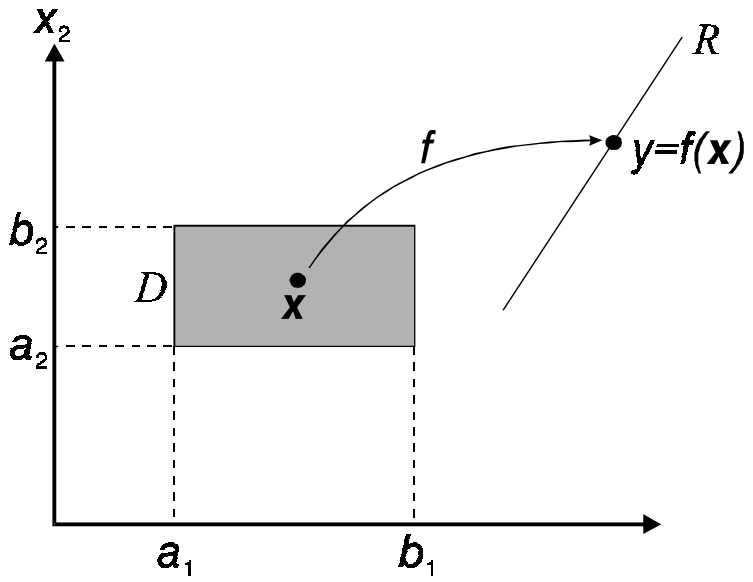
2. prednáška

Optimalizačný problém a kódovanie

$$f : D \rightarrow R$$

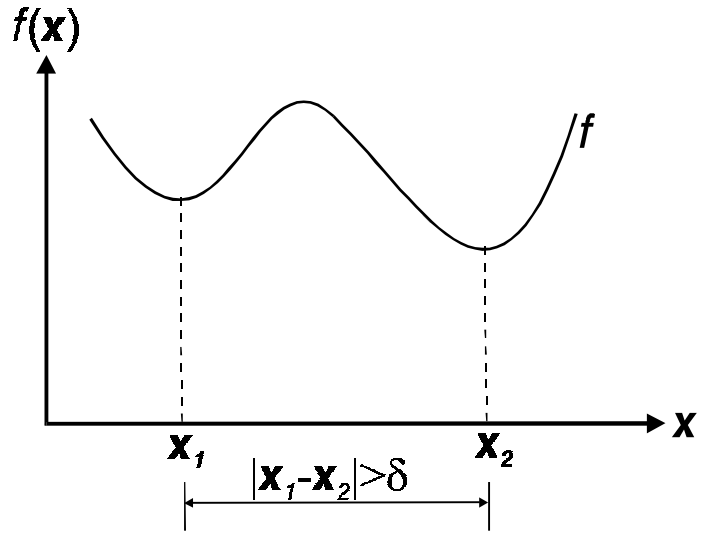
$$D = \prod_{i=1}^{n} [a_i, b_i] = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$$



Funkcia *f* je ohraničená podmienkami

(1) Existuje taký algoritmus, ktorý funkciu *f* "vypočíta dostatočne rýchlo" s požadovanou presnosťou pre každé $x \in D$ (hovoríme, že funkcia f je *dobre vypočitateľná*).

(2) Pre každú dvojicu lokálnych miním x_1, $x_2 \in D$ vzdialenosť $|x_1 - x_2|$ je väčšia ako dané kladné číslo $\delta > 0$, $|x_1 - x_2| > \delta$. Podmienka automaticky vylučuje z triedy prípustných funkcií tie funkcie, ktoré sú "fraktálového" typu, t.j. v každom okolí nejakého minima sa nachádza aspoň jedno iné minimum.



Globálne minimum funkcie *f* na kocke *D* je určené vzťahom

$$x_{opt} = \arg\min_{\boldsymbol{x} \in D} f(\boldsymbol{x})$$

Nájdenie globálneho minima použitím klasických optimalizačných metód patrí vo všeobecnosti medzi obtiažne numerické problémy pre funkcie, ktoré nie sú ohraničené ďalšími podmienkami

Z týchto dôvodov sa v súčasnosti [4,5] pri riešení globálneho optimaliyačného problému často používajú tzv. *evolučné optimalizačné algoritmy*, ktoré poskytujú riešenia blízke globálnemu, alebo s ním totožné.

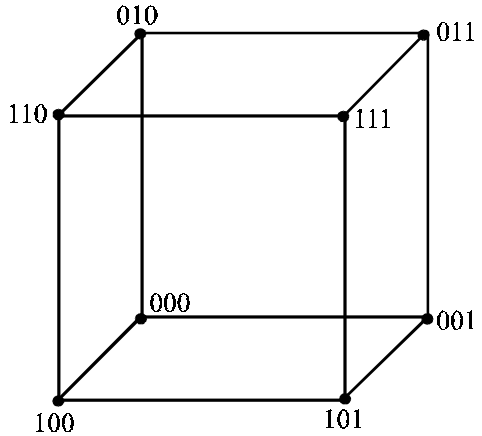
Binárna verzia funkcie má tvar

$$f:\{0,1\}^k \to R$$

$$y = f(\alpha)$$

Táto funkcia je definovaná nad množinou binárnych vektorov dĺžky *k*, kardinalita množiny binárnych vektorov dĺžky *k* je

