3. Horolezecké algoritmy

3.1 Základné stochastické optimalizačné algoritmy: *slepý algoritmus* a *horolezecký algoritmus*

Uvedieme dva základné typy stochastických optimalizačných algoritmov, ktoré aj keď neobsahujú evolučné rysy, budú slúžiť ako základ pre formuláciu evolučných optimalizačných algoritmov. *Slepý algoritmus* je základný stochastický algoritmus, ktorý opakovane generuje náhodne riešenie z oblasti *D* a zapamätá si ho len vtedy, ak bolo lepšie riešenie ako to, ktoré už bolo zaznamenané v predchádzajúcej histórii algoritmu. Z dôvodov kompatibility tohto algoritmu s evolučnými algoritmami uvedieme jeho implementáciu pre binárnu reprezentáciu vektorov - riešení (pozri algoritmus 3.1).

```
procedure Blind_Algoritmus(input: tmax,k,n; output: αfin,ffin);
begin ffin:=∞; t:=0;
      while t<tmax do
      begin t:=t+1;
            α:=náhodne generovaný binárny
               vektor dĺžky kn;
            if f(Γ(α))<ffin then
            begin αfin:=α; ffin:=f(Γ(α)) end;
      end;
end;
```

Algoritmus 3.1. Pseudopascalovská implementácia slepého algoritmu. Vstupné parametre procedúry sú t_{max} (maximálny počet iterácií) a konštanty *k* a *n* (dĺžka binárneho reťazca jednotlivej premennej, resp. počet premenných optimalizovanej funkcie *f*). Algoritmus začína inicializáciou premenných f_{fin} (výsledná hodnota nájdeného minima funkcie *f*) a *t* (počítadlo iterácií). Algoritmus sa opakuje t_{max}-krát, potom je ukončený a výstupné parametre α_{fin} a f_{fin} obsahujú najlepšie hodnoty riešenia v binárnej reprezentácii a príslušnú najlepšiu funkčnú hodnotu.

Jednoduchými úvahami sa dá dokázať, že tento jednoduchý stochastický optimalizačný algoritmus poskytuje korektné globálne minimum optimalizačného problému (2.5) realizovaného nad ortogonálnou mriežkou bodov z oblasti *D* za predpokladu, že parameter procedúry t_{max} asymptoticky rastie do nekonečna

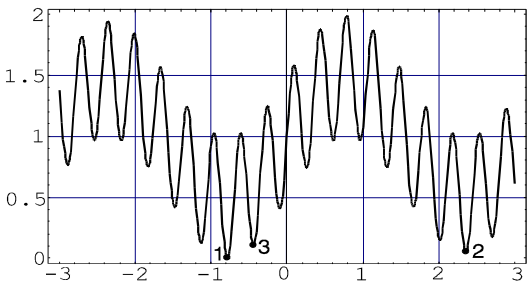
$$\lim_{k_{max}\to\infty} P\left(t_{max} \mid \boldsymbol{\alpha}_{fin} = \boldsymbol{\alpha}_{opt}\right) = 1 \qquad (3.1)$$

kde $P(t_{max}|\boldsymbol{\alpha}_{fin}=\boldsymbol{\alpha}_{opt})$ je pravdepodobnosť toho, že slepý algoritmus po t_{max} iteračných krokoch poskytne výstupné riešenie, ktoré je totožné s presným riešením (globálne minimum).

Príklad 3.1. *Ako príklad vysoko multimodálnej funkcie budeme používať túto funkciu*

$$f(x) = 0.993851231 + e^{-0.01x^2}\, sin(10x)\, cos(8x)$$

pre $x\in[-10,10]$. Výsek jej priebehu pre $x\in[-3,3]$ má tento tvar



Prvé tri minimá tejto funkcie sú: x_1=-0.7853024, $f(x_1)=5.69\times10^{-11}$ (globálne minimum), x_2=2.3559072, $f(x_2)$= 0.047848, x_3=-0.4402184, $f(x_3)$=0.11277. Konštanta δ, ktorá charakterizuje minimálnu vzdialenosť medzi dvoma minimami (pozri obr. 2.2), je približne rovná δ=1/4. To znamená, že neexistuje taká dvojica dvoch susedných miním, ktorých vzdialenosť by bola menšia ako δ=1/4. V nasledujúcich príkladoch sa táto funkcia bude často používať tak v 1-rozmernej, ako aj v mnohorozmernej verzii pre testovanie schopnosti nájsť globálne minimum. Overte uvedené údaje o tejto funkcii pomocou programu MAPLE [1] alebo MATHEMATICA [2]. Koľko lokálnych miním a lokálnych extrémov má táto funkcia?

Príklad 3.2. *Zovšeobecnite program z príkladu 3.1 pre funkciu n premenných, pričom každá premenná je vyjadrená binárnym reťazcom rovnakej dĺžky "k" a všetky premenné sú z rovnakého intervalu. Ako testovaciu funkciu môžete použiť zovšeobecnenú funkciu z pr9kladu 3.1*

$$F\left(x_1, x_2, ...x_n\right) = \sum_{i=1}^{n} f\left(x_i\right)$$

pre $\forall x_i \in$[-10,10]. Aké je globálne minimum tejto funkcie ? Koľko má lokálnych miním ?

Príklad 3.3. *Zovšeobecnite program pre slepý algoritmus (pozri algoritmus 3.1) pre funkciu f(x) z príkladu 3.1, ak dĺžka binárnej reprezentácie bude k=10, 20, 30. Zostrojte tabuľku výsledkov, v ktorej bude uvedené najlepšie zaznamenané minimum funkcie vzhľadom na počet iteračných krokov t_{max}=100, 1000, 10000. Diskutujte získané výsledky vzhľadom na dĺžku "k" binárnej reprezentácie a počet iteračných krokov t_{max}.*

Vo všeobecnosti môžeme povedať, že slepý algoritmus *neobsahuje žiadnu stratégiu* konštrukcie riešení (t.j. binárnych vektorov dĺžky *kn*) na základe predchádzajúcej histórie algoritmu. Každé riešenie je zostrojené úplne nezávisle (t.j. celkom náhodne) od predchádzajúcich riešení. Zaznamenáva sa to riešenie, ktoré v priebehu aktivácie procedúry poskytuje zatiaľ najnižšiu funkčnú hodnotu. Po ukončení aktivácie procedúry je toto riešenie výstupným parametrom.

Slepý algoritmus sa môže jednoducho zovšeobecniť na tzv. *horolezecký algoritmus* (hill climbing), kde sa iteračne hľadá najlepšie lokálne riešenie v určitom okolí a toto riešenie sa v ďalšom kroku použije ako "stred" novej oblasti. Na formalizácii horolezeckého algoritmu zavedieme niektoré základné pojmy, ktoré sú dôležité pre jeho jednoduchý popis. Operácia *mutácie* stochasticky transformuje binárny vektor $\boldsymbol{\alpha}$ na nový binárny vektor $\boldsymbol{\alpha}'$, pričom stochastičnosť toho procesu je určená pravdepodobnosťou P_{mut}

$$\boldsymbol{\alpha}' = O_{mut}\left(\boldsymbol{\alpha}\right) \tag{3.2a}$$

kde α a α' sú dva binárne vektory rovnakej dĺžky *kn*

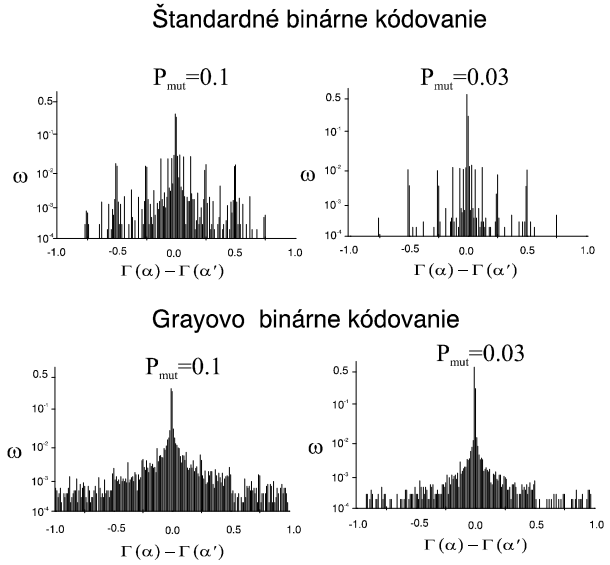
$$\boldsymbol{\alpha} = \left(\alpha_1, \alpha_2, ..., \alpha_{kn}\right) \text{ a } \boldsymbol{\alpha}' = \left(\alpha_1', \alpha_2', ..., \alpha_{kn}'\right) \tag{3.2b}$$

