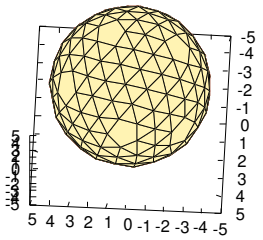
N=20



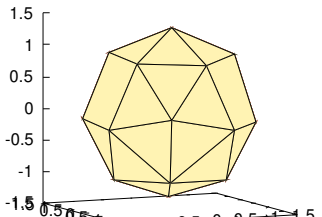
N=100



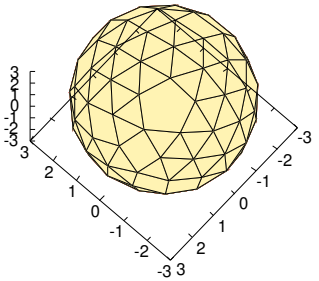
N=200



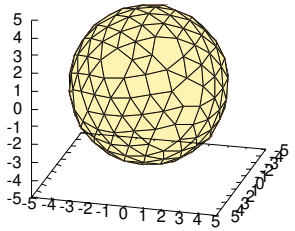
N=20



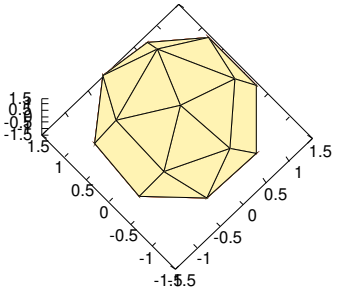
N=100



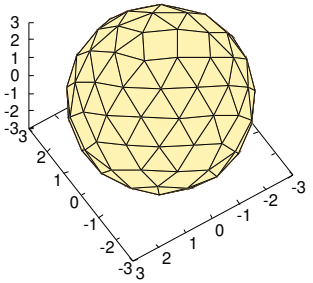
N=200



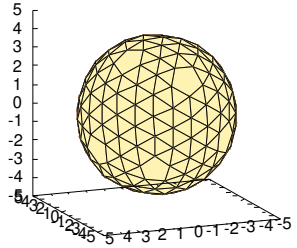
N=20



N=100



N=200



Przykład – zastosowanie metody SA w modelowaniu struktury przestrzennej fullerenów

- **fullereny** to puste w środku, otwarte lub zamknięte obiekty zbudowane wyłącznie z atomów węgla połączonych pojedynczymi i podwójnymi wiązaniami (atom C ma 4 elektrony walencyjne)
- inne, znane wcześniej formy węgla to: diament, grafit, sadza
- odkryte w 1985 (nagroda Nobla w 1996) i nazwane na cześć architekta Buckminstera Fullera, który projektował kopuły o podobnej strukturze geodezyjnej
- obiektami pokrewnymi do fullerenów to nanorurki węglowe oraz grafen – te obiekty też zbudowane są z pojedynczej warstwy atomów węgla, przy czym powierzchnia zbudowana jest z sześciokątów
- grafit możemy traktować jako nałożone na siebie warstwy grafenu oddziałujące ze sobą siłami van der Waalsa
- najpopularnieszym przykładem fullerenu jest C60 zbudowany z 60 atomów węgla, które zlokalizowane są w narożnikach pięcio- i sześciokątów – podobnie jak piłka futbolowa

- ze względu na wysokie przewodnictwo możliwe jest ich wykorzystanie w nanoelektronice, możliwe jest również umieszczanie w środku innych cząsteczek co znajduje zastosowanie w farmakologii

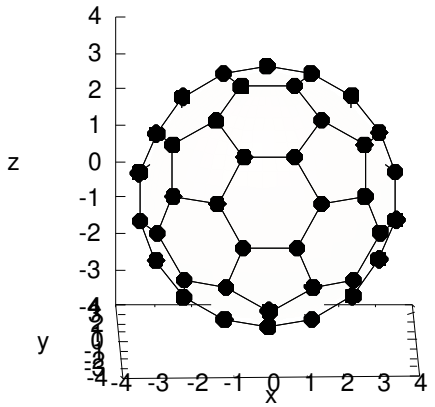
Fulleren C_{60}

- odległość do najbliższego sąsiada

$$r_{NN} = 1.42 - 1.45\text{Å}$$

- średnica

$$D \approx 7.04\text{Å}$$



Proces formowania fullerenów jest to proces częściowo stochastyczny, w którym silną rolę odgrywa kierunkowść wiązań atomów węgla. Fullereny C_n (n=20-100) mogą mieć różną konfigurację przestrzenne (izomery), a o ich stabilności decyduje energia całkowita (lub wiązania na jeden atom).

Ze względu na dużą liczbę atomów, nie jest możliwe bezpośrednio metod kwantowo-mechanicznych w celu wyznaczenia konfiguracji prztestrzennej atomów. Pierwszym krokiem jest zawsze użycie metody klasycznej, dopiero po wyznaczeniu położeń atomów używa się metod ab initio jak np. DFT (density functional theory).

Modelowanie fullerenów wykonamy przy użyciu **potencjału Brennera**. Parametryzacja potencjału pozwala opisać możliwość utworzenia przez pojedynczy atom maksymalnie czterech wiązań z najbliższymi sąsiadami.
Gdy w pobliżu atomu pojawi się 5 atom, wówczas potencjał oddziaływania rośnie (kara).

D.W. Brenner, Phys. Rev. B 42, (1990) 9458

Potencjał Brennera

- całkowita energia potencjalna układu N atomów

$$V_{tot} = \frac{1}{2}\sum_{i=1}^{N} V_i$$

każde wiązanie liczymy 2 razy, stąd ½ przed sumą

- potencjał oddziaływania z najbliższymi sąsiadami

$$V_i = \sum_{\substack{j=1 \\ j\neq i}}^{N} f_{cut}(r_{ij})\left[V_R(r_{ij}) - \overline{B}_{ij} V_A(r_{ij})\right]$$

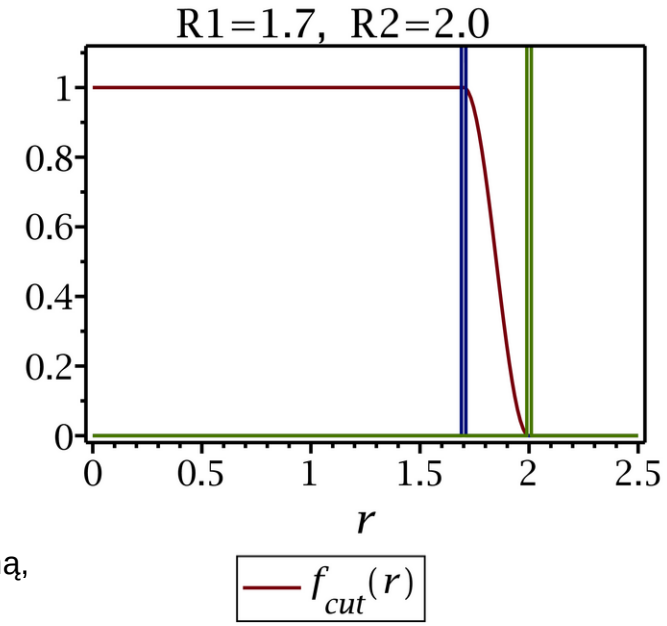
V_A – potencjał przyciągający (wiązanie kowalencyjne)
V_R – potencjał odpychający (uwzględnia np. zakaz Pauliego)
B_{ij} – czynnik uwzględniający aktualną liczbę najbliższych sąsiadów
f_{cut} – funkcja skalująca, sąsiedzi z wagą ~1, dalsze atomy z wagą ~0

parametry potencjału:

$$R_0 = 1.315\ \text{Å}$$
$$R_1 = 1.70\ \text{Å}$$
$$R_2 = 2.00\ \text{Å}$$
$$D_e = 6.325\ eV$$
$$S = 1.29$$
$$\lambda = 1.5\ \text{Å}^{-1}$$
$$\delta = 0.80469$$
$$a_0 = 0.011304$$
$$c_0 = 19$$
$$d_0 = 2.5$$

funkcja skalująca – jej zadaniem jest usuniecie wpływu dalszych atomów (wiązanie kowalencyjne jest krótkozasięgowe)

$$f_{cut}(r) = \begin{cases} 1, & r \le R_1 \\ \frac{1}{2}\left[1 + \cos\left(\frac{r-R_1}{R_2-R_1}\pi\right)\right], & R_1 < r \le R_2 \\ 0, & r > R_2 \end{cases}$$