$$\left|\{0,1\}^k\right| = 2^k$$

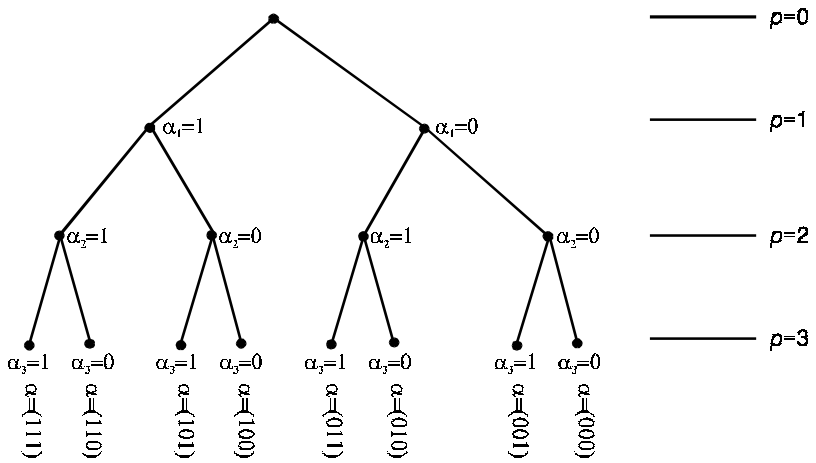


Optimalizačný problém pre binárne vektory má tvar

$$\boldsymbol{\alpha}_{opt} = \arg \min_{\boldsymbol{\alpha} \in \{0,1\}^k} f(\boldsymbol{\alpha})$$

Vo všeobecnosti, globálne optimum α_{opt} sa nájde po preskúšaní všetkých možných binárnych vektorov dĺžky *k*. Algoritmicky tento prístup môže byť implementovaný pomocou metódy spätného prehľadávania. CPU čas potrebný na riešenie optimalizačnej úlohy je potom úmerný kardinalite priestoru riešení

$$t_{CPU} \propto 2^k$$



Binárna reprezentácia reálnej premennej

Binárny vektor α dĺžky k

$$\alpha = (\alpha_1 \alpha_2 \ldots \alpha_k) \in \{0,1\}^k$$

je interpretovaný ako celé číslo

$$\text{int}(\alpha) = \sum_{i=1}^{k} \alpha_i 2^{k-i} =$$

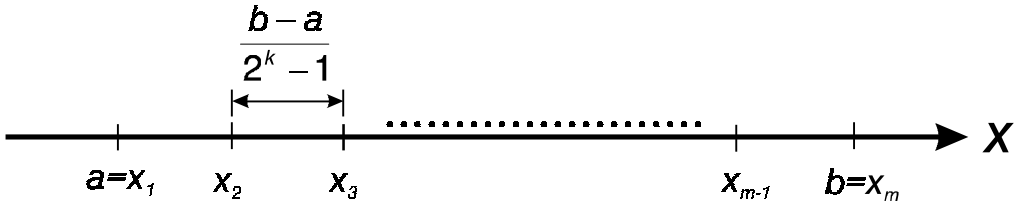
$$= \alpha_1 2^{k-1} + \alpha_2 2^{k-2} + \ldots + \alpha_{k-1} 2 + \alpha_k$$

K tomuto celému číslu priradíme racionálne číslo $x \in [a,b]$

$$x \approx \text{real}(\alpha) = a + \frac{b-a}{2^k - 1} \text{int}(\alpha)$$

real(α) aproximuje požadované reálne číslo x s presnosťou $(b-a)/2^k-1$.

Interval $[a,b]$ obsahuje $m=2^k$ bodov $x_1=a$, $x_2=a+(b-a)/(2^k-1)$, ..., $x_i=a+(i-1)(b-a)/(2^k-1)$, ..., $x_n=b$, pozri obr. 2.5 a tab. 2.1.



No.	α	int(α)	real(α)	$\tilde{\alpha}$ (Gray)
1	000	0	0	000
2	001	1	1/7	001
3	010	2	2/7	011
4	011	3	3/7	010
5	100	4	4/7	110
6	101	5	5/7	111
7	110	6	6/7	101
8	111	7	1	100

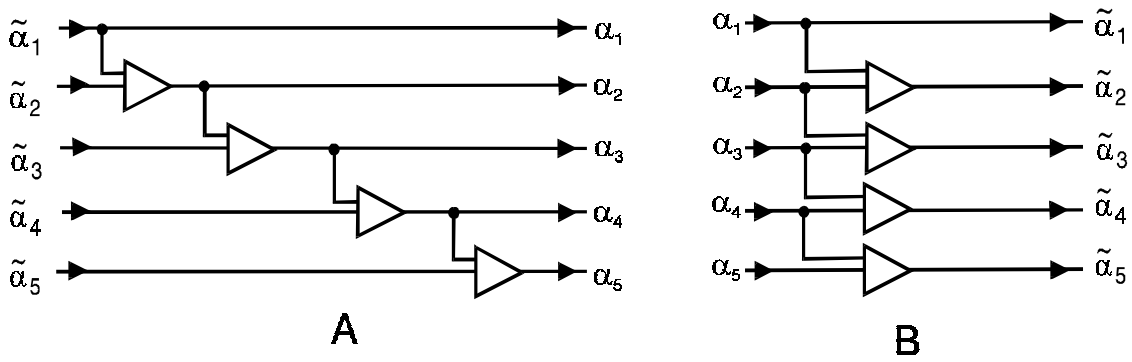
Grayova binárna reprezentácia

Dvojica binárnych reťazcov, ktoré sú odlišné vo všetkých polohách bitových premenných môže odpovedať dvom susedným celým číslam.

