

Evolučné algoritmy

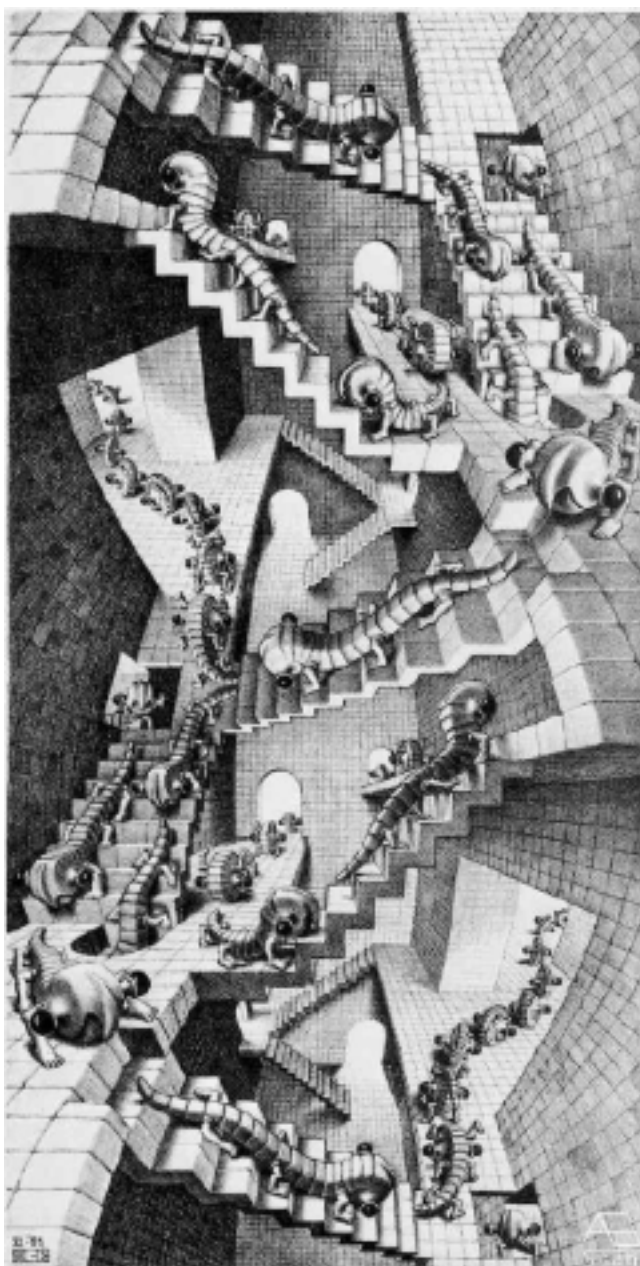
Vladimír Kvasnička

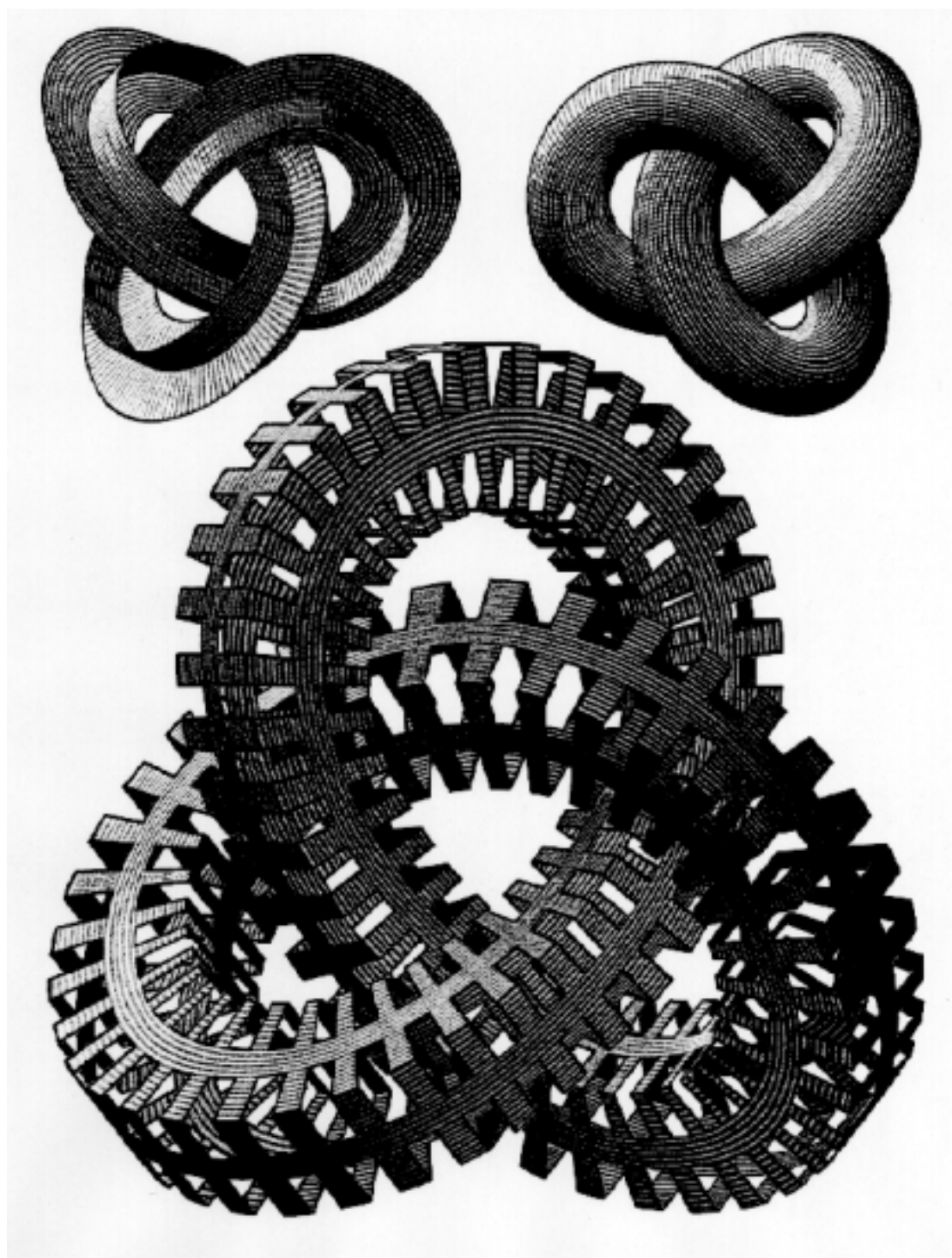
Jiří Pospíchal

Peter Tiňo



STU
Bratislava





Zoznam význačnejších symbolov a označení

Grécka abeceda

α	binárny vektor, tiež aj chromozóm, reťazec Readovho kódu, parameter pre znižovanie teploty u simulovaného žihania, rotačný uhol, parameter strmosti sigmoidy, sekvencia DNA
$\tilde{\alpha}$	binárny vektor odpovedajúci číslu zakódovanému Grayovým kódom
β	chromozóm, tiež aj parameter mutácie u evolučnej stratégie, strmost' v (10.28), sekvencia DNA
γ	konštanta váhy porušenia podmienky v (10.27)
δ	malé kladné číslo
$\delta(y, y_{opt})$	Kroneckerova funkcia
ε	malé kladné číslo
ζ	binárna hodnota v "messy" chromozómu
η	rýchlosť učenia
ϑ	prahový koeficient neurónu
θ	externý vstup do Hopfieldovej siete
κ	"messy" chromozóm
λ	koeficient učenia (malé kladné číslo), tiež aj Langtonov parameter na opis dynamických režimov celulárnych automatov
μ	index hodnoty v "messy" chromozómu, tiež aj stredná hodnota premennej, pokutová konštanta pri magnetizácii.
ν	binárna hodnota v "messy" chromozómu