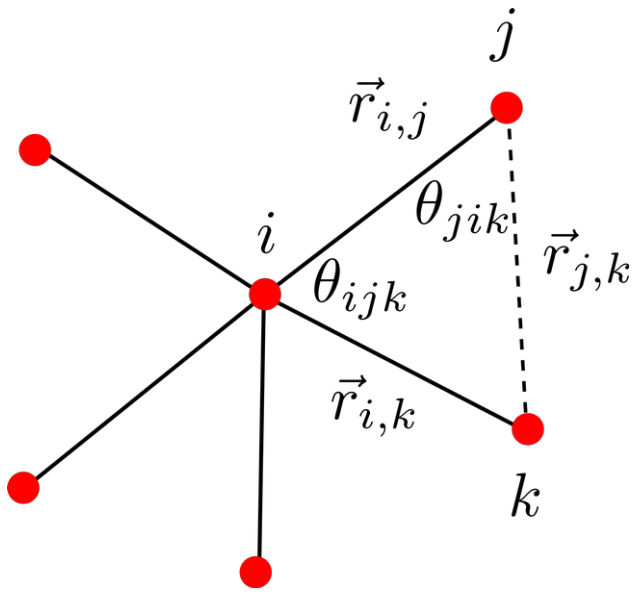
czynnik wielociałowy B_{ij} – określa otoczenia tj. najbliższych sąsiadów na energię potencjalną, jeśli jest ich więcej niż 4 to energia potencjalna rośnie, ma on **charakter oddziaływania trójciałowego**

$$\overline{B}_{ij} = \frac{B_{ij} + B_{ji}}{2}$$

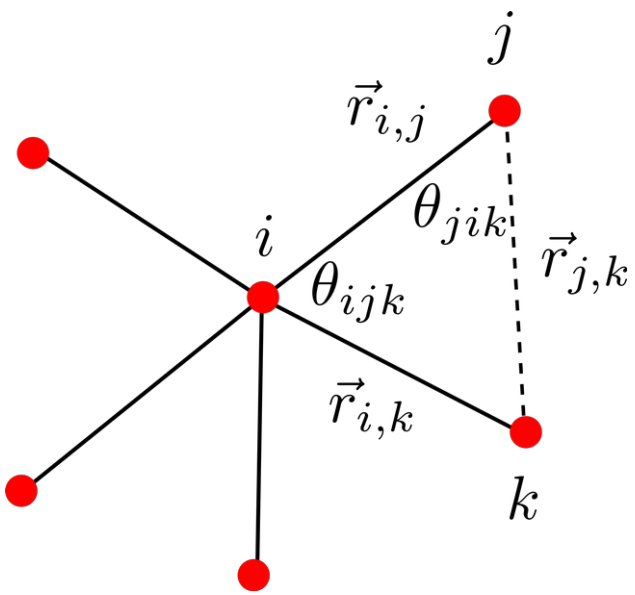
$$B_{ij} = (1 + \zeta_{ij})^{-\delta}$$

$$\zeta_{ij} = \sum_{\substack{k=1 \\ k \neq i,j}}^{n} f_{cut}(r_{ik}) g(\theta_{ijk}) \qquad \cos\theta_{ijk} = \frac{\vec{r}_{ij} \cdot \vec{r}_{ik}}{r_{ij}\, r_{ik}}$$

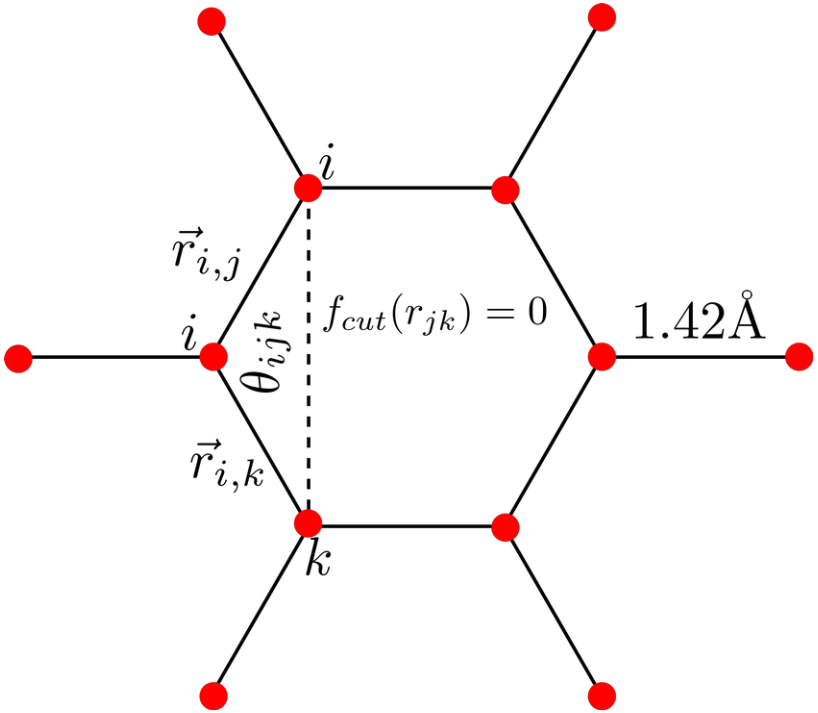
$$g(\theta_{ijk}) = a_0 \left[1 + \frac{c_0^2}{d_0^2} - \frac{c_0^2}{d_0^2 + (1 + \cos\theta_{ijk})^2}\right]$$



5 najbliższych sąsiadów – kara w postaci wzrostu energi potencjalnej



3 najbliższych sąsiadów – energia potencjalna najniższa



Ponieważ potencjał Brennera preferuje 4 najbliższych sąsiadów (4 elektrony walencyjne),
to algorytm SA w pierwszej kolejności wygeneruje konfiguracje zbudowane z czworokątów.
Takie konfiguracje nie są optymalne - mają wyższe energie niż te zbudowane z pięcio- i sześciokątów.