kde jednotlivé komponenty α' sú určené takto

$$\alpha_i' = \begin{cases} 1-\alpha_i \ (\text{pre} \ random < P_{mut}) \\ \\ \alpha_i \ (\text{ostatné prípady}) \end{cases} \tag{3.2c}$$

kde *random* je náhodné číslo z intervalu [0,1) generované s rovnomernou distribúciou (pozri algoritmus 3.2). Pravdepodobnosť P_{mut} určuje stochastičnosť operátora mutácie, v limitnom prípade ak $P_{mut} \to 0$, potom operátor O_{mut} nemení binárny vektor

$$\lim_{P_{mut} \to 0} O_{mut}(\boldsymbol{\alpha}) = \boldsymbol{\alpha}' \tag{3.2d}$$



Obrázok 3.1. Priebehy pravdepodobností číselných hodnôt binárnych vektorov, ktoré sú generované mutáciou pre štandardné a Grayovo kódovanie. Binárne vektory majú dĺžku k=30, pričom reprezentujú reálne čísla z intervalu [0,1]. Grafy sú zostrojené tak, že 10000-krát sa náhodne vygeneroval bitový vektor α, aplikovaním mutácie s pravdepodobnosťou P_{mut} sa vygeneroval nový binárny vektor $\alpha'=O_{mut}(\alpha)$. Na horizontálnej osi sa vynášajú rozdiely číselných hodnôt $-1 \le \Gamma(\alpha)-\Gamma(\alpha') \le 1$. Na vertikálnej osi sa vynášajú pravdepodobnosti ω výskytu rozdielu $\Gamma(\alpha)-\Gamma(\alpha')$.

Na obr. 3.1 je znázornený efekt mutácie (3.2) (pre dve pravdepodobnosti P_{mut}=0.1 a P_{mut}=0.03) na binárny vektor dĺžky *k*=30, ktorý reprezentuje reálne čísla z intervalu [0,1]. Použili sa dva rôzne prístupy ku kódovaniu, a to štandardné kódovanie a Grayovo kódovanie. Z obrázka je vidieť, že mutácia pre binárne vektory so štandardným kódovaním vytvára nové binárne vektory s číselnými hodnotami, ktoré sú rozdelené diskrétnym spôsobom okolo nulovej hodnoty. Ak sa použije Grayovo kódovanie, číselné hodnoty nových binárnych vektorov vytvárané mutáciou sú rozdelené pomerne "spojito" okolo nulovej hodnoty približne s Gaussovou distribúciou pravdepodobnosti. Môžeme teda povedať, že mutácia v rámci štandardného kódovania poskytuje nové binárne vektory, ktoré vzhľadom na pôvodné (t.j. nemutované) binárne vektory majú číselné hodnoty, ktoré sú relatívne diskrétne rozdelené okolo nuly. Táto "diskrétnosť rozdelenia" je potlačená, ak sa použije Grayov kód pri binárnej reprezentácii reálnych čísel. Na základe tohto výsledku možno konštatovať, že Grayov kód je

zrejme vhodnejší pre binárnu reprezentáciu reálnych premenných, mutáciou vytvorené binárne vektory sa vyskytujú "spojito" v celej oblasti prípustných hodnôt (pozri obr. 3.1).

Príklad 3.4. *Zreprodukujte grafy z obr. 3.1. Zvolíme si dĺžku "k" binárnej reprezentácie reálnych čísel z intervalu [a,b], kde a<b. Binárny reťazec α je náhodne generovaný, aplikovaním operátora mutácie O_{mut} dostaneme nový binárny reťazec α', $\alpha'=O_{mut}(\alpha)$. Pôsobenie operátora mutácie sa realizuje vzťahmi (3.2a-c). Zostrojíme rozdiel $\Delta(\alpha,\alpha')=\Gamma(\alpha)-\Gamma(\alpha')$, ktorý je z intervalu [a-b,b-a], pričom dĺžka tohto intervalu je 2(b-a). Výpočet reálneho čísla priradeného binárnemu reťazcu je realizovaný tak v štandardnej, ako aj v Grayovej reprezentácii. Nech je tento interval rozdelený na N podintervalov, pričom prvý (posledný) interval je indexovaný 1(N), pričom dĺžka týchto podintervalov je $\xi = 2(b-a)/N$. Reálnemu číslu $z\in$[a-b,b-a] priradíme index podľa toho prepisu*

$$index(z) = 1 + \left\lceil \frac{z - a + b}{\xi} \right\rceil$$

kde $\lceil x \rceil$ je celá časť reálneho čísla x. Ak položíme $z=\Delta(\alpha,\alpha')$, potom pre tento rozdiel dostaneme

$$index\left(\Delta(\alpha,\alpha')\right) = 1 + \left\lceil \frac{\Delta(\alpha,\alpha') - a + b}{\xi} \right\rceil$$

Frekvencie výskytu ω priradené rozdielom $\Delta(\alpha,\alpha')$ sa približne spočítajú tak, že namiesto funkčnej hodnoty tohto rozdielu sa uvažuje index tohto rozdielu, t.j. interval možných funkčných hodnôt intervalu [a-b,b-a] sa diskretizuje pomocou N bodov. Algoritmus sa inicializuje tak, že všetky frekvencie výskytu sú nulové, v priebehu výpočtu sa obnovujú len tie frekvencie, ktoré sú určené indexom index($\Delta(\alpha,\alpha')$). Tento postup je vyjadrený nasledujúcim algoritmom

```
for i:=1 to N do ωi;=0;
time:=0
while time<time_max do
begin time:=time+1;
      α:= náhodne generovaný binárny vektor;
      α':=O_mut(α);
      Δ(α,α'):=Γ(α)-Γ(α');
      index:= 1 + ⌈ (Δ(α,α') - a + b) / ξ ⌉ ;
      ω_index:=ω_index+1;
end;
for i:=1 to N do ωi:=ωi/N;
```

Grafy z obr. 3.1 dostaneme tak, že pomocou vhodného softvéru vynesieme hodnoty frekvencií ω_i vzhľadom na indexy (pre lepšiu názornosť grafu sa namiesto indexu používa príslušná reálna hodnota určená vzťahom

$$real(index) = a - b + (index - 1) \cdot \xi$$

Výpočty pomocou tohto algoritmu realizujte pre rôzne hodnoty k (dĺžky binárnych reťazcov) a P_{mut} (pravdepodobnosť jednobitovej mutácie). Pokúste sa diskutovať výsledky.

Základná idea *horolezeckého algoritmu* spočíva v tom, že vzhľadom na určité zvolenému riešenie 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. Tento proces sa opakuje predpísaný počet-ráz, pričom sa zaznamenáva najlepšie riešenie, ktoré sa vyskytlo v priebehu histórie algoritmu. (Možná modifikácia tohto algoritmu spočíva v prehľadávaní všetkých riešení, ktoré sa líšia od aktuálneho riešenia v jednom bite.)

```
procedure Mutation_Bin(input:α; output α');
begin for i:=1 to kn do
      if random<P_mut then
      α'_i:=1-α_i else α'_i:=α_i;
end;
```

Algoritmus 3.2. Implementácia mutácie binárneho reťazca dĺžky *kn*. Pravdepodobnosť P_{mut} určuje 1-bitovú mutáciu, t.j. zmenu bitu na jeho komplement. Premenná random je náhodné číslo s rovnomernou distribúciou z intervalu [0,1).