α_1 =(011) a α_2=(100) sú interpretované ako int(α_1)=4 resp. int(α_2)=5

Táto nevýhoda štandardného binárneho kódu je odstránená použitím tzv**. Grayovho kódu**. Jeho základná myšlienka spočíva v tom, že kóduje binárne čísla tak, že dve susedné celé čísla sú binárne reprezentované reťazcami, ktoré sú rôzne len v jednej polohe binárneho reťazca

$\tilde{\alpha}_1$=(010) a $\tilde{\alpha}_2$=(110) sú interpretované ako celé čísla int($\tilde{\alpha}_1$)=4 resp. int($\tilde{\alpha}_2$)=5



$$\tilde{\alpha}_1 = \alpha_1$$
$$\tilde{\alpha}_2 = \alpha_1 \oplus \alpha_2$$
$$\tilde{\alpha}_3 = \alpha_2 \oplus \alpha_3$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$\tilde{\alpha}_k = \alpha_{k-1} \oplus \alpha_k$$

$$\alpha_1 = \tilde{\alpha}_1$$
$$\alpha_2 = \tilde{\alpha}_1 \oplus \tilde{\alpha}_2 = \alpha_1 \oplus \tilde{\alpha}_2$$
$$\alpha_3 = \tilde{\alpha}_1 \oplus \tilde{\alpha}_2 \oplus \tilde{\alpha}_3 = \alpha_2 \oplus \tilde{\alpha}_3$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$\alpha_k = \tilde{\alpha}_1 \oplus \tilde{\alpha}_2 \oplus \ldots \oplus \tilde{\alpha}_k = \alpha_{k-1} \oplus \tilde{\alpha}_k$$

Transformácia spojitého optimalizačného problému na lineárny optimalizačný problém

Predpokladajme, že premenná $\boldsymbol{x} \in D$ je vyjadrená v binárnej reprezentácii bitovým vektorom dĺžky k. Budeme predpokladať, že dĺžka binárnej reprezentácie k je zvolená tak, že platí

$$\delta >> \frac{(b - a)}{(2^k - 1)}$$

Minimálna vzdialenosť medzi dvoma minimami funkcie $f(\boldsymbol{x})$ na oblasti D musí byť o mnoho väčšia ako "presnosť" binárnej reprezentácie.

Transformácia binernej reprezentácie na reálne číslo je formálne chápaná ako zobrazenie Γ

$$\Gamma : \{0,1\}^k \rightarrow D$$

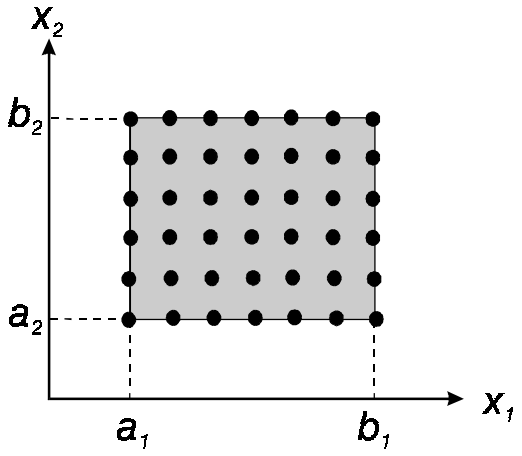
$$\boldsymbol{x} = \Gamma(\boldsymbol{\alpha})$$

Nech $\tilde{\boldsymbol{\alpha}}_{opt}$ je binárny vektor dĺžky *k*, ktorý bol získaný riešením optimalizačného problému

$$\tilde{\boldsymbol{\alpha}}_{opt} = \arg \min_{\boldsymbol{\alpha} \in \{0,1\}^{kn}} f\left(\Gamma(\boldsymbol{\alpha})\right)$$

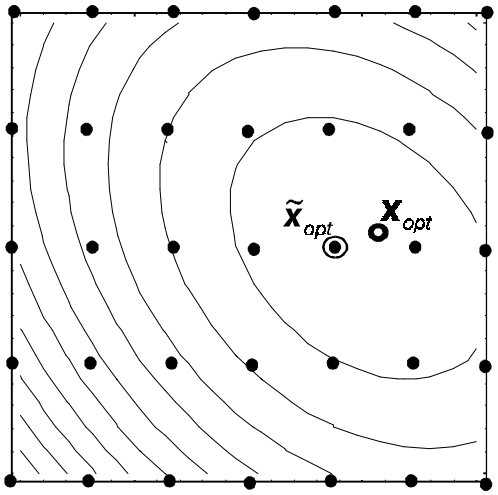
$$\tilde{\boldsymbol{x}}_{opt} = \Gamma\left(\tilde{\boldsymbol{\alpha}}_{opt}\right)$$

$$\boldsymbol{x}_{opt} \approx \tilde{\boldsymbol{x}}_{opt} = \Gamma\left(\tilde{\boldsymbol{\alpha}}_{opt}\right)$$



Závery

- Presnosť riešenia optimalizačného problému (2.2) pri prechode zo spojitej reprezentácie k binárnej reprezentácii závisí na konštante k, ktorá určuje dĺžku binárnych vektorov reprezentujúcich jednotlivé reálne premenné.
- Ak funkcia f obsahuje málo miním, ktoré sú dostatočne navzájom izolované (konštanta δ je veľká) a "široké", konštanta k nemusí byť veľká.
- Avšak, ak funkcia f obsahuje množstvo miním, ktoré ležia blízko seba (konštanta δ musí byť malá), potom konštanta k musí byt pomerne veľká.
- Ortogonálna mriežka bodov nad oblasťou D, ktoré sú generované zvolenou binárnou reprezentáciou reálnych premenných, musí byť dostatočne jemná, aby sa postihli a odlíšili blízko seba ležiace minimá funkcie f.