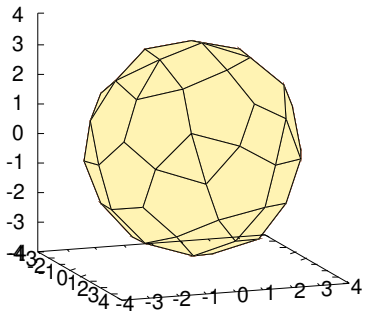
Aby ograniczyć mozliwość tworzenia wiązań 4-krotnych, wprowadzamy modyfikację w postaci wzrostu energii,
gdy wykryjemy kąt pomiędzy dwoma wiązaniami jest mniejszy niż π/2

$$if(\cos(\theta_{ijk}) > 0)\zeta_{ij} = \text{ large number}$$

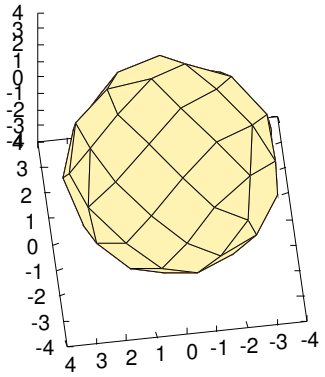
Optymalizacja Monte Carlo

- bez ingerencji w potencjał, konfiguracja znaleziona przez algorytm SA ma wysoką energię całkowitą, a atomy mają 3 lub 4 sąsiadów

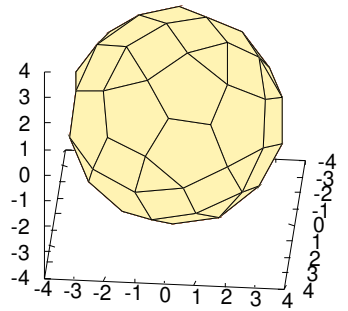
N=60, max 4-nearest neaighbours, E_{tot}= -358.2 eV



N=60, max 4-nearest neaighbours, E_{tot}= -358.2 eV

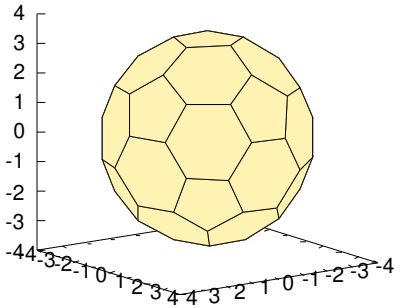


N=60, max 4-nearest neaighbours, E_{tot}= -358.2 eV

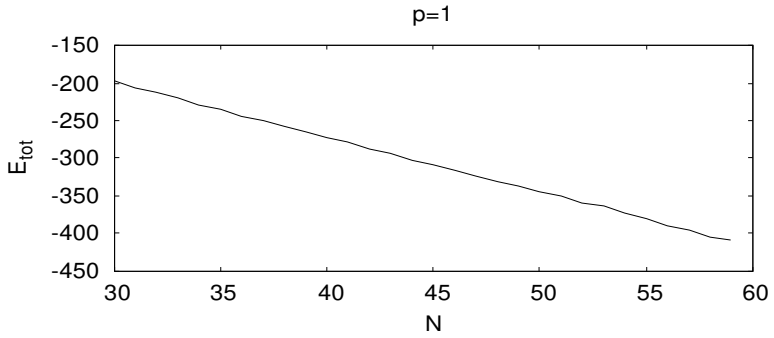


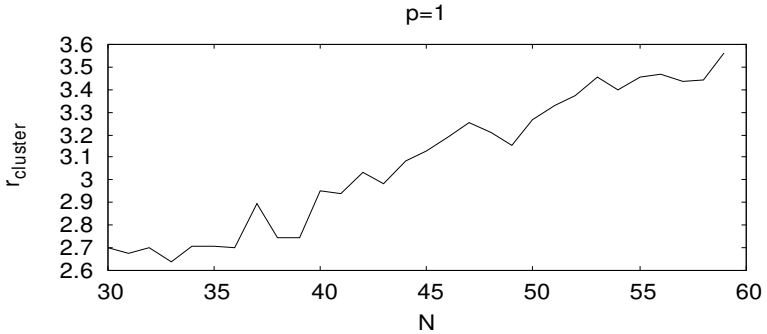
- optymalna konfiguracja C60, ma znacznie niższą energię całkowitą, atomy mają tylko 3 sąsiadów

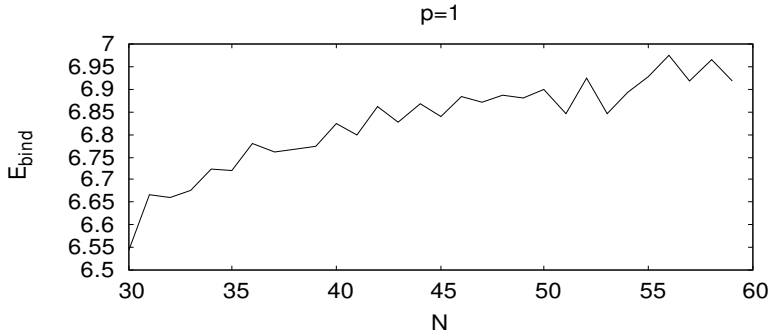
N=60, max 3-nearest neaighbours, E_{tot}= -421.2 eV

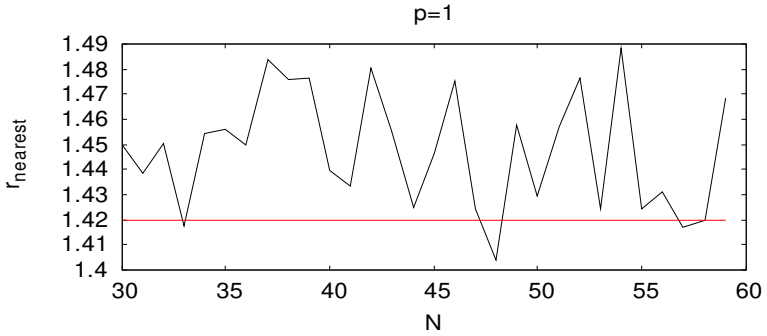


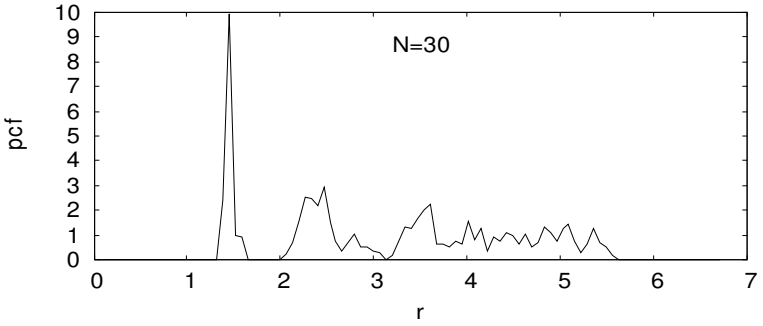
Przykładowe wyniki dla p=1

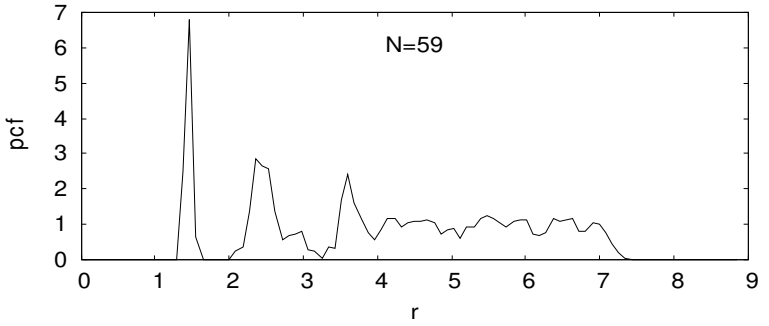




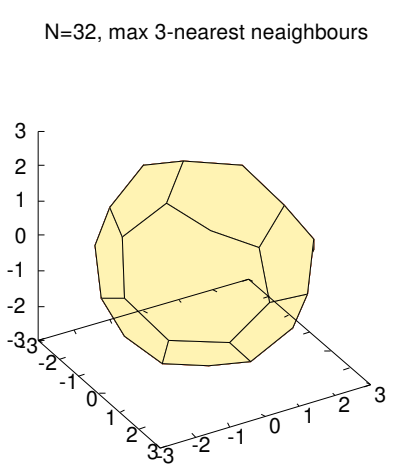


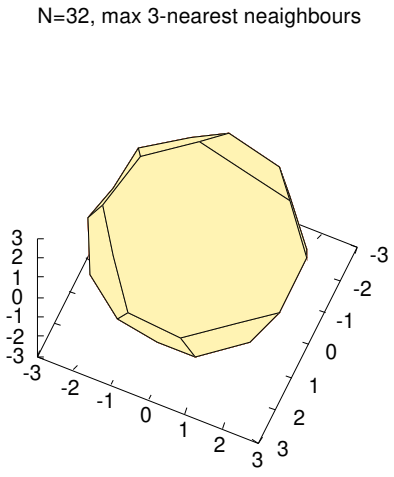






- w metodzie SA jeśli klaster straci połączenie z najbliższymi sąsiadami
  to może prowadzić to do otwarcia jego powierzchnii
- klaster otwarty może być nadal stabilny (ujemna energia całkowita), ale będzie mniej stabilny niż układ zamknięty
- przykład dla C32