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

$$U(\boldsymbol{\alpha}) = \{\boldsymbol{\alpha}' = O_{mut}(\boldsymbol{\alpha})\} \tag{3.3}$$

pričom budeme predpokladať, že kardinalita (počet elementov) sa rovná predpísanej hodnote, $|U(\boldsymbol{\alpha})| = c_0$, kde c_0 je dané kladné celé číslo. Poznamenajme, že v dôsledku stochastičnosti aplikácie operácie mutácie na daný binárny vektor α má zloženie okolia $U(\boldsymbol{\alpha})$ tiež stochastický charakter. To, či nejaký vektor $\boldsymbol{\alpha}'$ patrí alebo nepatrí do okolia $U(\boldsymbol{\alpha})$, je určené len pravdepodobnostne, a nie deterministicky. Najlepšie riešenie v okolí $U(\boldsymbol{\alpha})$ je určené takto

$$\boldsymbol{\alpha}^* = \arg \min_{\boldsymbol{\alpha}' \in U(\boldsymbol{\alpha})} f(\Gamma(\boldsymbol{\alpha}')) \tag{3.4}$$

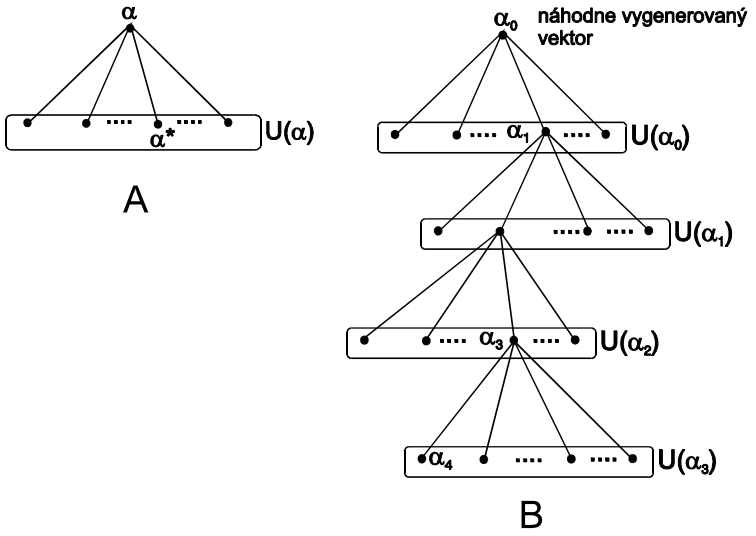
V horolezeckom algoritme sa takto získané riešenie $\boldsymbol{\alpha}^*$ použije ako "stred" v ďalšom iteračnom kroku algoritmu, pozri obr. 3.2. Implementácia horolezeckého algoritmu v pseudopascale je uvedená v algoritme 3.3.

Analóg vzorca (3.1) zo slepého algoritmu, ktorý hovorí, že tento jednoduchý algoritmus je asymptoticky schopný nájsť globálne minimum, platí aj v horolezeckom algoritme. V tomto prípade sa avšak kardinalita okolia $c_0=|U(\boldsymbol{\alpha})|$ musí asymptoticky zväčšovať do nekonečna

$$\lim_{c_0 \to \infty} P(c_0 \mid \boldsymbol{\alpha}_{fin} = \boldsymbol{\alpha}_{opt}) = 1 \tag{3.5}$$

Potom je však zbytočné opakovať iteračné kroky horolezeckého algoritmu pre nové lokálne optimálne riešenia, už v rámci jedného iteračného kroku získame globálne riešenie pre $c_0 \to \infty$.

Ako naznačujú jednoduché numerické aplikácie, horolezecký algoritmus, aj keď explicitne neobsahuje evolučnú stratégiu, je pomerne efektívny a robustný stochastický optimalizačný algoritmus, ktorý je schopný pre jednoduchšie úlohy nájsť globálne minimum. V nasledujúcej časti tejto kapitoly uvedieme dve jednoduché zovšeobecnenia horolezeckého algoritmu, ktoré sú už blízke evolučným algoritmom a môžu sa považovať ako určitý prototyp týchto algoritmov.



Obrázok 3.2. Schematické znázornenie generovania okolia binárneho vektora α a najlepšieho riešenia α^* v okolí $U(\alpha)$ (diagram A). Počet binárnych vektorov v okolí je konštantný, rovná sa c_0. Horolezecký algoritmus je znázornený na diagrame B, tento algoritmus pozostáva z tvorby postupností okolí $U(1)$, $U(2)$, $U(3)$, ..., Stred okolia $U(i)$ je totožný s najlepším riešením z predchádzajúceho okolia $U(i-1)$. Algoritmus sa inicializuje riešením α_0, ktoré sa náhodne generuje.

```
procedure Hill_Climbing(input:tmax,c0,Pmut; output: ffin,αfin);
begin α:= náhodne generovaný binárny vektor dĺžky kn;
      ffin:=∞; t:=0;
      while t<tmax do
      begin t:=t+1;
            α* = arg min_{α'∈U(α)} (Γ(α))
            if f(Γ(α*)) < ffin then
            begin ffin:=f(Γ(α*));
                  αfin:=α*
            end;
            α:=α*;
      end;
end;
```

Algoritmus 3.3. Pseudopascalovská implementácia procedúry realizujúcej horolezecký algoritmus. Vstupnými parametrami sú konštanty t_{max}, c_0 a P_{mut}, ktoré opisujú maximálny počet iterácií horolezeckého algoritmu, kardinalitu okolia $U(\alpha)$, resp. pravdepodobnosť 1-bitovej mutácie. Algoritmus sa inicializuje náhodným generovaním binárneho vektora α, ktorého dĺžka je kn (kde k je dĺžka binárnej reprezentácia reálnej premennej a n je počet premenných optimalizovanej funkcie f). Binárny vektor α^* je najlepšie riešenie nájdené v okolí $U(\alpha)$, toto riešenie sa v nasledujúcom kroku použije ako stred nového okolia. Najlepšie riešenie získané v priebehu celej histórie je uložené vo výstupných premenných f_{fin} a α_{fin}.

Príklad 3.5. *Napíšte program pre horolezecký algoritmus (pozri algoritmus 3.3) pre funkciu f(x) definovanú v príklade 3.3 a jej n-rozmerné zovšeobecnenie (pozri príklad 3.2). Vektor reálnych premenných* ***x*** *je binárne reprezentovaný tak v štandardnom, ako aj v Grayovom kódovaní. Pokúste sa pre n=1 optimalizovať parametre* c_0 *a* P_{mut} *tak, aby ste skoro so 100% istotou dostali globálne minimum s čo najmenším počtom iteračných krokov. Použite tieto optimálne hodnoty parametrov aj pre n=2,3,4. Zistite, či dostávate so skoro 100% istotou globálne minimum? Pokúste sa pre tieto viacrozmerné prípady optimalizovať parametre* c_0 *a* P_{mut}*, tak, aby ste s vysokou pravdepodobnosťou dostali globálne minimum. Vyneste do tabuľky (alebo grafu) tieto optimálne hodnoty parametrov* c_0 *a* P_{mut} *pre n=1,2,3,4, diskutujte túto tabuľku.*

3.2 Horolezecký algoritmus s učením

Horolezecký algoritmus s učením [3-6] patrí medzi jednoduché modifikácie štandardného horolezeckého algoritmu. Táto modifikácia (podobne ako pri metóde zakázaného hľadania) sa dotýka konštrukcie okolia $U(\boldsymbol{\alpha})$. Pôvodná definícia okolia (3.3) využíva stochastický operátor mutácie O_{mut}, zo "stredu" α sa pomocou tohto operátora generujú nové binárne operátory, pričom pravdepodobnosť zmeny binárnej hodnoty na jej komplement je určená pravdepodobnosťou P_{mut} (pozri (3.2b-c)). V prípade, že pravdepodobnosť mutácie je malá ($P_{\text{mut}}\approx 0$), potom nové stavy generované mutačným operátorom sú veľmi blízke pôvodnému stavu α (t.j. stredu okolia $U(\boldsymbol{\alpha})$). Opačne, ak sa pravdepodobnosť P_{mut} blíži k 1/2, potom okolie $U(\boldsymbol{\alpha})$ obsahuje binárne vektory, ktorú sú veľmi vzdialené od "stredu" $\boldsymbol{\alpha}$. Táto jednoduchá úvaha nás vedie k myšlienke konštrukcie okolia tak, že pre každú polohu bitového vektora máme zadanú zvlášť pravdepodobnosť. Zavedieme dva nové koncepty, ktoré zaujímavým spôsobom umožňujú modifikovať horolezecký algoritmus