Kontúrový graf funkcie *f* nad oblasťou *D*, ktorá je v binárnej reprezentácii aproximovaná ortogonálnou mriežkou bodov. Presné minimum funkcie *f* nad oblasťou *D* je označené $\boldsymbol{x}_{opt}$, minimum funkcie *f* nad bodmi ortogonálnej mriežky je označené $\tilde{x}_{opt}$.

Jednoduché stochastické optimalizačné algoritmy

Slepý algoritmus a horolezecký algoritmus

Slepý algoritmus je základný stochastický algoritmus, ktorý opakovane generuje náhodne riešenie z oblasti *D* a zapamätá si ho len vtedy, ak bolo získané lepšie riešenia ako to, ktoré už bolo zaznamenané v predchádzajúcej histórii algoritmu.

procedure

```
procedure Blind_Algoritmus;
begin f_fin:=∞; t:=0;
      while t<t_max do
      begin t:=t+1;
            α:=randomly generated;
            if f(Γ(α))<f_fin then
            begin α_fin:=α;
                  f_fin:=f(Γ(α))
            end;
      end;
end;
```

Tento jednoduchý stochastický optimalizačný algoritmus poskytuje korektné globálne minimum t_{max} asymptoticky rastie do nekonečna

$$\lim_{t_{max} \to \infty} P\left(t_{max} | \alpha_{fin} = \alpha_{opt}\right) = 1$$

Slepý algoritmus *neobsahuje žiadnu stratégiu* konštrukcie riešení na základe predchádzajúcej histórie algoritmu. Každé riešenie je zostrojené úplne nezávisle (t.j. plne náhodne) od predchádzajúcich riešení. Zaznamenáva sa to riešenie, ktoré v priebehu aktivácie procedúry poskytuje zatiaľ najnižšiu funkčnú hodnotu.

Horolezecký algoritmus (hill climbing) iteračne hľadá najlepšie lokálne riešenie v určitom okolí, toto riešenie je v ďalšom kroku použité ako "stred" novej oblasti.

Operácia *mutácie* stochasticky transformuje binárny vektor α na nový binárny vektor α' , pričom stochastičnosť toho procesu je určená pravdepodobnosťou P_{mut}

$$\alpha' = O_{mut}(\alpha)$$

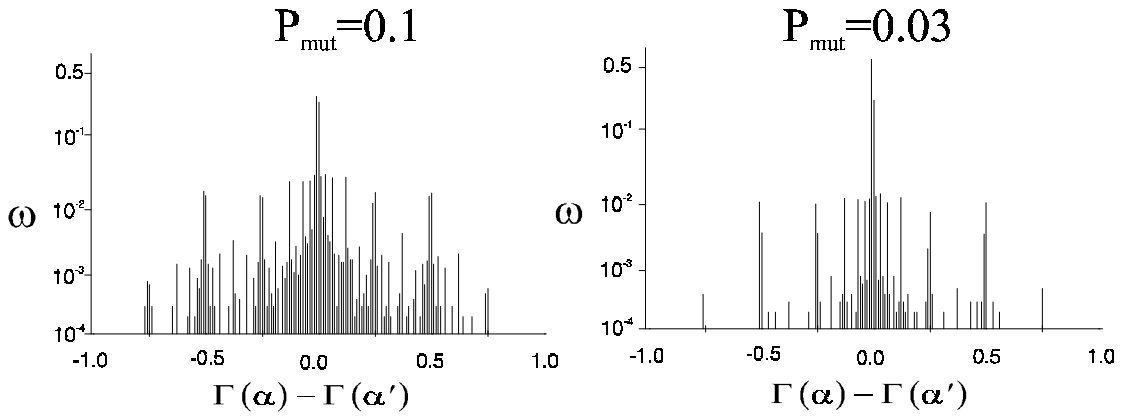
$$\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_{kn}) \quad \text{a} \quad \alpha' = (\alpha'_1, \alpha'_2, \ldots, \alpha'_{kn})$$

$$\alpha'_i = \begin{cases} 1 - \alpha_i & (\text{pre } random < P_{mut}) \\ \\ \alpha_i & (\text{ostatné prípady}) \end{cases}$$

Pravdepodobnosť P_{mut} určuje stochastičnosť operátora mutácie, v limitnom prípade, ak $P_{mut} \to 0$, potom operátor O_{mut} nemení binárny vektor

$$\lim_{P_{mut} \to 0} O_{mut}(\alpha) = \alpha$$

Štandardné binárne kódovanie



Priebehy pravdepodobností číselných hodnôt binárnych vektorov, ktoré sú generované mutáciou pre štandardné binárne kódovanie.

Základná idea *horolezeckého algoritmu* spočíva v tom, že vzhľadom k určitému zvolenému riešeniu zostrojíme náhodne predpísaný počet nových riešení tak, že vo zvolenom riešení sa náhodne zmenia bitové premenné (hovoríme, že zvolené riešenie je stred oblasti z neho náhodne generovaných riešení). Z tejto oblasti vyberieme najlepšie riešenie (t.j. s minimálnou funkčnou hodnotou nad bodmi z daného okolia), ktoré sa použije v nasledujúcom iteračnom kroku ako stred novej oblasti.