ξ	index hodnoty v "messy" chromozómu, tiež aj potenciál neurónu, dĺžka intervalu
π	n -tica prirodzených čísel, tiež aj zobrazenie množiny na množinu, Ludolfovo číslo
ρ	okraj intervalu
σ	schéma, tiež aj štandardná odchýlka
$\sigma^2(T)$	disperzia funkcie pri teplote T
τ	mutačný bod, tiež aj parameter mutácie u evolučnej stratégie, časová konštanta diferenciálnej rovnice v (10.23)
$\tau(w)$	parameter nasýtenia
ϕ	zobrazenie vrcholov v grafoch na vrcholy alebo symboly
φ	koeficient úspešnosti
$\chi(w)$	parameter usporiadania (3.10), (4.18)
χ	penalizačná konštanta (3.18), "messy" chromozóm (4.38)
$\chi_T(y)$	počet výskytov stavu x , ktorý odpovedá funkčnej hodnote y a ktoré sa vyskytujú v Metropolisovom algoritme pre danú teplotu T
ω	frekvencia výskytu na intervalu, tiež frekvencia transformácií
$\omega_c(x)$	funkcia hustoty rozdelenia náhodnej premennej
Γ	zobrazenie množiny binárnych vektorov dĺžky kn na body - n -tica reálnych čísel
Δ	rozdiel, tiež aj dĺžka schémy
Θ	prahový koeficient skrytého neurónu
Ξ	abeceda nad DNA bázami
Ξ^+	množina neprázdnych konečných slov nad abecedou Ξ
Π	produkt (násobenie)
Σ	suma, súčet