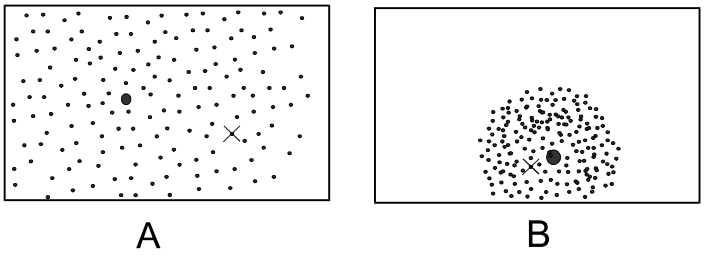
(1) *Pravdepodobnostný vektor* $\boldsymbol{w}=(w_1,w_2,\ldots w_{\text{kn}})\in[0,1]^{\text{kn}}$. Jeho jednotlivé zložky $0\le w_i \le 1$ určujú pravdepodobnosti výskytu premennej '1' v danej pozícii. Napr. ak $w_i=0(1)$, potom $\alpha_i=0(1)$, pre $0<w_i<1$, potom premenná α_i je náhodne určená vzťahom

$$\alpha_i = \begin{cases} 1\,(\text{ak } random < w_i) \\ 0\,(\text{opačný prípad}) \end{cases} \tag{3.6}$$

kde *random* je náhodné číslo z intervalu [0,1) s rovnomernou distribúciou. Tento stochastický prístup ku generovaniu bitového vektora α vyjadríme pomocou funkcie $\boldsymbol{\alpha}=R(\boldsymbol{w})$. Potom je okolie $U(\boldsymbol{w})$ zostrojené z binárnych vektorov náhodne generovaných vzhľadom k pravdepodobnostnému vektoru $\boldsymbol{w}$ určené vzťahom

$$U(\boldsymbol{w}) = \{\boldsymbol{\alpha} = R(\boldsymbol{w})\} \tag{3.7}$$

Budeme predpokladať, že kardinalita tohto okolia je konštantná, $c_0=|U(\boldsymbol{w})|$. Ak všetky zložky pravdepodobnostného vektora sú buď blízko nuly alebo jedničky, potom je "priemer" okolia $U(\boldsymbol{w})$ veľmi malý, každý element takéhoto okolia je tesne vztiahnutý k binárnemu vektoru α, ktorý je jednoznačne určený pravdepodobnostným vektorom $\boldsymbol{w}$ zaokrúhlením jeho prvkov na 0 alebo 1. V opačnom prípade, ak zložky pravdepodobnostného vektora sú všetky blízke 1/2, potom bitové vektory α zostrojené predpisom (3.6) sú rozložené v celom priestore $\{0,1\}^{\text{kn}}$ (pozri obr. 3.3).



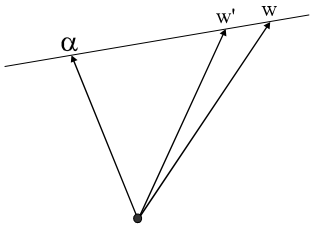
Obrázok 3.3. Schematické znázornenie okolia $U(\boldsymbol{w})$ pre daný pravdepodobnostný vektor $\boldsymbol{w}$. Za predpokladu, že všetky zložky $\boldsymbol{w}$ sú blízke 1/2, potom binárne vektory okolia $U(\boldsymbol{w})$ sú "rozmiestnené" v celom priestore binárnych vektorov $\{0,1\}^{kn}$ (diagram A). V prípade, že zložky $\boldsymbol{w}$ sú blízke buď 1 alebo 0, potom binárne vektory okolia $U(\boldsymbol{w})$ sú rozložené blízko binárneho vektora, ktorý vznikne z $\boldsymbol{w}$ zaokrúhlením jeho zložiek (diagram B).

(2) *Učenie* pravdepodobnostného vektora $\boldsymbol{w}$. Nech $B(\boldsymbol{w})$ je množina s predpísanou kardinalitou $b=|B(\boldsymbol{w})|$, ktorá obsahuje b najlepších riešení z okolia $U(\boldsymbol{w})$, formálne

$$B(\boldsymbol{w}) = \arg \min_{\boldsymbol{\alpha} \in U(\boldsymbol{w})} f\left(\Gamma(\boldsymbol{\alpha})\right) \tag{3.8}$$

Pravdepodobnostný vektor je modifikovaný - *učený* pomocou Hebbovho pravidla (známeho z teórie neurónových sietí [7])

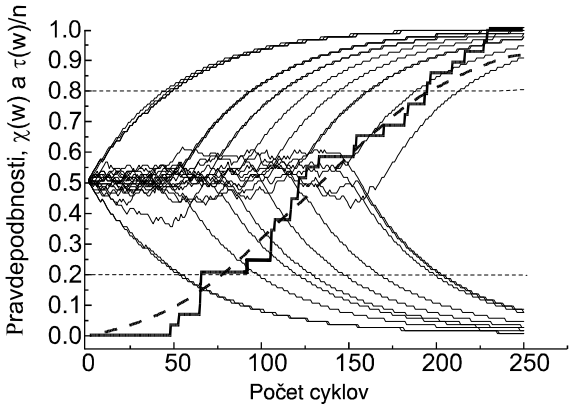
$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \lambda \sum_{\boldsymbol{\alpha} \in B(\boldsymbol{w})} (\boldsymbol{\alpha} - \boldsymbol{w}) \tag{3.9}$$



Obrázok 3.4. Geometrická interpretácia Hebbovho učiaceho pravidla (3.9). Nový pravdepodobnostný vektor $\boldsymbol{w}'=\lambda\alpha+(1-\lambda)\boldsymbol{w}$ leží bližšie k najlepšiemu riešeniu α.

kde λ je *koeficient učenia* (malé kladné číslo) a sumácia beží cez b najlepších riešení z $B(\boldsymbol{w})$. Pravidlo učenia (3.9) má veľmi jednoduchú geometrickú interpretáciu. Pre jednoduchosť predpokladajme, že množina $B(\boldsymbol{w})$ obsahuje len jeden element, potom práva strana vzorca (3.9) je konvexná kombinácia dvoch vektorov $\boldsymbol{w}$ a α, môže sa prepísať do tvaru $\boldsymbol{w}\leftarrow\lambda\alpha+(1-\lambda)\boldsymbol{w}$. To znamená, že výsledný vektor tejto konvexnej kombinácie musí ležať na úsečke, ktorá spája "body" $\boldsymbol{w}$ a α (λ je malé kladné číslo, $0<\lambda<<1$, pozri obr. 3.4). Učenie (3.9) posunie pravdepodobnostný vektor w smerom k najlepším riešeniam z množiny $B(\boldsymbol{w})$.

Oba tieto nové koncepty (pravdepodobnostný vektor a učenie) môžu byť jednoducho včlenené do štandardného horolezeckého algoritmu. V tomto prípade, namiesto náhodnej generácie vektorov okolia pomocou aplikácie mutačného operátora O_{mut} na fixný binárny vektor, teraz vektory okolia sa generujú pomocou pravdepodobnostného vektora $\boldsymbol{w}$. Okrem toho, pravdepodobnostný vektor $\boldsymbol{w}$ sa systematicky obnovuje pomocou Hebbovho učenia, ktoré ho posúva smerom k najlepším riešeniam $B(\boldsymbol{w})$ z množiny riešení $U(\boldsymbol{w})$, generovanej pomocou pravdepodobnostného vektora $\boldsymbol{w}$, $B(\boldsymbol{w})\subset U(\boldsymbol{w})$ (pozri algoritmus 3.4).



