Tematy

1. Problemy NP i wprowadzenie do algorytmów heurystycznych
2. Postawowe strategie projektowania algorytmów heurystycznych
   - Hill-climbing (algorytm wspinaczki ?)
   - Symulowane wyżarzanie
   - Przeszukiwanie tabu
   - Algorytmy genetyczne
3. Podsumowanie

Problemy NP

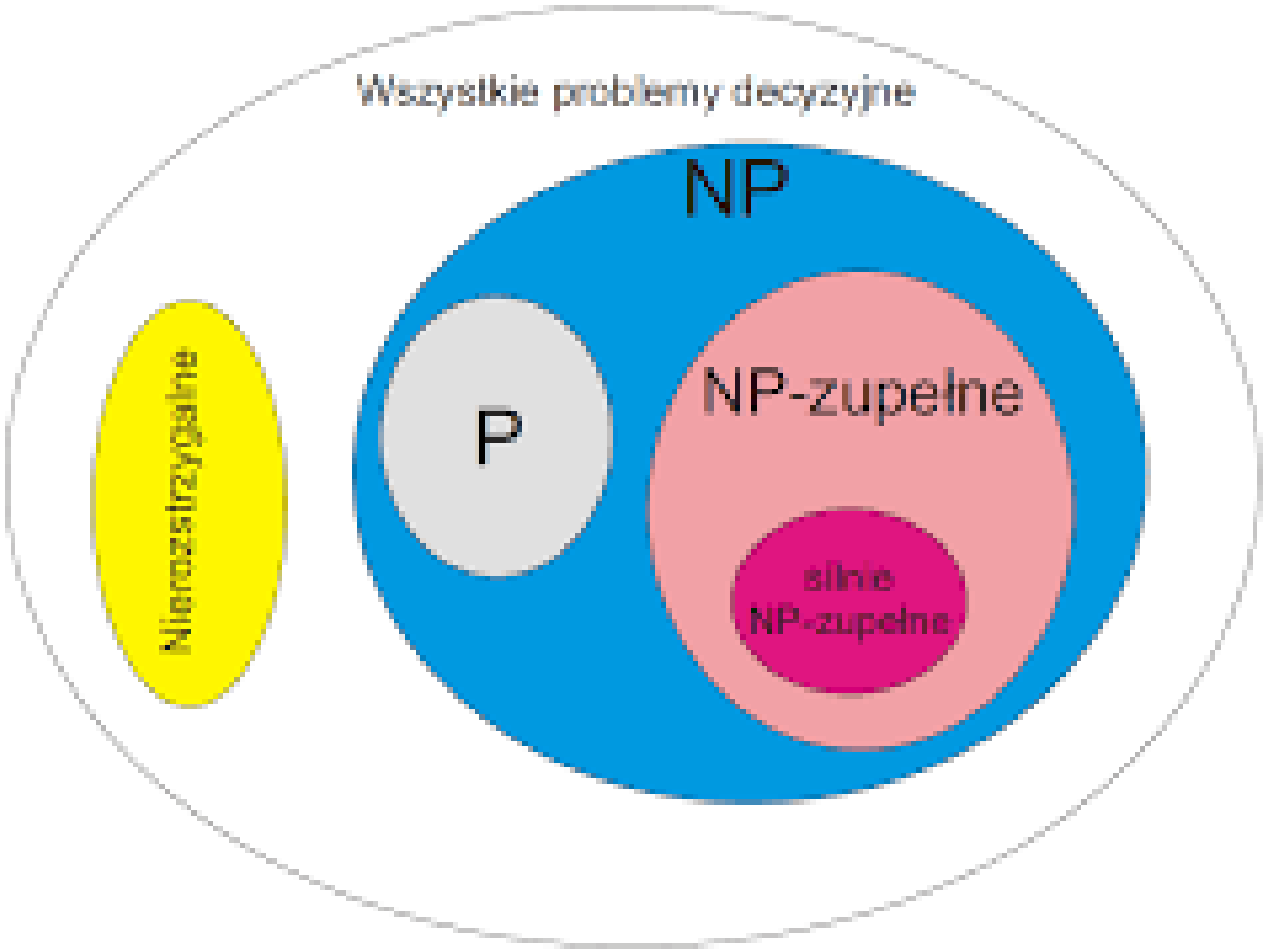
Złożoność czasowa algorytmu to ilośc operacji podstawowych koniecznych do wykonania algorytmu w zależności od rozmiaru wejścia

Klasa P Istnieje wielomianowy algorytm dla problemu na deterministycznej maszynie Turinga.

Klasa NP Istnieje wielomianowy algorytm dla problemu na niedeterministycznej maszynie Turinga.