Množinové symboly

$\{a, b, \dots\}$	prvky množiny
$\{0, 1\}^k$	množina reťazcov (binárnych vektorov) dĺžky k zložená zo symbolov 0 a 1
$ X $	kardinalita (počet prvkov) množiny X
$\in$	je prvkom množiny
$A \subset B$	A je vlastnou podmnožinou B
$A \subseteq B$	A je podmnožinou B (teda A môže byť aj totožné s B)
$\dim(X)$	kardinalita množiny alebo veľkosť priestoru X
$A \setminus B$	rozdiel dvoch množín
$A \cup B$	zjednotenie dvoch množín
$A \cap B$	prienik dvoch množín
$\emptyset$	prázdna množina

Množiny

D	n -rozmerná kocka
R	priestor reálnych čísel
R_+	priestor nezáporných reálnych čísel
U	pomocná množina
$U(\alpha)$	okolie binárneho vektora α zostrojené z binárnych vektorov
$U(w)$	okolie zostrojené z binárnych vektorov náhodne generovaných vzhľadom k pravdepodobnostnému vektoru w
$\mathcal{T}_i$	množina obsahujúca všetky možné koreňové stromy, ktoré sú zložené z i vrcholov, tiež aj množina syntaktických stromov

Symboly binárnych operácií

$\times$	karteziánsky produkt intervalov, tiež aj násobenie
$\oplus$	zreťazenie podvektorov, tiež aj "xor" súčet binárnych čísel
$*$	voľný symbol, tiež aj násobenie
$\vee$	logický súčin (konjunkcia)
$\bullet$	oddeľovač kódovaný špeciálnou postupnosťou báz v DNA, oddeľujúci sekvencie kódujúce páskové symboly

Symboly porovnania

$\propto$	je priamo úmerné
$\approx$	je približne, je izomorfný
$\gg$	je omnoho väčšie
$>$	špeciálna postupnosť DNA rozpoznávaná reštrikčným enzýmom
$\neq$	nerovná sa

Šípky

$\rightarrow$	zobrazenie z priestoru do priestoru, tiež aj hodnota sa blíži číslu, zámena čísla za číslo
$x \leftarrow y$	priradenie hodnoty y premennej x , <i>perturbed</i> $\leftarrow$ <i>current</i> prechod zo stavu <i>current</i> na stav <i>perturbed</i>
$\Leftrightarrow$	je ekvivalentné
$\Rightarrow$	z toho vyplýva
$\uparrow x$	molekula DNA, v ktorej za sebou nasledujúce bázy v orientácii 5'-3' (pozri obr. 11.1) zodpovedajú symbolom slova x
$\downarrow x$	molekula DNA komplementárna k molekule $\uparrow x$
$\updownarrow x$	molekuly $\uparrow x$ a $\downarrow x$ spojené väzbami medzi komplementárnymi bázami, dvojzávit

Rôzne symboly

$\forall$	univerzálny kvantifikátor, "pre každé"
∞	nekonečno
#	symbol voľne zameniteľný za 0 alebo 1
$\frac{\partial X}{\partial T}$	parciálna derivácia X podľa T