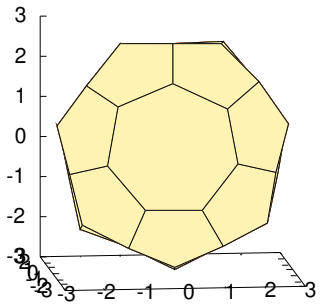




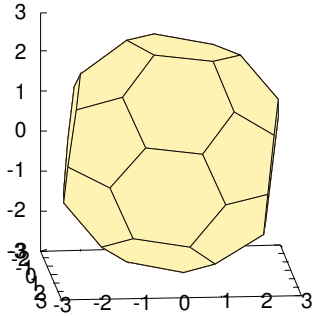
C42 jest bardzo stabilny i bardzo regularny, posiada:

- 14 pięciokątów
- 6 sześciokątów
- 2 siedmiokąty

N=42, max 3-nearest neaighbours

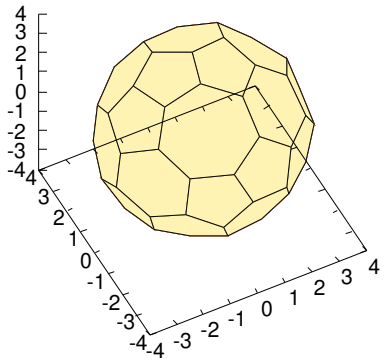


N=42, max 3-nearest neaighbours



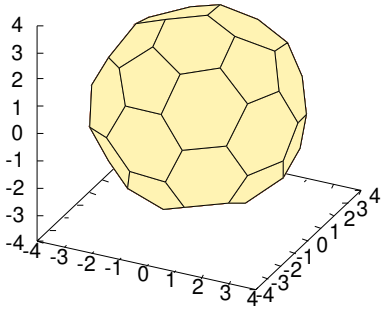
- najniższa energia uzyskana metodą SA dla C60 to -418.8 eV, około 2.4eV powyżej rzeczywistej najstabilniejszej konfiguracji
- znaleziona konfiguracja nie jest regularna, posiada 1 siedmiokąt, przez co położenia pozostałych atomów są zabużone (nieoptymalne)

N=60, max 3-nearest neaighbours, E_{tot}= -418.8 eV



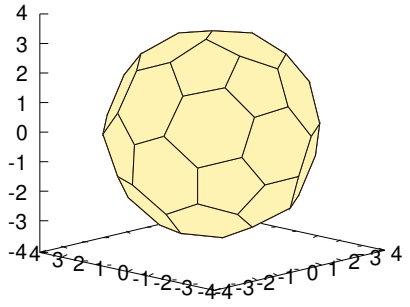
- siedmiokąt łamiący symetrię fullerenu

N=60, max 3-nearest neaighbours, E_{tot}= -418.8 eV



- sześciokąt otoczony dwoma pięciokątami i czterema sześciokątami

N=60, max 3-nearest neaighbours, E_{tot}= -418.8 eV



- sześciokąt otoczony niesymetrycznie trzema pięciokątami i trzema sześciokątami

Algorytmy genetyczne (GA – ang. genetic algorithm)

Omówiony algorytm symulowanego wyżarzania jest wynikiem naśladowania naturalnego procesu stopniowego obniżania temperatury obserwowanego np. w procesie metalurgicznym. Podstawowym założeniem algorytmu SA było stopniowe dostosowywanie się symulowanego układu do warunków narzuconych przez otoczenie (temperatura).

W świecie biologicznym też istnieją procesy, które powodują iż organizmy żywe dostosowują się do zmieniających się warunków zewnętrznych. Algorytmy komputerowe naśladujące te mechanizmy to algorytmy ewolucyjne, zaliczają się do nich: algorytmy genetyczne, algorytmy roju pszczół, algorytm kolonii mrówek i ich wariacje.

Spośród algorytmów ewolucyjnych największą elastycznością wykazują się **algorytmy genetyczne**, co oznacza że znajdują zastosowanie w największej liczbie problemów.

Zasada działania GA:

Zawsze rozważamy grupę osobników (**populację**), przy czym każdy z nich charaketryzuje się określonym indywidualnym zestawem **genów**. Ten zestaw genów determinuje sposób przystosowania osobnika do warunków otoczenia, który określony jest wartością **funkcji dopasowania** (poprzednio: **kosztu/celu**). W GA jedna iteracja oznacza tworzenie **nowego pokolenia**. Nowe osobniki powstają w wyniku **krzyżowania** najlepiej dostosowanych osobników (**selekcja naturalna**), po czym zastępują one osobniki najgorzej przystosowane. Tak samo jak w przyrodzie, geny wybranych osobników poddajemy **losowym mutacjom**, co zapobiega degeneracji (stagnacji) populacji. Oczekujemy (pewności nigdy nie ma), że po wygenerowaniu wielu tysięcy pokoleń, funkcja dopasowania (kosztu/celu) będzie miała najmniejszą wartość, a jego geny będą reprezentować wartości zmiennych dla których funkcja ta przyjmuje **minimum**.

- zdefiniujmy populację w GA, ma ona N osobników
- każdy osobnik ma swój indywidualny zestaw genów (chromosom)
- geny w tym przypadku interpretujemy jako zestaw argumentów funkcji dopasowania/celu/kosztu.

$$population = \begin{bmatrix} chromosome_1 \\ chromosome_2 \\ \vdots \\ chromosome_N \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} & \dots & g_{1M} \\ g_{21} & g_{22} & \dots & g_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ g_{N1} & g_{N2} & \dots & g_{NM} \end{bmatrix} = \begin{bmatrix} \vec{r}_{11} & \vec{v}_{12} & \dots & \alpha_{1M} \\ \vec{r}_{21} & \vec{v}_{22} & \dots & \alpha_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \vec{r}_{N1} & \vec{v}_{N2} & \dots & \alpha_{NM} \end{bmatrix}$$

- dla każdego osobnika obliczamy wartość funkcji dopasowania

$$f(chromosome) \rightarrow \vec{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix} = \begin{bmatrix} f(chromosome_1) \\ f(chromosome_2) \\ \vdots \\ f(chromosome_N) \end{bmatrix}$$

- musimy także określić sposób
  - realizacji selekcji naturalnej
  - doboru rodziców
  - krzyżowania → nowe osobniki
  - wprowadzania mutacji do populacji

Selekcja naturalna

W każdym pokoleniu/iteracji przeżywają tylko najlepiej dostosowane osobniki. Jak je wybrać? Mamy dwie możliwości