Podaje się też że dla problemów P znalezienie rozwiązania ma być możliwe w czasie wielomianowym, a dla problemów NP sprawdzenie podanego z góry rozwiązania ma być wykonywalne w takim czasie.

Problemy NP


Wszystkie problemy decyzyjne
NP
P
NP-zupełne
silnie NP-zupełne
Nierozstrzygalne

Algorytmy heurystyczne

Algorytm heurystyczny – algorytm niedający (w ogólnym przypadku) gwarancji znalezienia rozwiązania optymalnego, umożliwiający jednak znalezienie rozwiązania dość dobrego w rozsądnym czasie.
Algorytmy tego typu używane są w takich problemach obliczeniowych, gdzie znalezienie rozwiązania optymalnego ma zbyt dużą złożoność obliczeniową (w szczególności są to problemy NP-trudne) lub w ogóle nie jest możliwe.

Problemy optymalizacyjne

Ogólna forma problemu optymalizacyjnego:
Dane:
Skończony zbiór X
Funkcja celu P: $X \rightarrow \mathbb{Z}$
Funkcje ograniczające g_j: $X \rightarrow \mathbb{Z}, 1 \leq$ j $\leq$ m
Szukane:
Maksymalna wartość $P(x)$, gdzie x $\in X$ oraz $g_j(x) \geq 0$ dla $1 \leq$ j $\leq$ m

Funkcja sąsiedztwa

W celu projektowania algorytmów heurystycznych potrzebna nam będzie **Funkcja sąsiedztwa – N**: $X \rightarrow 2^X$
Wynik dziełania tej funkcji na pewnym elemencie x z X będziemy nazywali sąsiedztwem, ma ono reprezentować elementy dziedziny podobne (bliskie) elementowi x.
Typowo nasza funkcja sąsiedztwa będzie wyglądała nastepująco:
$N_{d_0}(x) = y \in X : d(x, y) \leq d_0$
gdzie d_0 jest zadaną wartością graniczną a d metryką zadana na X.

Przeszukiwanie sąsiedztwa

Przeszukiwanie sąsiedztwa – algorytm który, przy danej funkcji sąsiedztwa N, przyjmuje dopuszczalne rozwiązanie x $\in X$ i zwraca dopuszczalne rozwiązanie y $\in N(x)\backslash\{x\}$, lub informacje o błędzie.
Wyróżniamy 4 podstawowe strategie przeszukiwania sąsiedztwa:
1. Znajdź i zwróć dopuszczlne rozwiązanie y $\in N(x)\backslash\{x\}$ maksymalizujące $P(x)$, zwróc 'błąd' jeśli nie ma dopuszczalnego rozwiązania w $N(x)\backslash\{x\}$
2. Znajdź dopuszczlne rozwiązanie y $\in N(x)\backslash\{x\}$ maksymalizujące $P(x)$, jesli takie rozwiązanie y istnieje i $P(y)/ > P(x)$ zwróć y, w przeciwnym wypadku zwróc 'błąd'.
3. Znajdź jakiekolwiek dopuszczlne rozwiązanie y $\in N(x)\backslash\{x\}$.
4. Znajdź jakiekolwiek dopuszczlne rozwiązanie y $\in N(x)\backslash\{x\}$, jeśli $P(y)/ > P(x)$ zwróć y, w przeciwnym wypadku zwróc 'błąd'.

Przeszukiwanie sąsiedztwa

Ogólna forma algorytmu heurystycznego dla problemu optymalizacji opartego na przeszukiwaniu sąsiedztwa:
Dane:
X,N,P,g_j: $X \rightarrow$ Z, 1 $\leq$ j $\leq$ m, c_{max},h_n(funkcja przeszukująca sąsiedztwo (jedna z wcześniejszych 4))
Algorytm:

Znajdź dowolne dopuszczalne rozwiązanie x $\in$X

$x_{best} = x$

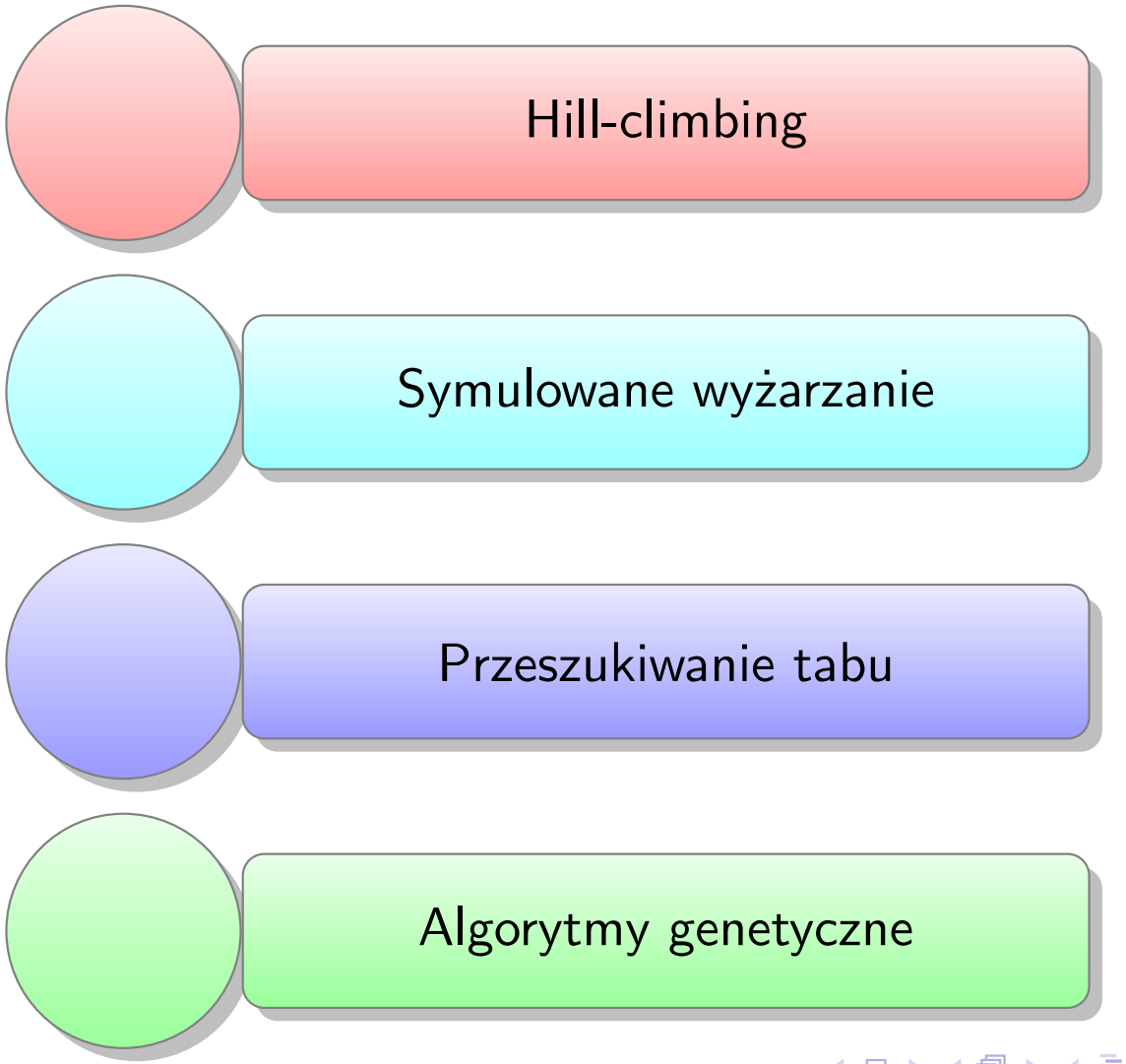
c=0

Przeszukiwanie sąsiedztwa

Algorytm c.d.:

while c$\leq c_{max}$
  y=h$_n(x)$
  if y $\neq'$ bd'
    x=y
    if P(x) > P(x$_{best}$)
      x$_{best} = x$
  c=c+1
return x$_{best}$