Obrázok 3.5. Priebeh jednotlivých pravdepodobností vzhľadom na počet iterácii horolezeckého algoritmu s učením v rámci modelového príkladu optimalizácie reálnej funkcie s jednou reálnou premennou reprezentovanou binárnym reťazcom s dĺžkou k=30. Prerušovaná sigmoidná čiara odpovedá veličine $\chi(\boldsymbol{w})$, schodová neprerušovaná čiara odpovedá veličine $\tau(w)/n$, kde n je počet premenných (v našom prípade n=1) . Funkcie $\chi(\boldsymbol{w})$ a $\tau(w)$ sú definované vzťahmi (3.10) resp. (3.11), pričom w_{eff}=0.2.

```
procedure HCwL(input:time_max,c_0,b,λ;output:α_fin);
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_b min_{a∈U(w)} f(Γ(a))
            w:= w + λ Σ_{α∈B(w)} (α − w);
      end;
      α_fin:= najlepšie riešenie z B(w);
end;
```

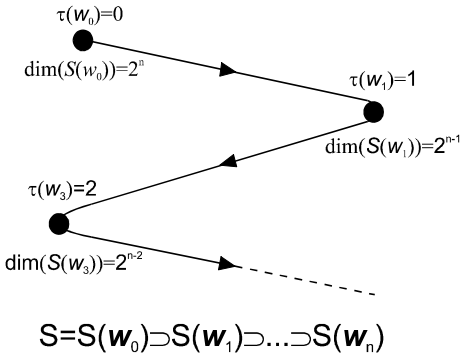
Algoritmus 3.4. Pseudopascalovská procedúra realizujúca horolezecký algoritmus s učením (HCwL, Hill Climbing with Learning). Algoritmus sa inicializuje tak, že pravdepodobnosti w_i sú rovné 1/2. Vonkajší cyklus **while** sa opakuje $time_{max}$ iterácií. V rámci tohto cyklu sa zostrojí množina $B(\boldsymbol{w})$ pre aktuálny pravdepodobnostný vektor $\boldsymbol{w}$, na základe znalosti tejto množiny sa adaptuje (učí) pravdepodobnostný vektor. Ako konečné (výstupné) riešenie sa berie najlepšie riešenie z množiny $B(\boldsymbol{w})$ po skončení iteračného procesu.

Na obr. 3.5 sú znázornené typické priebehy pravdepodobností vzhľadom na počet iterácií. Vo všeobecnosti možno konštatovať, že hodnoty každej pravdepodobnosti v počiatočnej fáze algoritmu fluktuujú okolo 1/2, potom sa monotónne približujú buď k 0 alebo k 1. Po určitom počte iterácií sú všetky pravdepodobnosti rovné buď 0 alebo 1. V tejto etape už nemá zmysel pokračovať v algoritme a ten môže byť zastavený. Ako vhodná veličina na postihnutie tejto skutočnosti sa používa *parameter usporiadania*

$$\chi(\boldsymbol{w}) = \frac{4}{n}\sum_{i=1}^{n}(w_i - 0.5)^2 \qquad (3.10)$$

Pre počiatočný vektor pravdepodobností $\boldsymbol{w}^{(0)}$=(1/2,1/2,...,1/2) je parameter usporiadania nulový. Pre pravdepodobnostné vektory so zložkami odlišnými od 1/2 sú hodnoty parametra

usporiadania kladné a menšie ako 1. Konečne, ak je pravdepodobnostný vektor rovný binárnemu vektoru (t.j. jeho komponenty sú buď 0 alebo 1), parameter usporiadania sa rovná 1. Môžeme teda povedať, že ak je v priebehu horolezeckého algoritmu parameter usporiadania väčší než určitá prahová hodnota 1-ε (kde ε je malé kladné číslo), potom metóda je ukončená, pretože výsledný binárny vektor určený ako najlepšie riešenie z množiny $B(\boldsymbol{w})$ sa už nemení (pozri obr. 3.5).



Obrázok 3.6. Schematické znázornenie trajektórie pravdepodobnostných vektorov v horolezeckom algoritme s učením. Body obratu zodpovedajú sekvencii pravdepodobnostných vektorov $\boldsymbol{w}_0$, $\boldsymbol{w}_1$, ..., $\boldsymbol{w}_n$, kde $\boldsymbol{w}_i$ je pravdepodobnostný vektor, v je ktorom *i* zložiek už fixovaných buď na 0 alebo 1. Na začiatku procesu má vektor w_0 všetky zložky približne rovné 1/2. To znamená, že odpovedajúci priestor riešení obsahuje všetkých možných 2^n binárnych vektorov dĺžky *n*, t.j. $S(w_0)=\{0,1\}^n$. Pre prechodný pravdepodobnostný vektor $\boldsymbol{w}_i$ (kde $0<i<n$), ktorý obsahuje *i* zafixovaných zložiek, dimenzia priestoru riešení je 2^{n-i}, kde exponent *n-i* je priradený počtu ešte stále neurčených zložiek pravdepodobnostného vektora. Symbol $S(\boldsymbol{w}_i)$ označuje podpriestor zložený z vektorov s troma zafixovanými komponentmi, napr. (##011####1##), kde *i* symbolov # je zamenených buď za 0 alebo za 1, platí $\dim(S(\boldsymbol{w}_i))=2^{n-i}$. Konečne, vektor $\boldsymbol{w}_n$ má všetky zložky zafixované, preto dimenzia priestoru riešení je 1 (obsahuje práve jeden binárny vektor) .

Priebeh horolezeckého algoritmu s učením a jeho interpretáciu podstatne uľahčuje tzv. *parameter nasýtenia*

$$\tau(\boldsymbol{w})=\text{počet zložiek } w_i \text{ pravdepodobnostného vektora } \boldsymbol{w}, \text{ ktoré sú menšie ako } w_{eff} \text{ alebo ak } (1-w_{eff}), \qquad (3.11)$$

kde w_{eff} je malé kladné číslo (napr. $w_{eff}=0.2$). Na začiatku algoritmu všetky zložky pravdepodobnostného vektora ležia v blízkosti 1/2, preto hodnota parametru nasýtenia je $\tau(\boldsymbol{w})=0$. V priebehu histórie algoritmu sa tento parameter skokovo zvyšuje, na záver histórie algoritmu sa parameter nasýtenia rovná počtu zložiek pravdepodobnostného vektora, $\tau(\boldsymbol{w})=n$ (pozri obr. 3.5 a 3.6). Táto podmienka sa tiež môže uvažovať ako alternatívne kritérium pre ukončenie horolezeckého algoritmu s učením.

Príklad 3.6. *Napíšte program pre horolezecký algoritmus s učením (pozri algoritmus 3.4) pre hľadanie minima funkcie n premenných, kde vektor premenných je reprezentovaný binárnym vektorom dĺžky kn. Zoptimalizujte parametre* $time_{max}$, c_0, *b a* λ *tak, aby pre danú funkciu f(**x**) bola 95% pravdepodobnosť získania globálneho minima. Pomocou programu zreprodukujete obr. 3.5, kde sú vynesené pravdepodobnosti v závislosti od počtu iteračných krokov. Experimentuje s funkciami* $\chi(w)$ *a* $\tau(w)$, *ktoré sú definované vzťahmi (3.10) resp. (3.11), pre určenie vhodného kritéria na zastavenie metódy. Vykreslite priebehy týchto funkcií od počtu iterácií.*

3.3 Metóda zakázaného hľadania (tabu search)

Metóda *zakázaného hľadania* (*tabu search*) bola navrhnutá koncom 80-tých rokov Gloverom [8,9] ako určité zovšeobecnenie horolezeckého algoritmu na riešenie zložitých optimalizačných úloh predovšetkým z operačného výskumu. Základná myšlienka tohto prístupu je neobyčajne jednoduchá. Vychádza z horolezeckého algoritmu, kde sa pre dané aktuálne riešenie generuje pomocou konečnej množiny transformácií určité okolie a funkcia sa v tomto okolí minimalizuje. Získané lokálne riešenie sa použije ako "stred " nového okolia, v ktorom sa lokálna optimalizácia opakuje; tento proces sa opakuje predpísaný počet raz. V priebehu celej histórie algoritmu sa zaznamenáva najlepšie riešenie, ktoré slúži ako výsledné optimálne riešenie. 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 tento problém odstrániť. 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. Týmto jednoduchým spôsobom možno podstatne obmedziť výskyt zacyklenia. Takto modifikovaný horolezecký algoritmus systematicky prehľadáva celú oblasť riešení, v ktorej hľadáme globálne minimum funkcie.