- sortujemy populację względem wartości funkcji dopasowania i wybieramy N_{best} osobników o najmniejszej wartości
- ustawiamy próg f_{max} który przekracza tylko $N_{treshold}$ osobników

$$\vec{f_i} = \begin{bmatrix} 18.75667 & (1) \\ -12.122 & (2) \\ 99.000 & (3) \\ 7899.987 & (4) \\ 0.0567 & (5) \\ -0.2354 & (6) \\ -1008.098 & (7) \\ 333.333 & (8) \end{bmatrix}$$

$$\vec{f}_{i,sort} \,\&\, N_{best} = 4 \quad \Longrightarrow \quad \left[\begin{array}{cc} -1008.098 & (7) \\ -12.122 & (2) \\ -0.2354 & (6) \\ 0.0567 & (5) \\ \hline 18.75667 & (1) \\ 99.000 & (3) \\ 333.333 & (8) \\ 7899.987 & (4) \end{array}\right]$$

$$\vec{f_i} \,\&\, threshold = 100 \quad \Longrightarrow \quad \left[\begin{array}{cc} -1008.098 & (7) \\ -12.122 & (2) \\ -0.2354 & (6) \\ 0.0567 & (5) \\ 18.75667 & (1) \\ 99.000 & (3) \\ \hline 333.333 & (8) \\ 7899.987 & (4) \end{array}\right]$$

- **usunięte osobniki zastępujemy nowymi**

Dobór partnerów

Po wykonaniu selekcji naturalnej określamy liczbę nowych osobników które musimy wygenerować, aby uzupełnić populację oraz wybrać i połączyć w pary osobniki najlepiej dostosowane – partnerzy.
Partnerów możemy dobierać z całej populacji, także tych usuwanych, lub tylko z części osobników które przeżywają.

- **ruletka** – każdemu z osobników przypisujemy określone prawdopodobieństwo wylosowania (+ warunek unormowania)

$$best \Longrightarrow \{1, 2, 3, \ldots, N_{best}\}$$

$$probabilities \Longrightarrow \{p_1, p_2, p_3, \ldots, p_{N_{best}}\}$$

$$\sum_{i=1}^{N_{best}} p_i = 1$$

przykład 1: podział równomierny

$$p_i = \frac{1}{N_{best}}$$

- wszyscy traktowani tak samo

przykład 2 - podział nierównomierny:

$$p_i = \frac{C_{N_{best}}}{d+i}$$

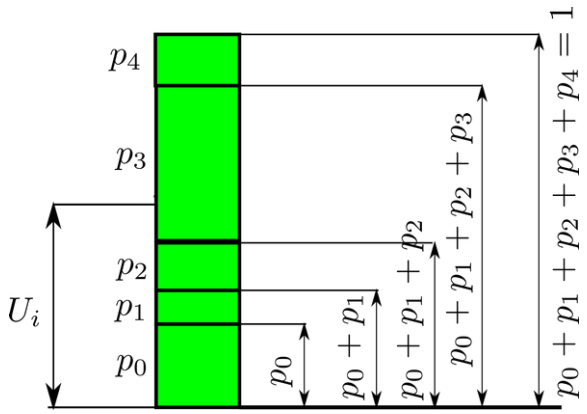
- najlepsze osobniki są uprzywilejowane (elita)

$$\sum_{i=1}^{N_{best}} p_i = C_{N_{best}} \sum_{i=1}^{N_{best}} \frac{1}{d+i} = 1 \quad \longrightarrow \quad C_{N_{best}} = \frac{1}{\sum_{i=1}^{N_{best}} \frac{1}{d+i}}$$

Jeśli mamy wylosować K osobników (K może być większe niż N_{best}) to stosujemy algorytm generowania liczb dla rozkładu dyskretnego (wykład → generatory)

```
inicjalizacja: K, mate[K]
for(i=1; i <= K; i++){
    U_i ~ U(0,1)
    for(l=1; l <= N_best; l++){
        if( sum_{i=1}^{l-1} <= U_i < sum_{i=1}^{l} ) new_mate=l
    }
    mate[i]=l
}
```

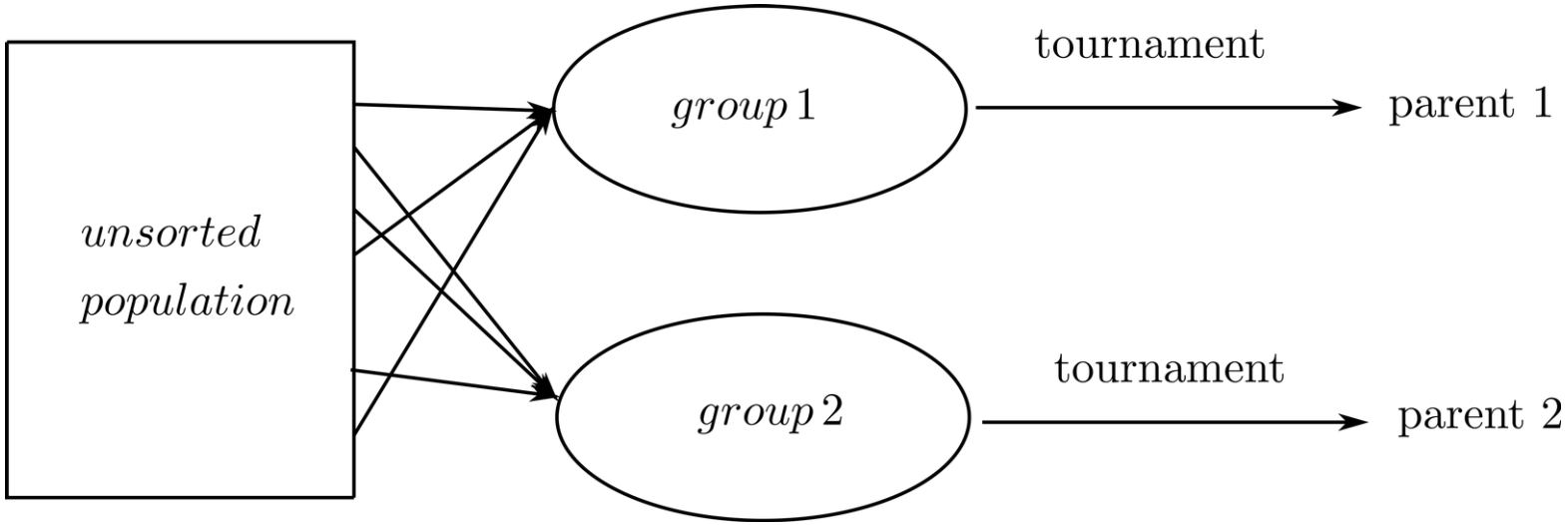
rysunek z innego wykładu
więc indeksowanie od 0



Uwagi:

- jeśli stosujemy selekcję naturalną opartą o pozostawienie N_{best} osobników w populacji to prawdopodobieństwa możemy wyznaczyć tylko raz, na początku, a później korzystać z nich w każdej iteracji
- dla selekcji progowej liczba osobników które przezywają może się zmieniać, wówczas należy prawdopodobieństwa obliczać w każdej iteracji

- **turniejowy dobór partnerów –** losowo tworzymy 2 niewielkie grupy osobników, z każdej grupy wybieramy zwycięzcę czyli osobnika z najlepszą funkcją dopasowania, procedurę powtarzamy do momentu, gdy uzyskamy odpowiednią liczbę partnerów



Krzyżowanie

Mając do dyspozycji odpowiednio liczną populację partnerów przystępujemy do generowania nowego pokolenia.
Geny nowego osobnika powinny pochodzić od obu rodziców.
W związku z tym pojawiają się kwestie

- jak określić liczbę przekazywanych genów? – pakiety genów nie muszą być równoliczne
- które geny przekazać?
- a może poszczególne geny lepiej zmieszać? wówczas: z jakimi wagami?