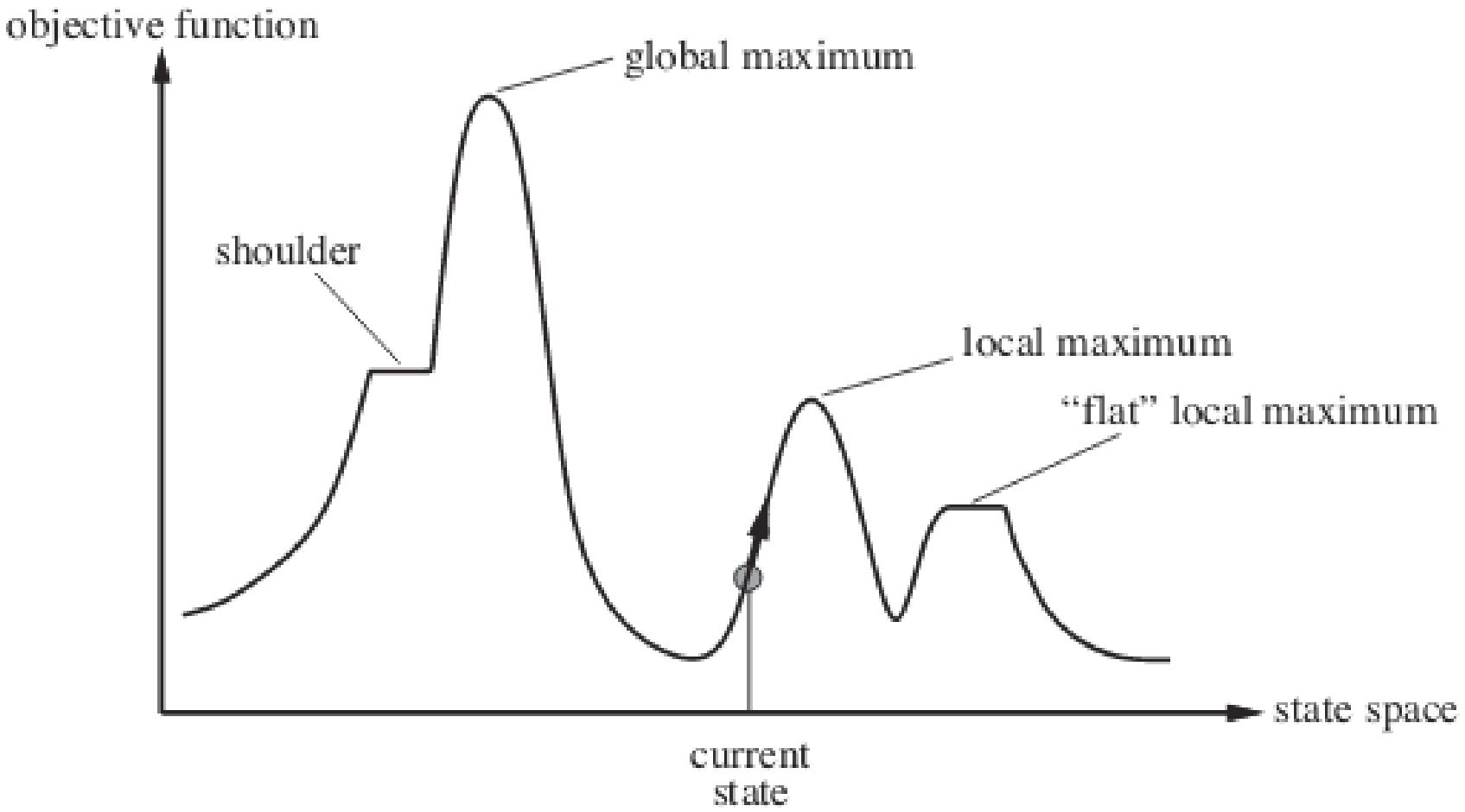
Przykłady algorytmów heurystycznych



Hill-climbing

Najprostszym podejściem do projektowania algorytmów heurystycznych jest **hill-climbing**, kiedy stosujemy to podejście chcemy aby P(y)>P(x) dla któregoś y z sąsiedztwa x aby kontynuować algorytm.
Takie podejście jest oparte na heurystycznym założeniu, że aby najszybciej dojśc na najwyższy szczyt trzeba cały czas iść do góry (nie zawsze to musi być prawdą).
Najczęstszym problemem przy tym podejściu jest możliwośc utknięcia w lokalnym maksimum.

Hill-climbing



Ogólna postać

Ogólna forma algorytmu hill-climbing:
Dane:
X,N,P, h_n
Uwaga wybieramy h_n tak by w przypadku zwrócenia dopuszczalnego rozwiązania P(y)>P(x)
Algorytm:

Znajdź dowolne dopuszczalne rozwiązanie x $\in$X
$x_{best} = x$
przeszukiwanie=1

Ogólna postać

Algorytm c.d.:

while przeszukiwanie == TAK
  y=$h_n(x)$
  if y $\neq'$ bd'
    x=y
    if P(x) > P(x_{best})
      $x_{best} = x$
  else
    przeszukiwanie = NIE
return x_{best}

Przykład - jednorodny podział grafu

W tym problemie chcemy podzielić wierzchołki grafu ważonego o parzystej liczbie wierzchołków na dwa równoliczne podzbiory tak aby suma wag krawędzi między tymi zbiorami była jak najmniejsza.
Dane:
Graf pełny G =(V,E) zawierające 2n wierzchołków. Funkcja kosztu c: E$\rightarrow \mathbb{Z}$
Szukane:
X_1, X_2 minimalizujące wartość $C(X_1, X_2) = \sum_{u \in X_1, v \in X_2} c(u, v)$
gdzie V=$X_1 X_2 oraz |X_1| = |X_2| = n$
Dla rozwiązania $x = [X_1, X_2]$ jego sąsiedztwem N(x) będą wszystkie równoliczne podziały powstałe poprzez zamienienie co najwyżej jednego elementu z $X_1 elementem z X_2$.