Zátvorky

$[a,b]$	uzavretý interval od a po b
$[a,b)$	zľava uzavretý, sprava otvorený interval od a po b
$\lceil x \rceil$	celá časť reálneho čísla x (zaokrúhlenie dole na celé číslo)
$\lfloor x \rfloor$	zaokrúhlenie reálneho čísla na najbližšie celé číslo (aj hore)
$ x $	absolútna hodnota premennej x
$ X $	kardinalita (počet prvkov) množiny X
$\langle x \rangle_T$	stredná hodnota veličiny x pri teplote T

Funkcie a operátory

$\arg \min_{x \in D} f(x)$	globálne minimum funkcie f premennej alebo vektora premenných x z oblasti D
$\text{code}(T)$	lineárny kód koreňového stromu T
$\exp(x)$	e^x
f	funkcia
$\text{int}(\alpha)$	celé číslo odpovedajúce binárnemu číslu α
$\ln$	prirodzený logaritmus (pri základe e)
$\max(a,b)$	maximum z dvoch čísel a a b
$\min(a,b)$	minimum z dvoch čísel a a b
$\text{mod } 2$	modulo 2, zvyšok po delení dvojkou
$N!$	faktoriál N
O_{cross}	operátor kríženia
O_{inv}	operátor inverzie
O_{mut}	operátor mutácie
O_{repro}	operátor reprodukcie
$P(A B)$	pravdepodobnosť, že nastane udalosť A pri splnení podmienky B
random	náhodné číslo s rovnomernou distribúciou z intervalu $[0,1)$.
$\text{random}(x)$	funkcia generujúca náhodné celé číslo z uzavretého intervalu $[0,x-1]$.
$\text{real}(\alpha)$	reálne číslo odpovedajúce binárnemu číslu α
$\tanh(x)$	hyperbolický tangens

Akcenty, horné a dolné indexy

$\hat{t}$	"najstaršia" transformácia zavedená do zakázaného zoznamu pred s iteráciami.
t^{-1}	inverzná transformácia k transformácii t
t^*	transformácia, ktorá vytvára lokálne minimálne riešenie
$p, \wp$	dĺžka Readovho kódu
$\langle x \rangle_T$	stredná hodnota veličiny x pri teplote T
t_{CPU}	čas centrálnej jednotky procesora potrebný na riešenie úlohy
P_{mut}	pravdepodobnosť mutácie
P_{cross}	pravdepodobnosť kríženia

Vladimír Kvasnička - Jiří Pospíchal - Peter Tiňo

Evolučné algoritmy

**Publikácia vyšla vďaka
finančnej podpore
Vzdelávacej nadácie Jana Husa a
Open Society Foundation - Program HESP**

Napísanie knihy bolo podporené grantom KEGA 50/97-162-ÚP

Evolučné algoritmy

Vladimír Kvasnička

Jiří Pospíchal

Peter Tiňo

2000

Kniha je venovaná evolučným algoritmom (genetický algoritmus, genetické programovanie, simulované žihanie, evolučné stratégie a evolučné programovanie, tabu search). Tieto nové prístupy sú v súčasnosti považované za efektívny nástroj simulácie evolúcie a emergencie kolektívnych vlastností multiagentových systémov a sú tiež aplikovateľné ako optimalizačný algoritmus pre riešenie vedecko-technických problémov. Okrem priamych aplikácií sú uvedené prístupy základom modelovania vývoja biologických, sociálnych a kultúrnych javov v oblasti umelého života. Na záver publikácie sú uvedené výpočty na báze DNA.

Kniha je určená poslucháčom vysokých škôl, doktorandom a všetkým tým, ktorí sa zaujímajú o moderné metódy umelej inteligencie založené na metafore Darwinovej evolučnej teórie.

© Prof. Ing. Vladimír Kvasnička, DrSc., Doc. RNDr. Jiří Pospíchal, CSc.,
Ing. Peter Tiňo, CSc.

Recenzenti: Doc. RNDr. Gabriela Andrejková, CSc.
Doc. Ing. Martin Breza, CSc.
Doc. Ing. Július Csontó, CSc.