- **krzyżowanie jednorodne** – jeśli geny reprezentowane są przez liczby,
  wówczas w sposób jednorodny możemy mieszać **wszystkie** geny obojga rodziców

$$A = [a_1\, a_2\, a_3\, \ldots\, a_M]$$
$$B = [b_1\, b_2\, b_3\, \ldots\, b_M]$$

- zestaw genów rodziców

$$C = \beta\, A + (1-\beta) B$$
$$D = (1-\beta)\, A + \beta B$$

- zestaw genów potomstwa

$$\beta \in (0,1)$$

- parametr mieszania

- **krzyżowanie jednopunktowe** – określamy numer genu, który podlega mieszaniu, natomiast geny przed i za tą pozycją pochodzą tylko od jednego z rodziców

$$k = int\left(U_1 \cdot M + 1\right), \qquad U_1 \sim U(0,1)$$

$$A = [a_1\, a_2\, a_3\, \ldots\, a_k\, \ldots\, a_M]$$
$$B = [b_1\, b_2\, b_3\, \ldots\, b_k\, \ldots\, b_M]$$

- zestaw genów rodziców

$\beta \in (0,1)$

$$c_k = a_k - \beta(a_k - b_k)$$
$$d_k = b_k + \beta(a_k - b_k)$$

- mieszamy tylko jeden k-ty gen

$$C = [\underbrace{a_1\, a_2\, a_3\, \ldots\, a_{k-1}}_{A}\ \underbrace{c_k}_{mix}\ \underbrace{b_{k+1}\, b_{k+2}\, \ldots\, b_M}_{B}]$$

- zestaw genów potomstwa

$$D = [\underbrace{b_1\, b_2\, b_3\, \ldots\, b_{k-1}}_{B}\ \underbrace{d_k}_{mix}\ \underbrace{a_{k+1}\, a_{k+2}\, \ldots\, a_M}_{A}]$$

uwaga:

- jeśli β=0 lub β=1 to wówczas wymieniane są pakiety genów (bez mieszania), podczas jednego krzyżowania pakiety przekazywanych genów mogą zawierać różną liczbę genów, ale uśredniając te liczby po wielu pokoleniach okaże się, że rodzice przekazują zbliżoną liczbę genów

- **mutacje** – mutacje wprowadzamy po to aby wprowadzić różnorodność do populacji i uniknąć stagnacji rozwiązania w lokalnym minimum

Uwaga:
- mutacji nie podlega najlepszy osobnik, jeśli znajduje się on w minimum funkcji dopasowania/celu, to mutacja mogłaby go przemieścić i otrzymalibyśmy gorszy wynik
- mutujemy tylko osobniki o indeksach i=2,3,...,N

- mutacje wprowadzamy z określonym prawdopodobieństwem: p_{mut}
- jest to zazwyczaj mała liczba p_{mut}~0.01-0.05, duże prawdopodobieństwo może spowodować zbyt duże/szybkie zmiany w populacji
- trzeba jeszcze określić w jaki sposób wykonać mutację, jeśli jest to liczba to zmieniamy jej wartość

algorytm mutacji genów zakodowanych jako liczby

define: p_{mut}, Δ
find best $f_{i_{best}} = \min\{f_1, f_2, \ldots, f_N\} \longrightarrow i_{best}$

```
for(i=1; i <= N; i++){
    if(i != i_best){
      for(j=1; j <= M; j++){
          U_1 ~ U(0,1)
          if(U_1 <= p_mut){
              U_2 ~ U(0,1)
              a_j = a_j + (2U_2 - 1)Δ
          }
      }
    }
}
```

Wiedząc jak działają poszczególne elementy GA możemy naszkicować ogólną postać algorytmu

```
for(it=1; it <= IT_max; it++){
    find: f_{i_best} ⟶ i_best
    do:
        natural selection
        mate selection
        crossover
        upgrade population
        mutation
    check STOP criterion
}
```

Uwagi:

- ze względu na ogólną postać, algorytm jest elastyczny
- mamy swobodę wyboru dokonania: (i) selekcji naturalnej, (ii) doboru partnerów, (iii) krzyżowania, (iv) mutacji
- warunek STOP może uwzględniać, oprócz maksymalnej liczby iteracji, także:
  - przekroczenie minimalnej wartości funkcji dopasowania
  - braku zmian funkcji dopasowania najlepszego osobnika w kilku kolejnych iteracjach (stagnacja rozwiązania)
  - przekroczenie czasu symulacji
- geny mogą reprezentować dowolne wielkości, nie tylko liczbowe (ilościowe), ale też jakościowe
- na wydajność algorytmu silnie wpływa też liczność populacji, zazwyczaj im większa tym lepiej,
  ale może to spowalnić generowanie kolejnych generacji (np. w przypadku sortowania kilku tysięcy osobników)

Przykład → poszukiwanie najkrótszej trasy w mieście przy użyciu GA

populacja osobników

$\vec{r}_1$	$\vec{r}_{1,2}$	$\vec{r}_{1,3}$	$\dots$	$\vec{r}_{1,n-1}$	$\vec{r}_n$
$\vec{r}_1$	$\vec{r}_{2,2}$	$\vec{r}_{2,3}$	$\dots$	$\vec{r}_{2,n-1}$	$\vec{r}_n$
$\vec{r}_1$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vec{r}_n$
$\vec{r}_1$	$\vec{r}_{N,2}$	$\vec{r}_{N,3}$	$\dots$	$\vec{r}_{N,n-1}$	$\vec{r}_n$

krzyżowanie - ruletka

$R_1 =$ | * | * | * | * | * | * | * | * | * |

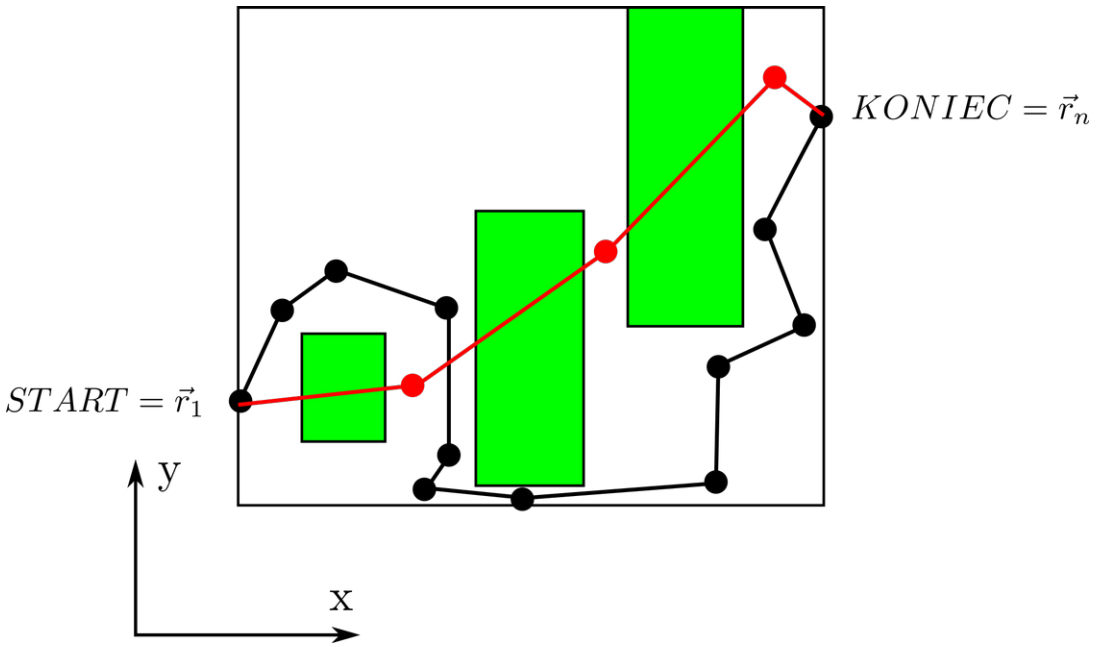
$R_2 =$ | * | * | * | * | * | * | * | * | * |