Przykład - jednorodny podział grafu

Dane:

G,N,C,c_{max},h_n

Algorytm:

Znajdź dowolne dopuszczalne rozwiązanie $[X_1, X_2]$

c=1

(globalnie) błąd=0

while c $\leq c_{max}$

$[Y_1, Y_2] = h_n([X_1, X_2])$

if błąd $\neq 1$

$X_1 = Y_1$

$X_2 = Y_2$

c=c+1

return $[X_1, X_2]$

Przykład - jednorodny podział grafu

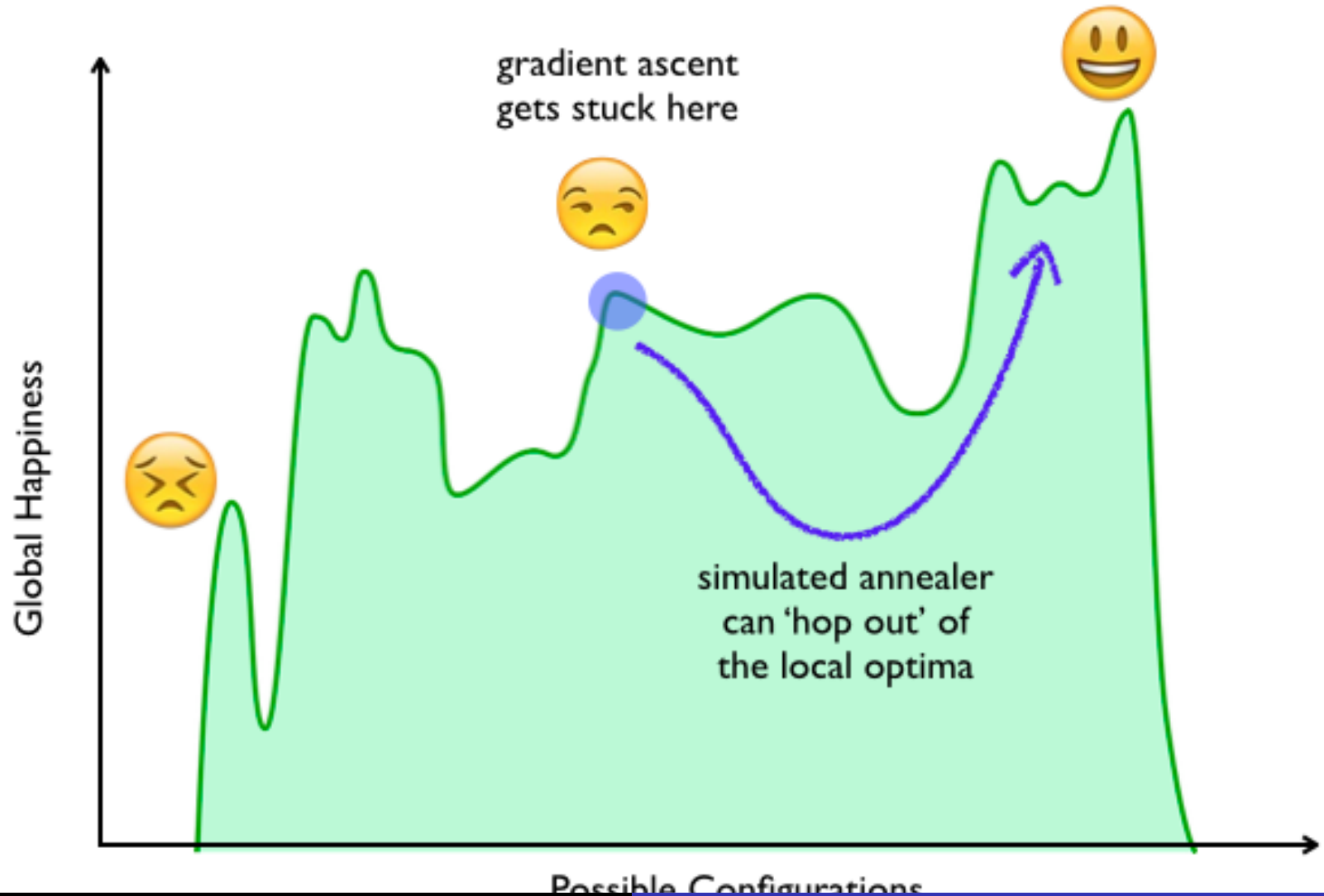
Funkcja h_n :
Dane:
$X_1, X_2, G, N, C, bd(globalnazmienna)$
Algorytm:

$bd = 1 \quad foreach\ i \in X_1$
$foreach\ j \in X_2$
$[Y_1, Y_2] = [X_1 + j - i, X_2 + i - j]$
$if\ \ C(Y_1, Y_2) < C(X_1, X_2)$
$[X_1, X_2] = [Y_1, Y_2]$
$bd = 0$
$return\ [X_1, X_2]$