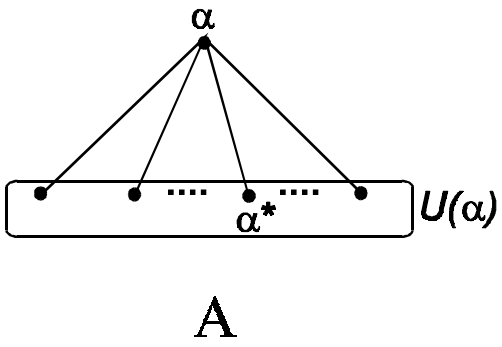
Okolie $U(\alpha)$ binárneho vektora α sa zostrojí pomocou vektorov $\alpha' = O_{mut}(\alpha)$

$$U(\alpha) = \{\alpha' = O_{mut}(\alpha)\}$$

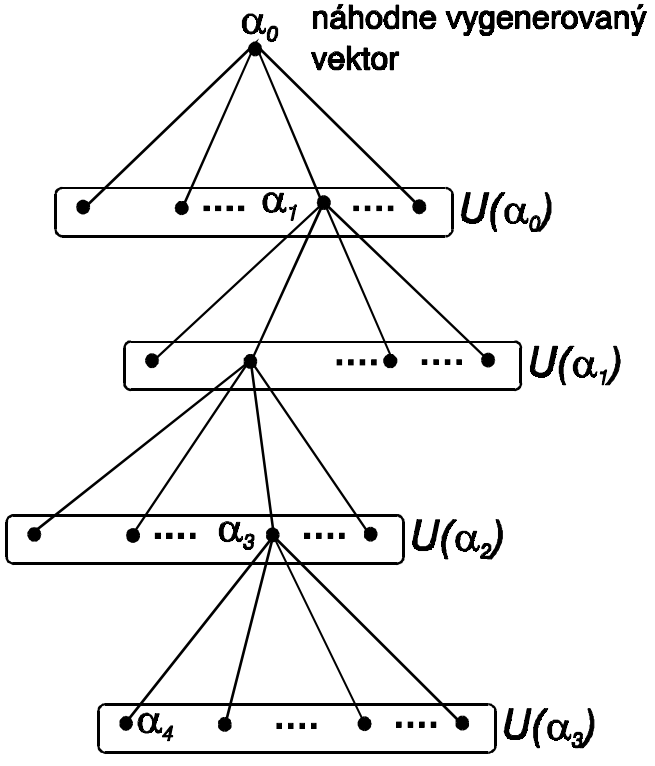
Najlepšie riešenie v okolí $U(\alpha)$ je určené takto

$$\alpha^* = \arg \min_{\alpha' \in U(\alpha)} f\left(\Gamma(\alpha')\right)$$

V horolezeckom algoritme sa takto získané riešenie α^* použije ako "stred" v ďalšom iteračnom kroku.



A



B

```
procedure Hill_Climbing;
begin α:=randomly generated;
      f_fin:=∞; t:=0;
      while t<t_max do
      begin t:=t+1;
            α* = arg min_{α'∈U(α)} f(Γ(α))
            if f(Γ(α*)) < f_fin then
            begin f_fin:= f(Γ(α*));
                  α_fin:=α*
            end;
            α:=α*;
      end;
end;
```

Horolezecký algoritmus s učením

Horolezecký algoritmus s učením je jednoduchá modifikácia štandardného horolezeckého algoritmu, ktorá sa dotýka konštrukcie okolia $U(\alpha)$.

Zavedieme dva nové koncepty, ktoré umožňujú modifikovať horolezecký algoritmus

(1) *Pravdepodobnostný vektor*

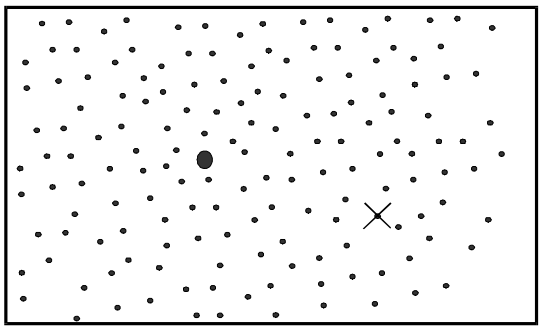
$$\boldsymbol{w}=(w_1,w_2,\ldots w_{kn})\in[0,1]^{kn}.$$

Jeho komponenty $0\le w_i \le 1$ určujú pravdepodobnosti výskytu premennej ’1’ v danej pozícii.

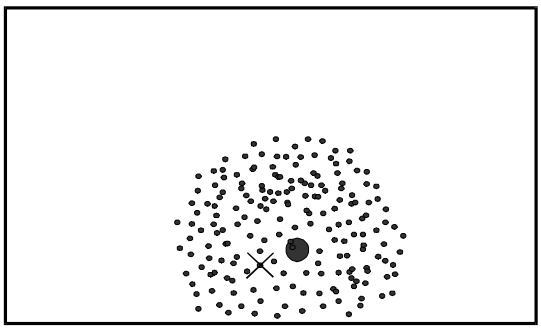
$$\alpha_i = \begin{cases} 1 & (\text{ak } random < w_i) \\ 0 & (\text{opaèný prípad}) \end{cases}$$

Okolie $U(\boldsymbol{w})$ zostrojené z binárnych vektorov náhodne generovaných vzhľadom k pravdepodobnostnému vektoru $\boldsymbol{w}$ je určené vzťahom