$\uparrow$

$k \sim U(2, n-1)$

$D_1 =$ | * | * | * | * | * | * | * | * | * |

$D_2 =$ | * | * | * | * | * | * | * | * | * |



mutacja

$$\vec{r}_{i,j}^{new} = \vec{r}_{i,j} + \Delta[U_x, U_y], \qquad U_x, U_y \sim U(-1, 1)$$

selekcja naturalna – kasujemy połowę populacji

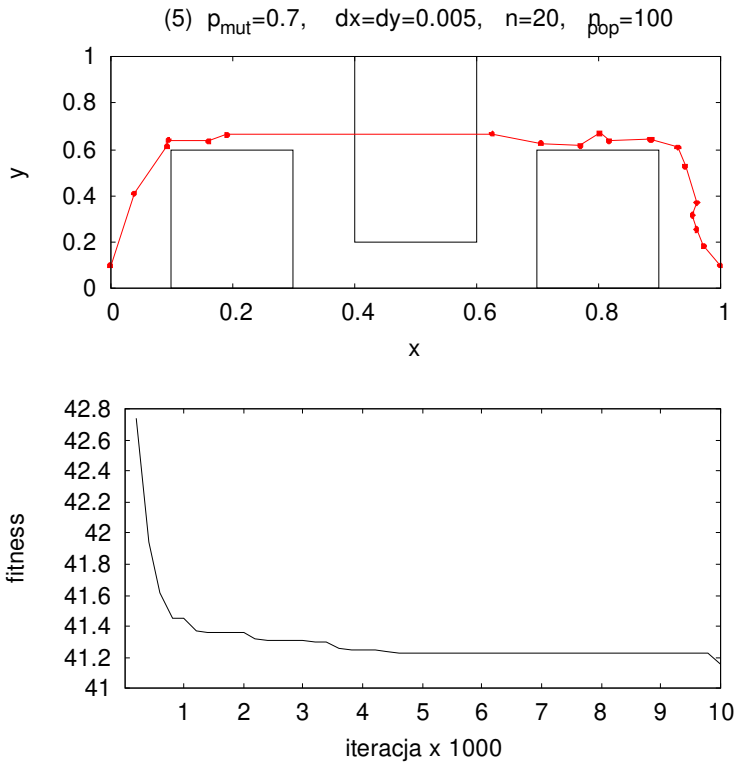
Dane zbiorcze serii symulacji

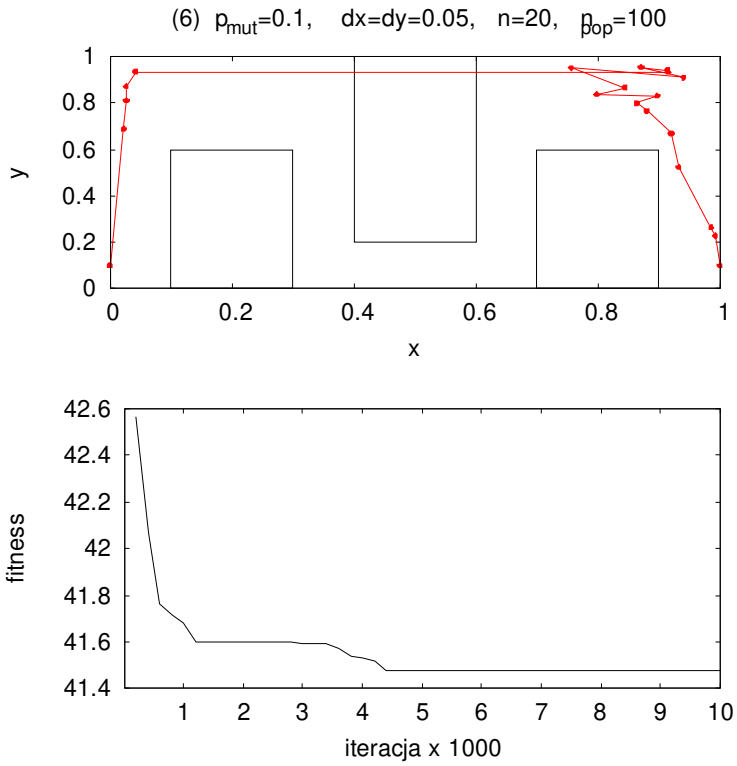
	p_mut	dx	dy	n	n_pop	IT_MAX
1	0.7	0.005	0.005	20	1000	10000
2	0.2	0.005	0.005	20	1000	10000
3	0.1	0.005	0.005	20	1000	10000
4	0.7	0.005	0.005	30	1000	10000
5	0.7	0.005	0.005	20	100	30000
6	0.1	0.05	0.05	20	100	30000
7	0.1	0.05	0.05	30	200	30000

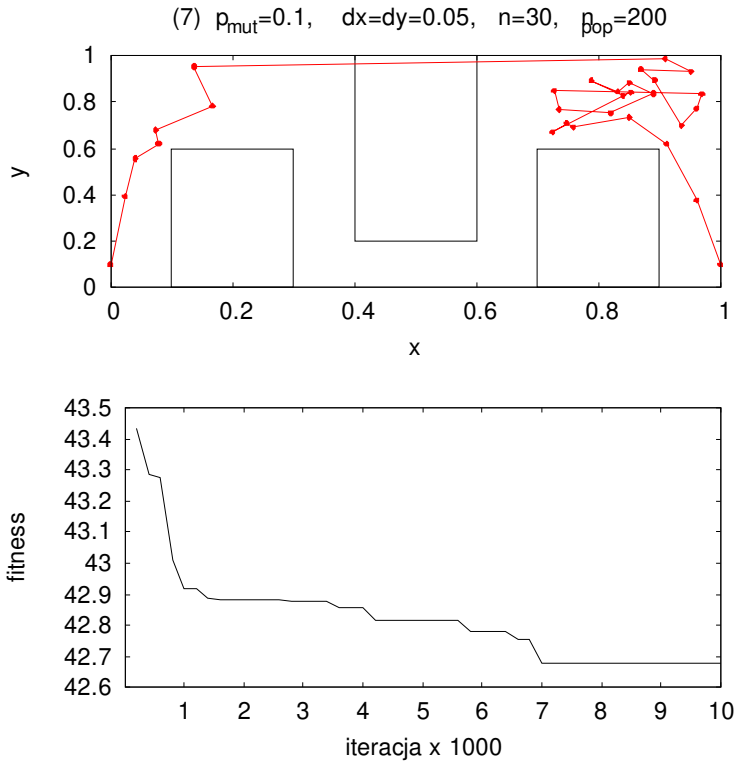
dane: 1,2,3,4 - uzyskano zbieżność

dane: 5,6,7 – brak zbieżności

- na początek zobaczmy błędne wyniki – co może pójść nie tak?