Symulowane wyżarzanie

Jedną z metod ucieczki z lokalnego maksimum jest tzw. **symulowane wyżarzanie** które ma być analogią do chłodzenia rozgrzanego metalu.
W tym podejściu używamy zrandomizowanej strategii do sąsiedztw. Jeśli y=$h_n(x)$ jest dopuszczalne i P(y)>P(x) to postępujemy jak w hill-climbing, jesli jednak jest inaczej to możemy czasem (z pewnym prawdopodobieństwem) przejść do y i tak.
Zmienną związaną z ta metodą jest T (odpowiednik temperatury), inicjalizowane jako $T_0 > 0$. Wraz z biegiem algorytmu zmienna T maleje, zwykle z każdą iteracją poprzez przemnożenie przez stałą a z przedziału (0,1).
Prawdopodobieństwo przejścia z x do y dla przypadku P(y)<P(x) jest zadane poprzez wzór $e^{(P(y)-P(x))/T}$

Symulowane wyżarzanie



Ogólna postać

Ogólna forma algorytmu symulowanego wyżarzania:
Dane:
c_{max}, T_0, a,X, N, P,h_n
Algorytm:
c=0
T=T_0
Znajdź dowolne dopuszczalne rozwiązanie x $\in$X
$x_{best} = x$

Ogólna postać

Algorytm c.d.:

while c $\leq$ c_{max}
  y=$h_n(x)$
  if y $\neq'$ bd'
    if P(y) > P(x)
      x=y
      if P(x) > P(x_{best})
        $x_{best} = x$
    else
      r=random(0,1)
      if r<$e^{(P(y)-P(x))/T}$
        x=y
  c=c+1
  T=a*T
return x_{best}

Przykład - problem plecakowy

W tym problemie chcemy ustalić które z n możliwych przedmiotów chcemy spakowac do plecaka, tak aby zmaksymalizować sumę ich wartości, nie przekraczając równocześnie maksymalnej wagi.

Dane:

$p_0, ..., p_{n-1}$-wartości przedmiotów.

$w_0, ..., w_{n-1}$-wagi przedmiotów.

M-maksymalna waga

Szukane:

x=$[x_0, ..., x_{n-1}]$, gdzie $x_i = 1$ oznacza spakowanie przedmiotu i, $x_i = 0$ oznacza jego niespakowanie, maksymalizujące $P(x) = \sum_{i \in [0,n-1]} x_i * p_i$, gdzie w(x)=$\sum_{i \in [0,n-1]} x_i * w_i \leq M$.

Przykład - problem plecakowy

Dane:

G,N,C,c_{max},h_n

Algorytm:

Znajdź dowolne dopuszczalne rozwiązanie $[X_1, X_2]$

c=1

(globalnie) błąd=0

while c $\leq c_{max}$

$[Y_1, Y_2] = h_n([X_1, X_2])$

if błąd $\neq 1$

$X_1 = Y_1$

$X_2 = Y_2$

c=c+1

return $[X_1, X_2]$

Przykład - problem plecakowy

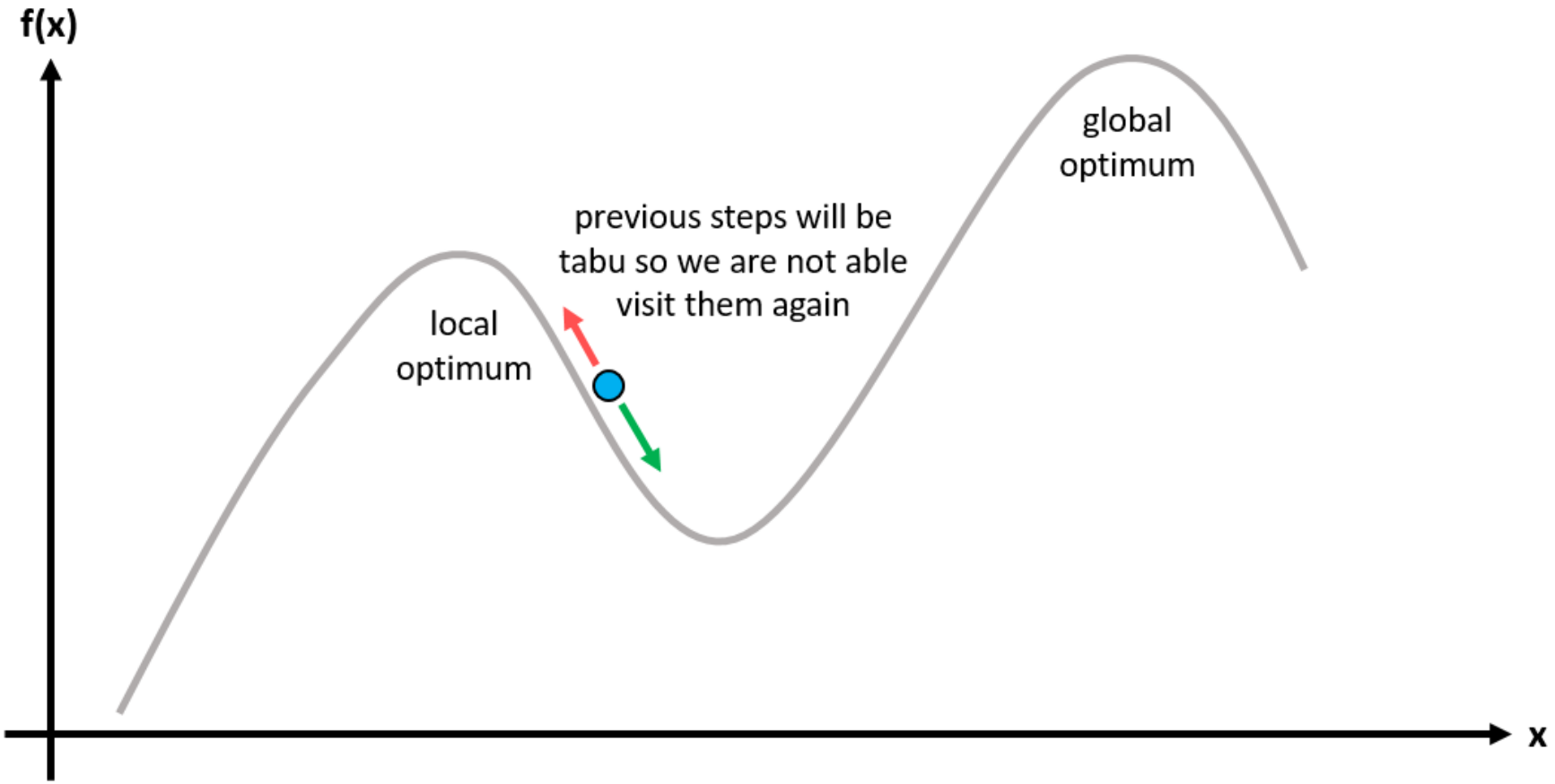
Funkcja $\mathbf{h}_n$:
Dane:
$\mathbf{X}_1, X_2, G, N, C, bd(globalnazmienna)$
Algorytm:

błąd=1
for each i $\in X_1$
for each j $\in X_2$
$[\mathbf{Y}_1, Y_2] = [X_1 + j - i, X_2 + i - j]$
if $\mathbf{C}(\mathbf{Y}_1, Y_2) < C(X_1, X_2)$
$[\mathbf{X}_1, X_2] = [Y_1, Y_2]$
błąd=0
return $[\mathbf{X}_1, X_2]$

Przeszukiwanie tabu

Przeszukiwanie tabu zakłada zastępowanie wyjściowego rozwiązania x przez dopuszczalne y należące do jego sąsiedztwa, maksymalizujące funkcję P w tym sąsiedztwie. W tym algorytmie nie wymagamy aby P(y)>P(x), co daje możliwość ucieknięcia z lokalnego maksimum. Aby zabazpieczyć się przed szybkim powrotem wykorzystujemy 'listę tabu' przetrzymującą zmiany jakich musielibyśmy dokonać aby z y wrócic do x. Teraz nie będziemy mogli wprowadzić tych zmian przez ustalona liczbę iteracji. "Liste tabu' implementujemy następująco (przy założeniu że jesteśmy w iteracji nr. c) TabuList[c]=change(y,x)
Heurystyczne przeszukiwanie będzie miało teraz następującą postać: $h_n(x) = y, gdziey \in$ N(x), y jest dopuszczalne, change(x,y) $\notin$ Tabulist[d], dla c-L$\leq$d$\leq$c-1, oraz y jest maksimum w sąsiedztwie x.

Przeszukiwanie tabu



Ogólna postać

Ogólna forma algorytmu przeszukiwania tabu:
Dane:
c_{max}, L,X, N, P,h_n
Algorytm:
c=0
Znajdź dowolne dopuszczalne rozwiązanie x $\in$X
$x_{best} = x$