Glover [8,9] navrhol ďalšie metódy intenzifikácie a diverzifikácie metódy zakázaného hľadania, predovšetkým prístup tzv. dlhodobej pamäti, v ktorom sa "pokutujú" (znevýhodňujú) tie transformácie, ktoré síce nepatria do krátkodobej pamäti, ale často sa vyskytovali v predchádzajúcej histórii algoritmu. Glover a Laguna v knihe [10] naznačuje teoretické základy metódy. Avšak ich argumenty sa prezentujú vo veľmi vágnej a všeobecnej rovine. Možno konštatovať, že táto metóda v súčasnosti ešte nemá solídne teoretické základy, ktoré by dávali odpoveď na dôležité otázky, za akých podmienok môže poskytovať riešenie, ktoré je totožné s globálnym alebo mu je veľmi blízke. Preto je snáď lepšie hovoriť o heuristike zakázaného hľadania, a nie o metóde zakázaného hľadania. Ide totiž viac o súbor algoritmických trikov a heuristík než o metódu so solídnym teoretickým základom. Súčasne však musíme konštatovať, že patrí medzi numericky veľmi efektívne metódy riešenia globálnej optimalizácie nad diskrétnymi oblasťami [11], výsledky poskytuje za zlomok času potrebného pri použití buď presných metód (typu napr. spätného prehľadávania - backtracking [12]) alebo stochastických optimalizačných metód. Práve v tomto nesúlade medzi neexistujúcim teoretickým základom a vysokou numerickou efektívnosťou vidíme "krásu" tejto metódy, v jej neobyčajnej flexibilnosti pri modifikovaní pre daný problém.

Definujme si množinu prípustných transformácií

$$S = \{t_1, t_2, ..., t_p\} \tag{3.12}$$

Transformácia $t \in S$ zobrazuje binárny vektor $\alpha \in \{0,1\}^{kn}$ na iný binárny vektor $\alpha' \in \{0,1\}^{kn}$

$$t : \{0,1\}^{kn} \rightarrow \{0,1\}^{kn} \tag{3.13a}$$

pre $\forall t \in S$. Jednoduchá realizácia týchto transformácií je

$$t_i(...\alpha_i...) = (...1-\alpha_i...) \tag{3.13b}$$

pre i=1,2,...,$p=kn$. Operátor t_i zmení v i-tej polohe binárnu hodnotu na jej komplement. Vo všeobecnosti sú transformácie z S ohraničené nasledujúcimi podmienkami:

(1) Nech $t_1, t_2 \in$ S sú dve rôzne transformácie, $t_1 \neq t_2$, potom pre $\forall \alpha \in \{0,1\}^{kn}$ platí $t_1\alpha \neq t_2\alpha$.

(2) Pre každú transformáciu $t \in S$ existuje taká transformácia $t^{-1} \in S$, ktorá je inverzná k t, $tt^{-1}\alpha = t^{-1}t\alpha = \alpha$, pre $\forall \alpha \in \{0,1\}^{kn}$.

(3) Pre každú dvojicu $\alpha_1,\alpha_2 \in \{0,1\}^{kn}$ rôznych binárnych vektorov, $\alpha_1 \neq \alpha_2$, existuje taká postupnosť transformácií $t_{i_1}, t_{i_2}, ..., t_{i_n} \in S$, že "východiskový" vektor α_1 sa postupne pretransformuje na "konečný" vektor α_2

$$\alpha_1 = \beta'_1 \xrightarrow{t_{i_1}} \beta'_2 \xrightarrow{t_{i_2}} \ldots \xrightarrow{t_{i_n}} \alpha_2 = \beta'_n \tag{3.14}$$

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

$$U(\alpha) = \{t\alpha;\ \forall t \in S\} \tag{3.15}$$

Pôvodný horolezecký algoritmus sa bude teraz modifikovať tak, že namiesto okolia (3.3) generovaného náhodne pomocou stochastického operátora mutácie použijeme deterministicky definované okolie (3.15) generované pomocou prípustných transformácií z množiny *S*. Hlavné obmedzenie tejto jednoduchej modifikácie horolezeckého algoritmu je, že po určitom počte iteračných krokov sa výsledné riešenia začnú cyklicky opakovať. Po konečnom počte krokov sa tento algoritmus vráti k riešeniu, ktoré sa už vyskytovalo ako lokálne riešenie v predchádzajúcom iteračnom kroku, pričom najlepšie zaznamenané riešenie je obvykle vzdialené od globálneho riešenia.

Metóda zakázaného hľadania [8,9] využíva jednoduchú heuristiku ako pokračovať v hľadaní globálneho minima bez možnosti návratu do lokálneho minima, ktoré sa už zaznamenalo v predchádzajúcej histórii algoritmu. Hlavná myšlienka tejto heuristiky je tzv. *zakázaný zoznam T* (tabu list) majúci vlastnosť krátkodobej pamäti, ktorý dočasne obsahuje inverzné transformácie k použitým transformáciám v predchádzajúcich iteráciach. Zakázaný zoznam transformácií $T \subseteq S$, maximálnej kardinality *s*, $0 \leq |T| \leq s$, zostrojuje a systematicky obnovuje v priebehu celého algoritmu. Ak transformácia *t* patrí pre danú iteráciu do zakázaného zoznamu, $t \in T$, potom sa nemôže používať v lokálnej minimalizácii v rámci okolia aktuálneho riešenia α. Pri inicializácii algoritmu je zakázaný zoznam prázdny, po každej iterácii sa do zakázaného zoznamu dodá transformácia, ktorá poskytla lokálne optimálne riešenie zostrojené z riešenia z predchádzajúcej iterácie. Po *s* iteráciach obsahuje zakázaný zoznam už *s* transformácií, zo zakázaného zoznamu sa vylúči transformácia, ktorá sa tam dodala pred *s* iteráciami. To znamená, že po naplnení zakázaného zoznamu (t.j. po *s* iteráciach) je každé dodanie novej transformácie doprevádzané aj vylúčením "najstaršej" transformácie (dodanej práve pred *s* iteráciami); hovoríme, že zakázaný zoznam sa cyklicky obnovuje

$$T := \begin{cases} T \cup \{t^{*-1}\} & (\text{pre } (T(< s) \\ \\ (T \cup \{t^{*-1}\}) \setminus \hat{t}\,(\text{pre } (T(= s) \end{cases} \tag{3.16}$$

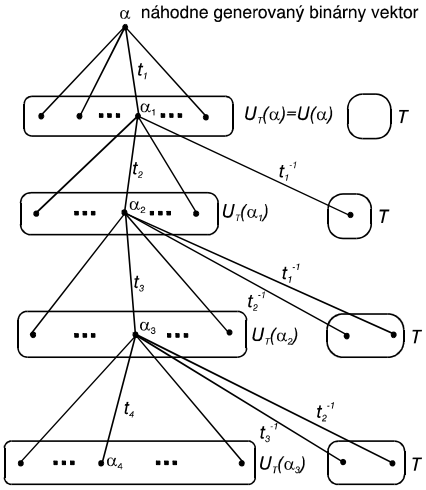
kde t^* je transformácia, ktorá vytvára lokálne minimálne riešenie, $\alpha^* = t^* \alpha$, a $\hat{t}$ je "najstaršia" transformácia zavedená do zakázaného zoznamu práve pred *s* iteráciami.

Numerické skúsenosti s algoritmom zakázaného hľadania ukazujú, že veľkosť *s* zakázaného zoznamu je veľmi dôležitým parametrom na prehľadávanie oblasti $\{0,1\}^{kn}$ s možnosťou vymaniť sa z lokálnych miním. Ak je parameter *s* malý, potom sa môžu vyskytnúť zacyklenie algoritmu podobne ako pri klasickom horolezeckom algoritme s okolím zostrojeným podľa (3.3). Zacyklenie sa síce neopakuje v susedných dvoch krokoch, no riešenie sa môže opakovať po viacerých krokoch. V prípade, že je parameter *s* veľký, potom pri prehľadávaní oblasti $\{0,1\}^{kn}$ s veľkou pravdepodobnosťou "preskočíme " hlboké údolia minimalizovanej funkcie $f(\alpha)$, t.j. vynecháme nádejné lokálne minima, ktoré môžu byť globálnymi minimami.