Obálka: M.C. Escher's "House of Stairs" and "Knots" © 2000 Cordon Art B.V.
-Baarn - Holland. All rights reserved.

Vydala Slovenská Technická Univerzita v Bratislave vo Vydavateľstve STU,
Bratislava, Vazovova 5

Schválilo vedenie Chemickotechnologickej fakulty STU v Bratislave dňa
10.5.2000

ISBN XXXXXXXXXX

Obsah

Predslov	1
1. Úvodné poznámky (Vladimír Kvasnička)	2
Literatúra	4
2. Základné pojmy evolučných algoritmov (Vladimír Kvasnička)	6
2.1 Optimalizačný problém	6
2.2 Binárna reprezentácia reálnej premennej	9
2.3 Transformácia spojitého optimalizačného problému na binárny optimalizačný problém	11
Literatúra	13
3. Horolezecké algoritmy (Vladimír Kvasnička)	14
3.1 Základné stochastické optimalizačné algoritmy: slepý algoritmus a horolezecký algoritmus	14
3.2 Horolezecký algoritmus s učením	20
3.3 Metóda zakázaného hľadania (tabu search)	24
3.4 Evolučné programovanie	28
Literatúra	30
4. Genetický algoritmus (GA) (Vladimír Kvasnička)	31
4.1 Evolúcia v živej prírode	31
4.2 Základné pojmy genetického algoritmu - populácia a fitness chromozómov	35
4.3 Kríženie chromozómov	38
4.4 Implementácia genetického algoritmu	43
4.5 Teória genetického algoritmu	45
4.6 Hammingova bariéra	48
4.7 "Messy"-chromozómy (M-chromozómy)	51
4.8 Paralelný genetický algoritmus	55
Literatúra	57
5. Genetické programovanie (GP) (Vladimír Kvasnička)	58
5.1 Úvodné poznámky	58
5.2 Koreňové stromy a Readov lineárny kód	59
5.3 Symbolická regresia	67

5.4	Ilustratívne príklady symbolickej regresie - parita a symetria binárnych vektorov	72
	Literatúra	76
6.	Simulované žihanie (Vladimír Kvasnička)	77
6.1	Úvodné poznámky	77
6.2	Vzťah k štatistickej fyzike	80
6.3	Modelové výpočty	83
6.3.1	Simulované žihanie s elitizmom	87
6.3.2	Paralelné simulované žihanie	88
	Literatúra	90
7.	Kombinatorické optimalizačné problémy (Vladimír Kvasnička)	92
7.1	Formulácia základných kombinatorických optimalizačných problémov	92
	Literatúra	99
8.	Evolučné stratégie (Jiří Pospíchal)	100
8.1	Historický úvod	100
8.2	Stratégia (1+1) – jeden rodič a jeden potomok	102
8.3	Distribúcia pravdepodobnosti a generovanie náhodných čísel	104
8.4	Mnohočlenné stratégie a kovariancie	107
8.5	Ukážka aplikácie algoritmu na Rosenbrockovu funkciu	114
8.6	Evolučné programovanie	117
8.7	Záver	118
	Literatúra	119
9.	Umelý život (Jiří Pospíchal)	121
9.1	Čo je to umelý život?	121
9.2	Definície života	122
9.3	Umelá inteligencia a agenti	123
9.4	Evolúcia pravidiel a Lindenmayerove systémy	126
9.5	Evolučná dynamika a modelovanie krídla	128
9.6	Celulárne automaty	134
9.7	Vznik komplexných javov	137
9.8	Kooperácia alebo súťaženie?	140
9.9	Vznik života evolúciou neživého	143
	Literatúra	150