$$U(\boldsymbol{w}) = \{\alpha = R(\boldsymbol{w})\}$$



A



B

w_i=0.5

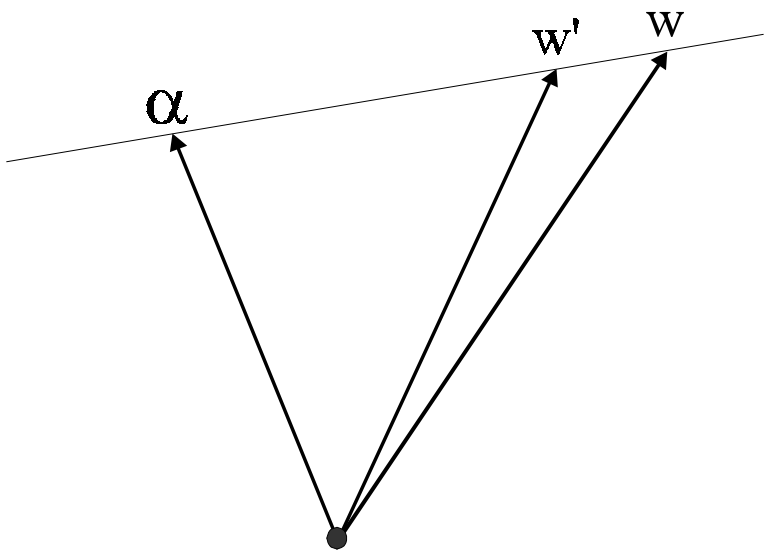
w_i=0 alebo 1

(2) *Učenie* pravdepodobnostného vektora ***w***. Nech *B*(***w***) je podmnožina, ktorá obsahuje *b* najlepších riešení z okolia *U*(***w***)

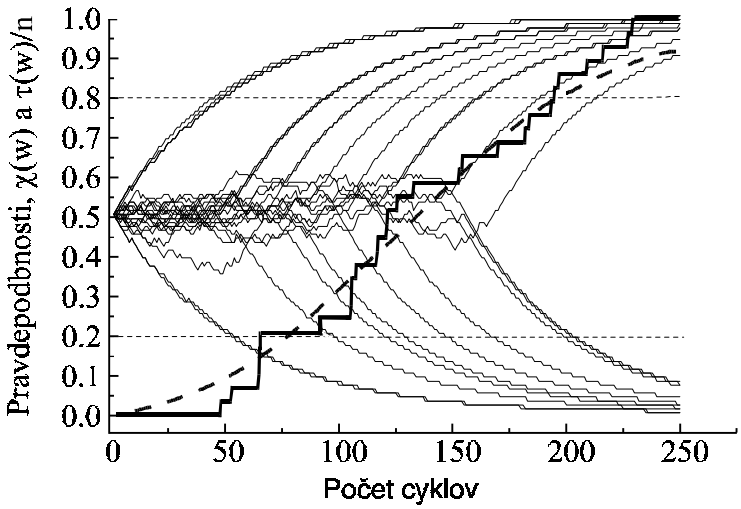
$$B(\boldsymbol{w}) = \arg \min_{\alpha \in U(\boldsymbol{w})} f\left(\Gamma(\alpha)\right)$$

Pravdepodobnostný vektor je modifikovaný - *učený* pomocou Hebbovho pravidla

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \lambda \sum_{\alpha \in B(\boldsymbol{w})} (\alpha - \boldsymbol{w})$$



```
procedure HCwL;
begin for i:=1 to n do w_i:=0.5;
      time:=0;
      while time<time_max do
      begin time:=time+1;
            B(w):= arg min_{α∈U(w)} f(Γ(α))
            w:= w + λ Σ_{α∈B(w)} (α − w);
      end;
      α_fin:=best solution
            of B(w);
end;
```



Metóda zakázaného hľadania (tabu search)

Metóda *zakázaného hľadania* (*tabu search*) bola navrhnutá koncom 80-tých rokov Gloverom [8,9] ako určité zovšeobecnenie horolezeckého algoritmu pre riešenie zložitých optimalizačných úloh menovite z operačného výskumu.

Metóda vychádza z horolezeckého algoritmu. Základnou nevýhodou tohto algoritmu je, že sa po určitom počte iteračných krokov vracia k lokálnemu optimálnemu riešeniu, ktoré sa vyskytlo už v jeho predchádzajúcom priebehu (problém zacyklenia).

Glover navrhol jednoduchú heuristiku ako odstrániť tento problém. Do horolezeckého algoritmu je zavedená tzv. *krátkodobá pamäť*, ktorá si pre určitý krátky interval predchádzajúcej histórie algoritmu pamätá inverzné transformácie k tým transformáciám riešení, ktoré poskytovali lokálne optimálne riešenia. Tieto inverzné transformácie sú zakázané (tabu) pri tvorbe nového okolia pre dané aktuálne riešenie.

Definujme si množinu prípustných transformácií

$$S = \{t_1, t_2, \ldots, t_p\}$$

$$t : \{0,1\}^k \rightarrow \{0,1\}^k$$

pre $\forall t \in S$.

Jednoduchá realizácia týchto transformácií je

$$t_i(\ldots\alpha_i\ldots) = (\ldots 1-\alpha_i\ldots)$$

Operátor t_i zmení v *i*-tej polohe binárnu hodnotu na jej komplement.

Okolie U(α) obsahuje obrazy α vytvorené transformáciami $t \in S$

$$U(\alpha) = \{t\alpha;\ \forall t \in S\}$$

Zakázaný zoznam T (tabu list), je krátkodobá pamät, ktorá dočasne obsahuje inverzné transformácie k použitým transformáciám v predchádzajúcich iteráciach. Ak transformácia *t* patrí pre danú iteráciu do zakázaného zoznamu, $t \in T$, potom sa **nesmie** používať v lokálnej minimalizácii v rámci okolia aktuálneho riešenia α.

Numerické skúsenosti s algoritmom zakázaného hľadania ukazujú, že **veľkosť zakázaného zoznamu** je veľmi dôležitým parametrom pre prehľadávanie oblasti $\{0,1\}^k$ s možnosťou vymaniť sa z lokálnych miním.