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

$$U_T(\alpha) = \{\alpha'; \forall t \in S \setminus T : \alpha' = t\alpha\} \tag{3.17}$$

ktorého kardinalita $p\text{-}s \leq |U_T(\alpha)| \leq p$, pričom $U_T(\alpha)=U(\alpha)$, pre $T=\emptyset$. Toto okolie obsahuje vektory $\alpha' \in \{0,1\}^{kn}$, ktoré sú vytvorené použitím transformácií z množiny S, ktoré nepatria do zakázaného zoznamu T. Lokálna minimalizácia sa vykonáva v modifikovanom okolí $U_T(\alpha)$ s výnimkou tzv. *aspiračného kritéria*. Toto kritérium porušuje obmedzenie zakázaného zoznamu vtedy, ak existuje taká transformácia $t \in S$, že vektor $\alpha'=t\alpha$ poskytuje nižšiu funkčnú hodnotu, ako dočasne najlepšie riešenie. Pascalovský pseudokód metódy zakázaného hľadania je uvedený v algoritme 3.5. a jeho diagramatická vizualizácia je znázornená na obr. 3.7.



Obrázok 3.7. Diagramatická vizualizácia metódy zakázaného hľadania pre veľkosť zakázaného zoznamu s=2. Metóda sa inicializuje náhodne generovaným vektorom α. Pomocou transformácií t z množiny prípustných transformácií S sa zostrojí prvé okolie $U(\alpha)$. Najlepšie riešenie z tohto okolia je označené α_1, v nasledujúcom iteračnom kroku sa toto riešenie použije ako "stred" pre konštrukciu nového okolia. Inverzná transformácia t_1^{-1} sa zavedie do zakázaného zoznamu T, ktorý bol pôvodne prázdny. Menšie bloky v pravom stĺpci odpovedajú zakázaným zoznamom pre jednotlivé iteračné kroky.

Ako už bolo povedané, algoritmus zakázaného hľadania je veľmi podobný horolezeckému algoritmu. Pri zakázanom hľadaní nie je okolie riešenia zostrojené stochastickým spôsobom ako v horeuvedenej verzii horolezeckého algoritmu pomocou operátora mutácie O_{mut}, ale systematickým deterministickým spôsobom pomocou prípustných transformácií z množiny S. Takto realizovaný horolezecký algoritmus má jedno vážne ohraničenie, a to cyklický výskyt riešení po určitom počte iteračných krokov. Základná myšlienka algoritmu zakázaného hľadanie na odstránenie tohto problému zacyklenia spočíva v zavedení tzv. zakázaného zoznamu, ktorý obsahuje určitý počet inverzných transformácií k tým transformáciám, ktoré boli použité v predchádzajúcej krátkej histórii algoritmu. Tento zoznam sa cyklicky obnovuje tak, že pri zavedení inverznej transformácie z aktuálneho iteračného kroku sa z neho odstráni aj "najstaršia" inverzná transformácia. Tento jednoduchý algoritmický trik odstraňuje spoľahlivo problém zacyklenia horolezeckého algoritmu s deterministickým generovaním okolia pomocou množiny prípustných transformácií.

```
procedure Tabu_Search(input:time_max,s ; output: f_fin,α_fin);
begin α:=náhodne generovaný binárny vektor dĺžky kn;
      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;
```

Algoritmus 3.5. Pseudopascalovská procedúra pre metódu zakázaného hľadania. Vstupnými parametrami sú $time_{max}$ a *s*, ktoré určujú maximálny počet iteračných krokov resp. veľkosť zakázaného zoznamu *T*. Procedúra obsahuje dva cykly: vonkajší cyklus **while** realizuje iteračné kroky zakázaného hľadania, zatiaľ čo vnútorný for-cyklus slúži pre konštrukciu okolia U(α). Poznamenajme, že toto okolie nie je explicitne zostrojené, generujú sa len jeho elementy a tie sa hneď testujú, či ich funkčná hodnota nie je menšia ako lokálne najlepšia funkčná hodnota $f_{fin\text{-}loc}$. Vonkajší cyklus sa ukončí operáciou obnovy zakázaného zoznamu.

Možnosti ďalšej sofistikácie metódy zakázaného hľadania sa široko diskutujú v literatúre [8-11]. Prístup založený na koncepcii dlhodobej pamäti patrí medzi základné prostriedky intenzifikácie a diverzifikácie metódy zakázaného hľadania. Využíva možnosť odmietnutia (pomocou penalizácie) transformácií, ktoré sa v predchádzajúcej histórii algoritmu vyskytujú najčastejšie (tzv. dlhodobá pamäť). Hľadanie lokálneho minima v modifikovanom okolí $U_T(\alpha)$ je v tomto prístupe založené nielen na zmenách funkcie $f(\alpha)$, ale aj na predchádzajúcej histórii algoritmu. Jednoduchá realizácia tejto všeobecnej idey spočíva v použití frekvencií transformácií $\omega(t)$. Pri inicializácii algoritmu sú tieto frekvencie nulové, v každom iteračnom kroku s výslednou transformáciou t^* sa potom odpovedajúca frekvencia zvýši o jednotku, $\omega(t^*) \leftarrow \omega(t^*)+1$. Po predpísanom počte krokov (obvykle rádovo väčšom ako veľkosť zakázaného zoznamu *T*) tieto frekvencie určujú, ako často boli jednotlivé transformácie z *S* použité v lokálnej minimalizácii. Frekvencie sa používajú ako penalizačné funkcie pri hľadaní minima v okolí $U_T(\alpha)$. Vektor $\alpha'=t\alpha \in U_T(\alpha)$, kde $t \in S \setminus T$, sa akceptuje ako dočasne najlepšie riešenie, ak je splnená nasledujúca podmienka

$$f(\alpha') + \chi\omega(t) < f(\alpha^*) \tag{3.18}$$

kde χ je empiricky určená malá kladná penalizačná konštanta. Najčastejšie používané transformácie sú penalizované v dôsledku vysokých hodnôt frekvencií. Prístup dlhodobej pamäti dáva príležitosť aj iným transformáciám než tým, ktoré aj keď poskytujú lokálne nižšiu funkčnú hodnotu $f(\alpha')$, sú v dôsledku ich frekventovaného výskytu v predchádzajúcej dlhodobej histórii algoritmu znevýhodnené - penalizované.

Príklad 3.7. *Napíšte program pre metódu zakázaného hľadania. Program hľadá minima n-rozmernej funkcie (napr. špecifikovanej v príklade 3.2), pričom kódovanie binárnej reprezentácie je štandardné a tiež aj podľa Graya. Transformačné operátory t z množiny S sú určené vzťahmi (3.13a-b).*

3.4 Evolučné programovanie

Evolučné programovanie [13-15] patrí medzi tie stochastické optimalizačné algoritmy, ktoré možno chápať ako jednoduché zovšeobecnenie horolezeckého algoritmu. V tejto kapitole uvedieme zjednodušenú verziu evolučného programovania, ktorá pokrýva vlastnosti tejto metódy dostatočne všeobecne.