Ogólna postać

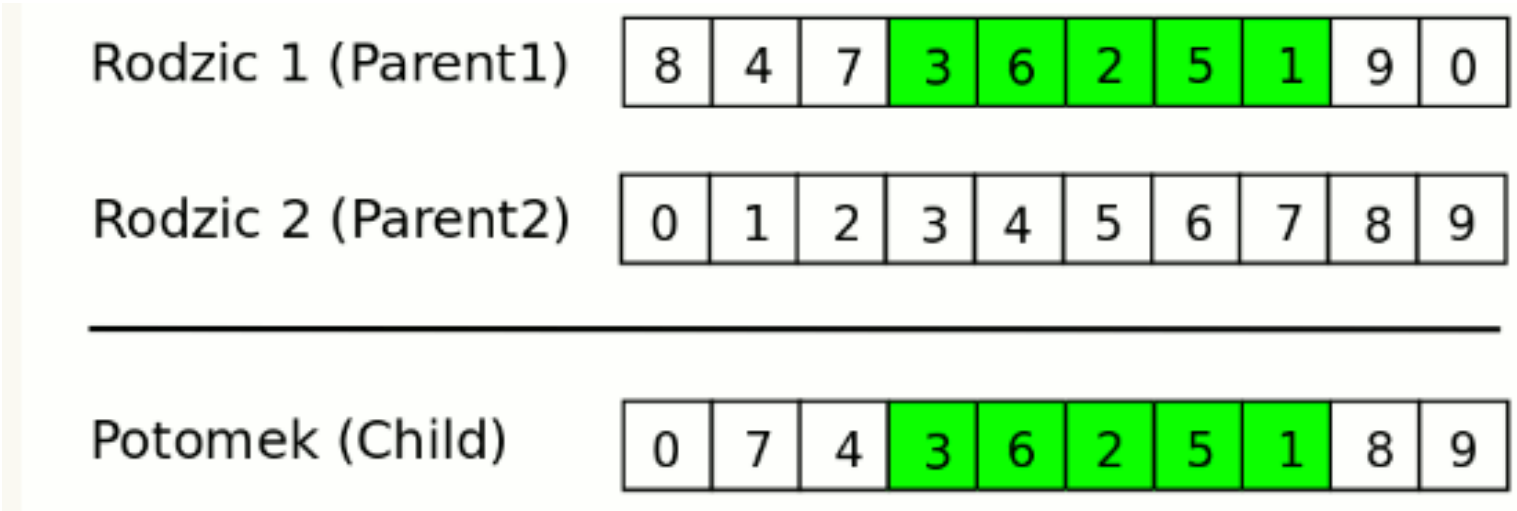
Algorytm c.d.:

while c $\leq$ c_{max}
N=N(x)\{ $Tabulist[d] : c - L \leq d \leq c\text{-}1$ }
for each y $\in$ N
if y jest niedopuszczalne
N=N\{y}
if N == $\varnothing$
zakończ
znajdź y $\in$ N, takie że P(y) maksymalne
TabuList[c]=change(y,x)
x=y
if P(x)>P(x_{best})
$x_{best} = x$
c=c+1
return x_{best}

Algorytmy genetyczne

W **Algorytmach genetycznych** inaczej niż w poprzednich podejściach rozpoczynamy pracę mając pewną ilośc dopuszczalnych rozwiązań. Rozwiązania te są grupowane (najczęściej w pary) w celu wyprodukowania rozwiązań pochodnych (nastepnej generacji rozwiązań). Algorytm musi precyzować jak dokładnie rozwiązania pochodne są produkowane, typowym podejściem jest wzięcie 2 rozwiązań i zastosowanie operacji rekombinacji tych rozwiązań aby wyprodukować dwa nowe rozwiązania.

Algorytmy genetyczne



Ogólna postać

Ogólna forma algorytmów genetycznych:
Dane:
c_{max}, L,X, N, P,h_n,
rec - funkcja rekombinująca dwa rozwiązania,
pp- rozmiar populacji
Algorytm:
c=1
Wybierz początkowy zbiór rozwiązań dopuszczalnych A, o liczności pp.
x_{best}=element A maksymalizujący funkcje celu P
foreach x $\in A$
x=$h_n(x)$

Ogólna postać

Algorytm c.d.:

while c $\leq$ c_{max}
Skonstruuj parowanie elementów z A (ozn pA)
Q=A
foreach [w,q] $\in pA$
(y,z)=rec(w,q)
y=$h_n(y)$
z=$h_n(z)$
Q=Q$\cup\{y, z\}$
A = pp najlepszych rozwiązań z Q
y=element z A maksymalizujący P
if P(y)>P(x_{best})
x_{best}=y
c=c+1
return x_{best}

Przykład - problem komiwojażera

W tym problemie chcemy wyznaczyć kolejnośc odwiedzanych wierzchołków (permutacje wszystkich wierzchołków G), tak aby suma krawędzi w uzyskanym cyklu hamiltona (długośc trasy) była jak najmniejsza. **Dane**:
Graf pełny G =(V,E) Funkcja kosztu c: E$\rightarrow \mathbb{Z}$
Szukane:
Permutacja x zbioru wierzchołków V
postaci$[x_1, ..., x_n]$, *minimalizujca* $\sum_{i\in[1,n-1]} c(x_i, x_{i+1}) + c(x_{n-1}, x_1)$

Podsumowanie

- Hill-climbing
- Symulowane wyżarzanie
- Przeszukiwanie tabu
- Algorytmy genetyczne