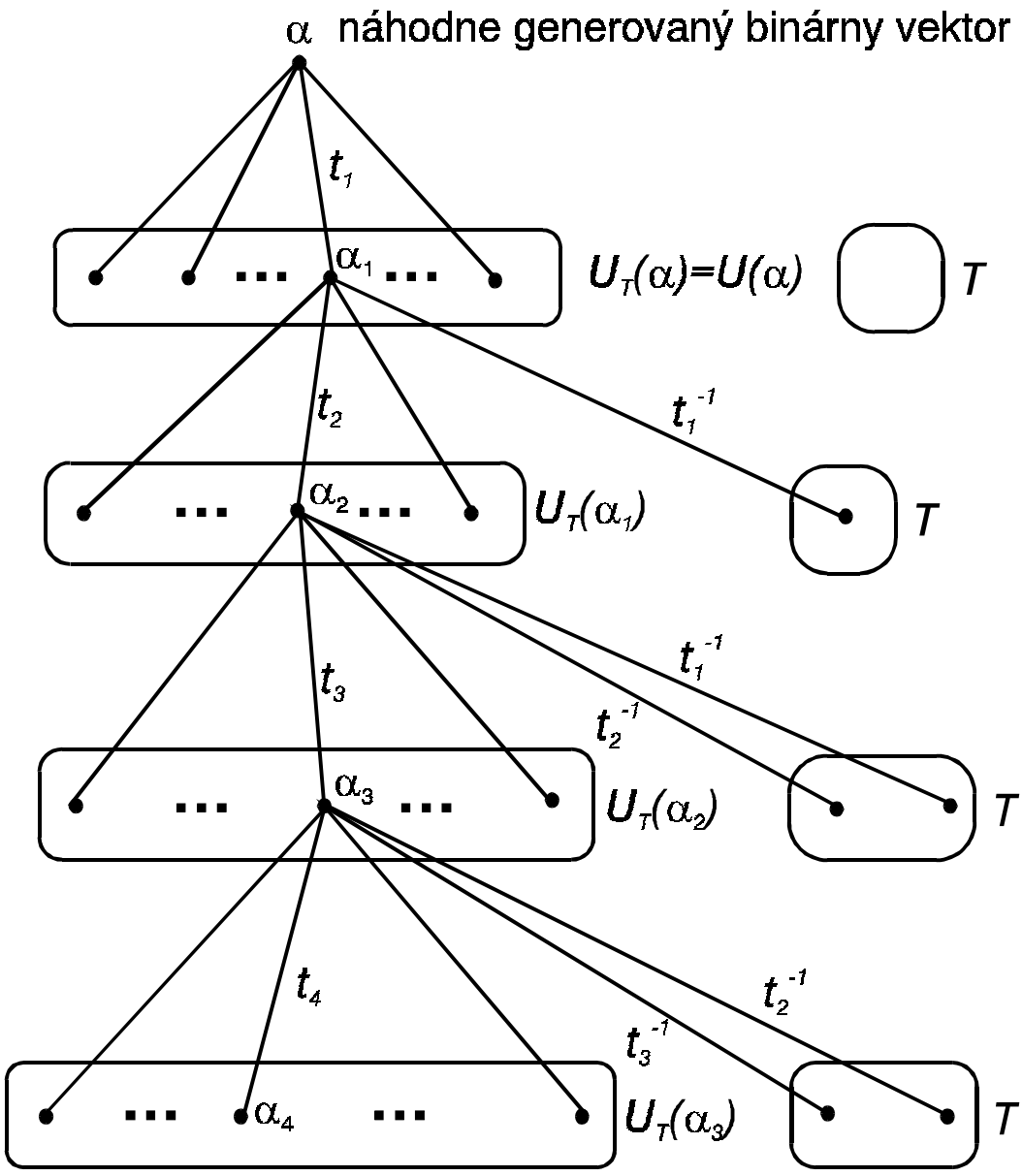
Ak je parameter *s* malý, potom sa môže

Zakázaný zoznam T sa používa pre konštrukciu modifikovaného okolia $U_T(\alpha)$

$$U_T(\alpha) = \{\alpha'; \forall t \in S \setminus T : \alpha' = t\alpha\}$$

pričom $U_T(\alpha)=U(\alpha)$, pre $T=\varnothing$. Toto okolie obsahuje vektory $\alpha' \in \{0,1\}^k$, ktoré sú vytvorené použitím transformácií z množiny *S* nepatriacich do zakázaného zoznamu *T*.

```
procedure Tabu_Search;
begin α:=randomly generated;
      f_fin:=∞; time:=0; T:=∅;
      while t<time_max do
      begin time:=time+1;
            f_fin-loc:=∞;
            for t∈S do
            begin α':=tα;
                  if t∉T and
                     (f(Γ(α'))<f_fin-loc)
                      or f(Γ(α'))<f_fin)
                  then
                  begin α*:=α';
                        t*:=t;
                        f_fin-loc:=f(Γ(α'))
                  end;
            end;
            if f_fin-loc<f_fin then
            begin f_fin:=f_fin-loc;
                  α_fin:=α*
            end;
            α:=α*;
            if |T|<s then
            T:=T∪{t*^-1} else
            T:=(T∪{t*^-1})\{t̂};
      end;
end;
```


α náhodne generovaný binárny vektor
t1
α1
UT(α)=U(α)
T
t2
t1-1
α2
UT(α1)
T
t3
t1-1
t2-1
α3
UT(α2)
T
t4
t2-1
t3-1
α4
UT(α3)
T

Evolučné programovanie

Evolučné programovanie (Fogel) patrí medzi stochastické optimalizačné algoritmy, ktoré sú jednoduchým zovšeobecnením horolezeckého algoritmu.

Budeme prezentovať zjednodušenú verziu evolučného programovania, ktorá ovšem pokrýva dostatočne všeobecne vlastnosti tejto metódy.