Základná úloha spočíva v riešení nasledujúceho optimalizačného problému

$$\boldsymbol{\alpha}_{opt} = \arg \min_{\boldsymbol{\alpha} \in \{0,1\}^k} f(\boldsymbol{\alpha}) \tag{3.19}$$

kde $f:\{0,1\}^k \to R$ je funkcia, ktorá zobrazuje binárne vektory dĺžky k na reálne čísla. Nech P je množina - populácia riešení tvaru

$$P = \{\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, ..., \boldsymbol{\alpha}_p\} \subseteq \{0,1\}^k \tag{3.20}$$

Každý prvok $\boldsymbol{\alpha}$ z populácie P je ohodnotený funkčnou hodnotou $f(\boldsymbol{\alpha})$. Z populácie P vyberieme podmnožninu - podpopuláciu rodičov $Q \subseteq P$, ktorej kardinalita je menšia alebo nanajvýš rovná kardinalite pôvodnej populácie P , $|Q| \leq |P|$. Riešenia z podpopulácie Q sa transformujú mutačným operátorom O_{mut} (pozri (3.2a-c)) na podpopuláciu potomkov

$$Q' = \{\boldsymbol{\alpha}' = O_{mut}(\boldsymbol{\alpha}) ; \boldsymbol{\alpha} \in Q\} \tag{3.21}$$

pričom kardinality Q a Q' sú rovnaké, $|Q|=|Q'|$. Potomkovia z podpopulácie Q' sa ohodnotia funkčnými hodnotami $f(\boldsymbol{\alpha})$. Podpopulácie Q a Q' sú zjednotené do spoločnej množiny obsahujúcej všetkých rodičov a ich potomkov

$$R = Q \cup Q' \tag{3.22}$$

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

Uvedieme niekoľko poznámok k realizácii tohto operátora. Najjednoduchší prístup spočíva v usporiadaní prvkov R podľa rastúcich funkčných hodnôt $f(\boldsymbol{\alpha})$, potom S je tvorená prvými $|Q|$ elementmi takto usporiadanej množiny R. Iný prístup je ten, že pre $|Q|$ náhodne vybraných dvojíc $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \in R$ sa realizuje malý "turnaj"; ak platí $f(\boldsymbol{\alpha}_1) < f(\boldsymbol{\alpha}_2)$, potom riešenie $\boldsymbol{\alpha}_1$ sa presunie do množiny S (pozri algoritmus 3.6).

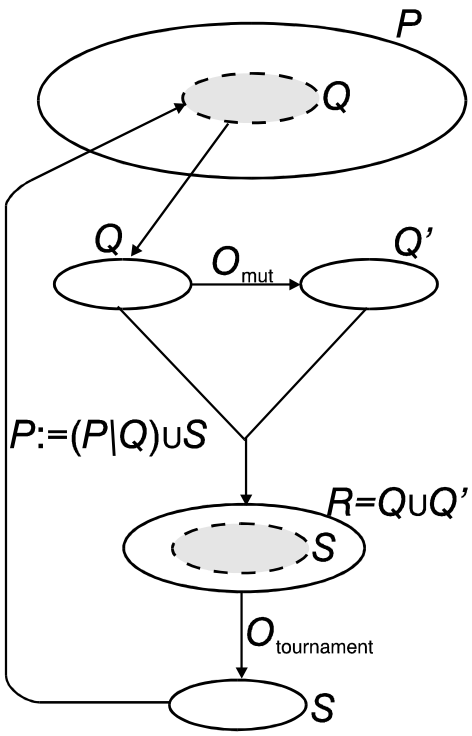
$$S = O_{tournament}(R) \tag{3.23}$$

Pre známu množinu S nasledovníkov obnovíme populáciu P takto

$$P \leftarrow (P \setminus Q) \cup S \tag{3.24}$$

to znamená, že v populácii P je množina rodičov Q nahradená množinou nasledovníkov S. Týmto je metóda evolučného programovania plne určená (pozri algoritmus 3.7 a obr. 3.8).

Príklad 3.8. *Napíšte program na evolučné programovanie (pozri algoritmy 3.6 a 3.7). Program je určený na optimalizáciu reálnej funkcie n premenných, pričom premenné sú reprezentované binárnym reťazcom dĺžky k. Porovnajte efektívnosť programu s podobnými programami pre horolezecký algoritmus a metódu zakázaného hľadania.*



Obrázok 3.8. Schematické znázornenie evolučného programovania. Z populácie *P* sa náhodne vyberie podpopulácia rodičov *Q*, ktorá je upravená pomocou operátora mutácie na podpopulácie potomkov *Q*'. Z týchto dvoch podpopulácií sa vytvorí zjednotená podpopulácia *R*. Aplikovaním turnaja na túto podpopuláciu *R* dostaneme podpopuláciu nasledovníkov *S*. Podpopulácia nasledovníkov sa vracia do pôvodnej populácie *P* tak, že sa pôvodná rodičovská podpopulácia *Q* odstráni.

```
procedure Tournament(input R; output S);
begin S:=∅;
      while |R|>0 do
      begin náhodne vyber z R dve riešenia α1 a α2;
            R:=R\{α1 , α2};
            if f(α1)<f(α2)then S:=S∪{α1} else
                               S:=S∪{α2};
      end;
end;
```

Algoritmus 3.6. Implementácia turnaja pre výber nasledovníkov zo zjednotenej podpopulácie rodičov a potomkov. Tento jednoduchý "párový turnaj" môže sa ľahko zovšeobecniť tak, že predpísaný počet ráz náhodne vyberáme dvojice riešení a usporiadame medzi nimi "párový" turnaj, víťazovi zvýšime o jednotku skóre víťazstva. Potom množina nasledovníkov S sa vytvorí z prvých |*Q*| riešení, ktoré majú najvyššie skóre.

```
procedure Evolutionary_Programmning(output: α_opt);
begin P:= náhodne generovaná populácia chromozómov;
      stop_criterion:=false;
      while not stop_criterion do
      begin Q:= náhodne vybraná podpopulácia z P;
            Q':=O_mut(Q); R:=Q∪Q'; S:=O_tournament(R); P:=(P\Q)∪S;
            if konvergenčné kritériá sú splnené then
            stop_criterion:=true;
      end;
      α_opt = arg min_{α∈P} f(α);
end;
```

Algoritmus 3.7. Implementácia evolučného programovania. Metóda sa inicializuje náhodným generovaním populácie P. Cyklus **while** sa opakuje tak dlho, pokiaľ neplatia podmienky konvergencie (napr. populácia je dostatočne homogénna). Výsledné riešenie α_{opt} sa určí ako najlepšie riešenie z výslednej populácie *P*.

Literatúra

[1] B. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, and S. Watt: *Maple V*. Springer Verlag, Berlin 1992.

[2] S. Wolfram: *Mathematica. A System for Doing Mathematics by Computers*. Addison Wesley, Reading, MA 1993.

[3] S. Baluja: Population-Based Incremental Learning. A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. *Technical Report* CMU-CS-94-163. School of Comp. Sci., Carnegie Mellon University, Pittsburgh 1994.

[4] S. Baluja and R. Caruana: Removing the Genetics from the Standard Genetic Algorithm. *Technical Report* CMU-CS-95-141. School of Computer Science, Carnegie Mellon University, Pittsburgh 1995 (preprinty referencií 3 a 4 sú dostupné na anonymnom ftp serveri reports.adm.cs.cmu.edu/usr/anon).

[5] V. Kvasnička, J. Pospíchal, and M. Pelikán: Hill Climbing with Learning (An Abstraction of Genetic Algorithm). *Proceedings of Mendel'95, First International Conference on Genetic Algorithms*, Brno, September 26-28, 1995, pp.65-70.

[6] V. Kvasnička, J. Pospíchal, and M. Pelikán: Hill climbing with learning (an abstraction of genetic algorithm). *Neural Network World,* **6** (1996) 773.

[7] V. Kvasnička, Ľ. Beňušková, J. Pospíchal, I. Farkaš, P. Tiňo, A. Kráľ*: Úvod do teórie neurónových sietí*. IRIS, Bratislava 1997.

[8] F. Glover:Tabu Search - Part I. *ORSA Journal of Operations Research*, **1** (1989) 190.

[9] F. Glover:Tabu Search - Part II. *ORSA Journal of Operations Research*, **2** (1990) 4.

[10] F. Glover and M. Laguna: *Tabu Search*. Kluwer Academic Publishers, Dordrecht, The Netherlands 1997.

[11] D. Karaboga, D.-T. Pham: *Intelligent Optimization Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*. Springer Verlag, Berlin 2000.

[12] L. Kučera: *Kombinatorické algoritmy*. SNTL, Praha 1983.

[13] D.B. Fogel: *Evolutionary Computation. Toward a new Philosophy of Machine Intelligence*. IEEE Press, New York 1995.

[14] L.J. Fogel: *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming*. Wiley, New York 1999.

[15] Z. Michalewicz, D.B. Fogel: *How to Solve It: Modern Heuristics*. Springer Verlag, Belin 1999.