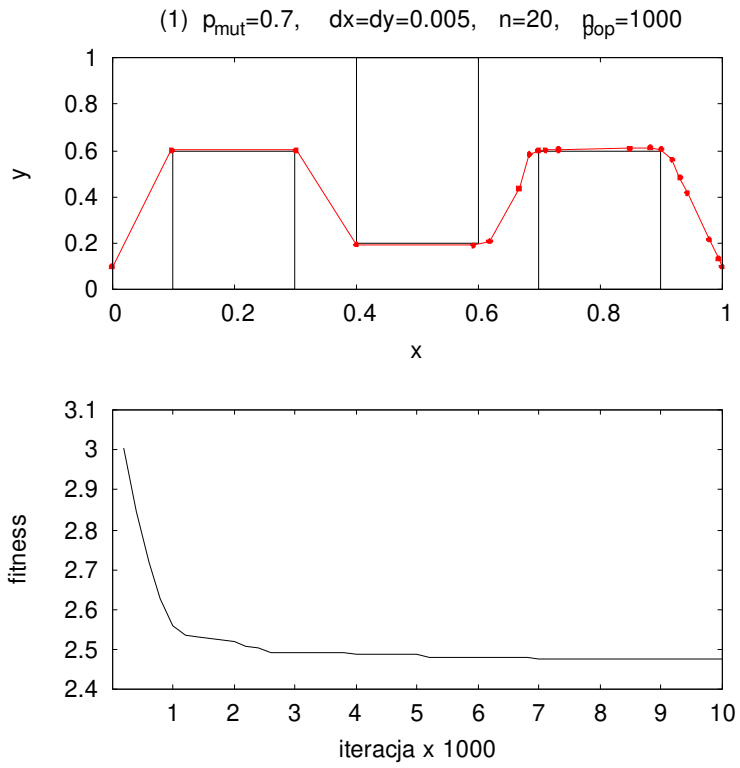


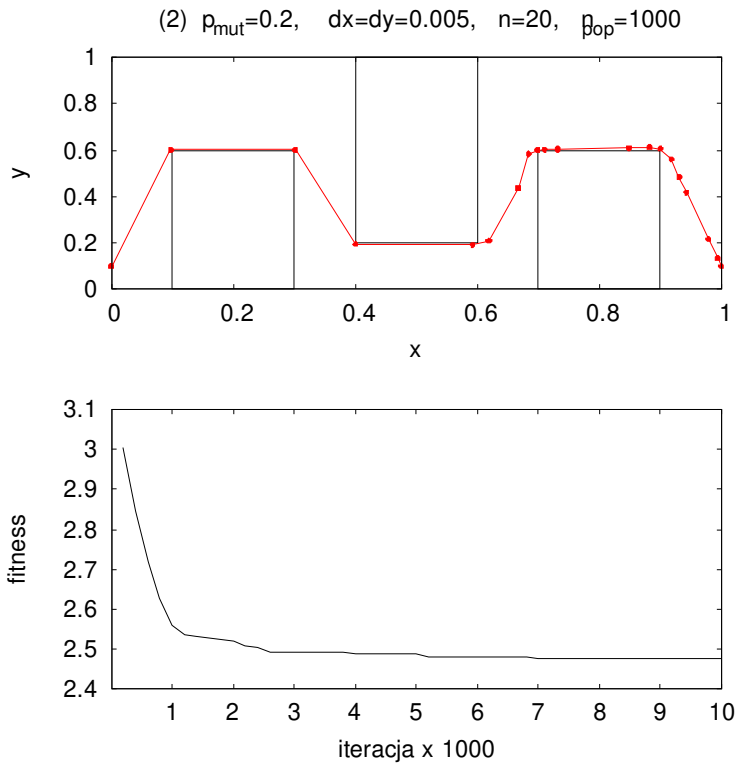


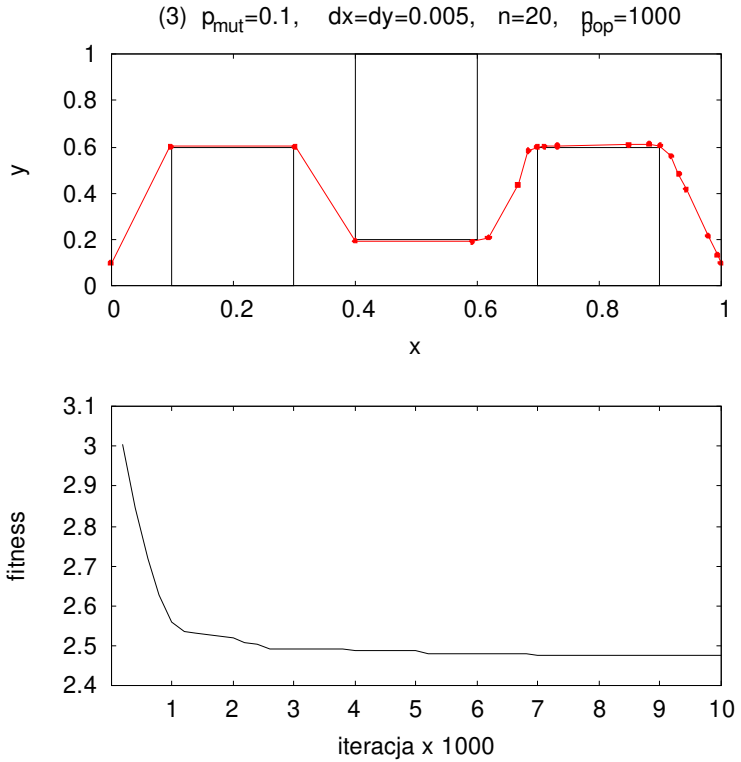
Uwagi:

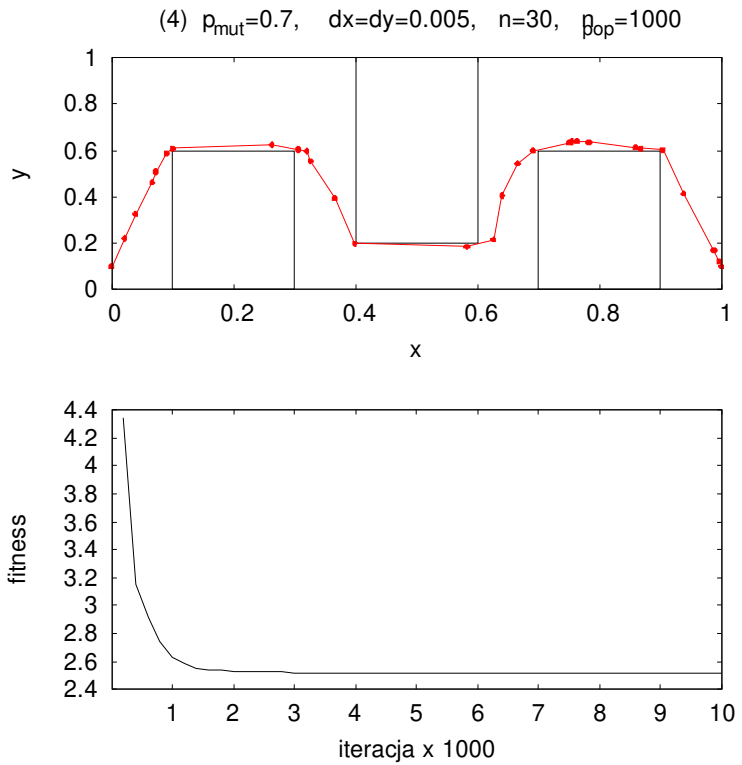
- brak zbieżności, trasy przechodzą przez budynki
- wyniki podobne dla małej i dużej wartości p_{mut}
- wyniki podobne dla n=20 i n=30
- wyniki podobne dla n_{pop}=100 i n_{pop}=200
- co trzeba zmienić aby ominąć budynki??

- **zwiększamy populację do 1000 osobników**









uwagi:

- parametrami kluczowymi są w tym przypadku:
  - liczność populacji n_{pop}
  - maksymalne przesunięcie Δ (nie może być zbyt duże)

- zauważmy że p_{mut} może zmieniać się w dużym zakresie 0.1-0.7
- najlepszy wynik dla n=20 (fitness<2.5), ale dla n=30 fintness ~2.5
- duża populacja=duża zbieżność, wynik bliski rzeczywistemu minimum dostajemy dla 1000-3000 iteracji