Nech *P* je množina - populácia riešení tvaru

$$P = \{\alpha_1, \alpha_2, \ldots, \alpha_p\} \subseteq \{0,1\}^k$$

Každý element α populácie z populácie *P* je ohodnotený funkcčnou hodnotou $f(\alpha)$.

Z populácie *P* vyberieme podmnožninu - podpopuláciu rodičov $Q \subseteq P$, Riešenia z podpopulácie Q sú transformované mutačným operátorom O_{mut} na podpopuláciu potomkov

$$Q' = \{\alpha' = O_{mut}(\alpha) ; \alpha \in Q\}$$

Potomkovia z podpopulácie *Q'* sú ohodnotené funkčnými hodnotami $f(\alpha)$. Podpopulácie *Q* a *Q'* sú zjednotené do spoločnej množiny obsahujúcej všetkých rodičov a ich potomkov

$$R = Q \cup Q'$$

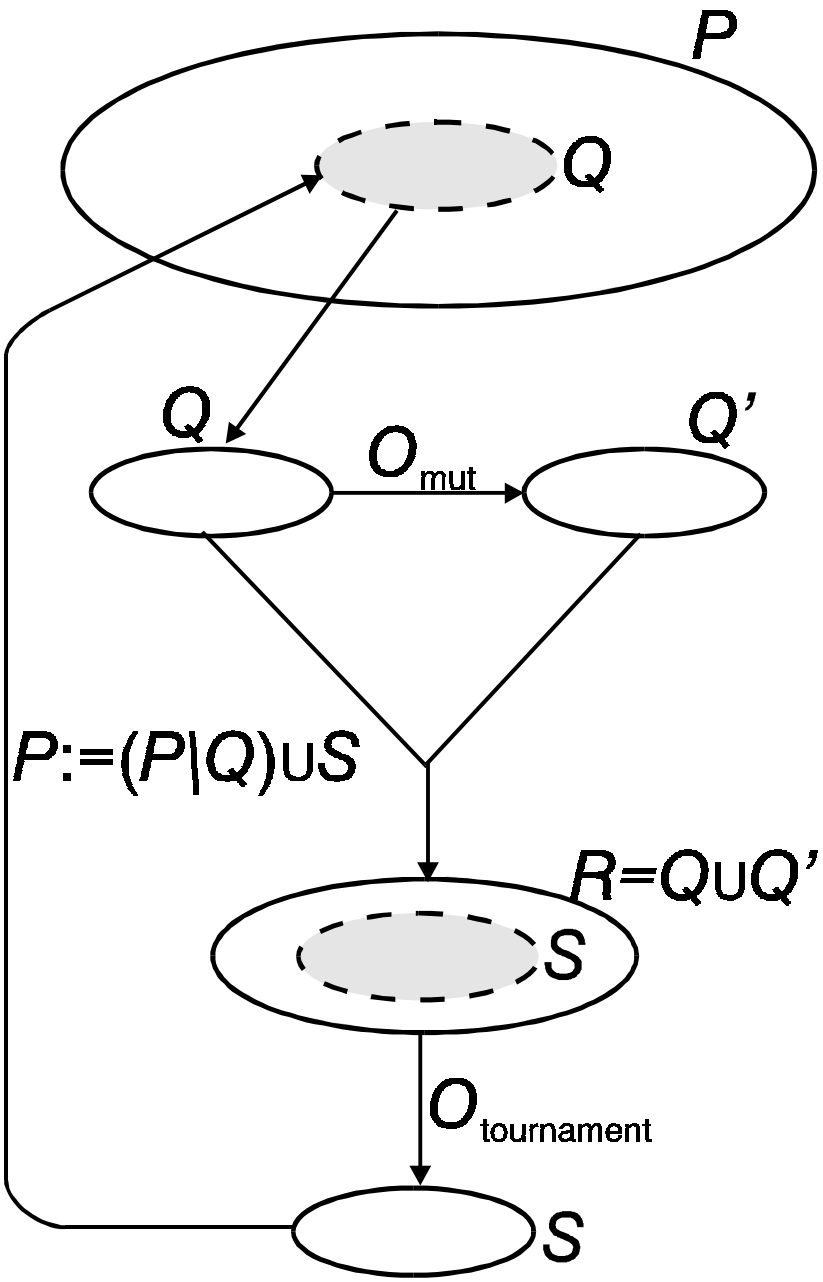
Z tejto zjednotenej podpopulácie vytvoríme novú podpopuláciu nasledovníkov S, pričom $|S|=|Q|=|Q'|$. Tento výber sa realizuje pomocou operátora "turnaja" $O_{tournament}$

$$S = O_{tournament}(R)$$

Najjednoduchší prístup k realizácii "turnaja" spočíva v usporiadaní elementov *R* podľa rastúcich funkčných hodnôt $f(\alpha)$, potom S je tvorená prvými $|Q|$ elementami takto usporiadanej množiny *R*.

Pomocou množiny *S* masledovníkov, obnovíme populáciu *P* takto

$$P \leftarrow (P \setminus Q) \cup S$$


P
Q
Q
Q'
O mut
P:=(P|Q)∪S
R=Q∪Q'
S
O tournament
S

Schématické znázornenie evolučného programovania.

```
procedure Evolutionary_Programmning;
begin P:=randomly generated
         population;
      stop_criterion:=false;
      while not stop_criterion do
      begin Q:=randomly selected
               subpopulation of P;
            Q':=O_mut(Q);
            R:=Q∪Q';
            S:=O_tournament(R);
            P:=(P\Q)∪S;
            if convergence criteria
               are fulfilled then
            stop_criterion:=true;
      end;
      α_opt = arg min_{α∈P} f(α);
end;
```