10. Neurónové siete (Jiří Pospíchal)	155
10.1 Historický úvod	155
10.2 Viacvrstvé neurónové siete s dopredným šírením	156
10.2.1 Všeobecný klasifikačný problém	156
10.2.2 Definícia neurónovej siete	158
10.2.3 Algoritmizácia neurónovej siete s dopredným šírením	161
10.2.4 Optimalizácia neurónových sietí s dopredným šírením	162
10.3 Optimalizácia pomocou neurónových sietí	168
10.3.1 Kombinatorická optimalizácia pomocou Boltzmannovho stroja a Hopfieldovej siete	169
10.3.2 Riešenie problému obchodného cestujúceho pomocou Kohonenovej siete	175
10.4 Záver	178
Literatúra	179
11. Výpočty na báze DNA (Peter Tiňo)	183
11.1 Štruktúra a replikácia DNA	183
11.2 Teoretické modely výpočtového procesu	189
11.3 Formálna notácia operácií nad DNA	190
11.4 Experiment L. Adlemana - hľadanie hamiltonovskej cesty v orientovanom grafe na báze DNA	191
11.5 R. Lipton - Určenie splniteľnosti logických formúl na báze DNA	194
11.6 Počítače na báze DNA sú univerzálne výpočtové systémy	196
11.7 Počítače na báze DNA in vivo	200
11.7.1 Editovanie RNA	200
11.7.2 Intróny a transpozóny	201
11.8 Nová vedná disciplína	203
Literatúra	203
Zoznam význačnejších symbolov a označení	205
Index	208

Predslov

Optimalizácia v informatike znamená hľadať odpoveď na otázku "ktoré riešenie je najlepšie" pre tie problémy, kde sa dá kvalita každého riešenia ohodnotiť jedným číslom (hodnotou optimalizovanej funkcie). Takéto problémy sa vyskytujú v matematike, fyzike, chémii, biológii, ekonómii, stavebníctve, architektúre a mnohých ďalších odboroch. Typické ťažké (NP-úplné) problémy, na ktoré sa používajú stochastické optimalizačné algoritmy, sú napríklad problémy v teórii grafov (pokrytie a rozklad, podgrafy a supergrafy, izomorfizmus), plánovanie sietí (kostry, rezy, spojitost', plánovanie ciest a pod.), rozklady a pokrytie množín, kompresia a reprezentácia dát, pridelenie úloh na multiprocesorovom počítači alebo strojom v továrni, matematické programovanie, hry (snažíme sa dostať robota k cieľu čo najmenším počtom posunov, môžeme pohybovať buď robotom alebo prekážkou), programovanie automatov.

Evolučné a genetické stochastické optimalizačné algoritmy využívajú modely základných mechanizmov evolúcie živej hmoty pre účely optimalizácie. Evolučné a genetické algoritmy pracujú s náhodnými zmenami navrhovaných riešení a nepoužívajú deriváciu optimalizovanej funkcie, iba hodnoty funkcie samotnej. Pokiaľ sú nové riešenia výhodnejšie, nahrádzajú v evolúcii riešení predchádzajúce riešenia.

Vo všeobecnosti evolučných a genetických optimalizačných algoritmov je ich sila ale aj slabosť. Používajú sa ako posledná možnosť, keď nevieme zostrojiť špecializované algoritmy založené na hlbkej vnútornej znalosti problému, alebo zlyhávajú klasické gradientové a negradientové metódy optimalizácie. Na druhej strane evolučné a genetické stochastické optimalizačné algoritmy môžeme použiť na optimalizáciu prakticky akejkoľvek funkcie, ktorej hodnota sa dá dostatočne rýchlo vypočítať.

Popri evolučných a genetických algoritmoch sú v tejto knihe predstavené aj iné stochastické optimalizačné algoritmy ako je "horolezecký" algoritmus, zakázané prehľadávanie (tabu search) a simulované žihanie (simulated annealing), ktoré využívajú iné ako evolučné princípy na účely optimalizácie. Pozornosť je venovaná aj genetickému programovaniu. To sa vyvinulo z genetických algoritmov, no svojim zameraním na symbolickú regresiu (kedy hľadáme nielen koeficienty regresnej funkcie, ale aj regresnú funkciu samotnú) sa oddelilo od ostatných stochastických optimalizačných metód, ktoré sú zamerané na optimalizáciu akejkoľvek funkcie. Najnovšou optimalizačnou metódou, ktorá je predstavená aj v tejto knihe a ktorá na výpočet optimálnej hodnoty nejakej funkcie využíva samotné molekuly nosiča genetickej informácie, t.j. molekuly deoxyribonukleovej kyseliny (DNA), je tzv. počítanie na báze DNA (DNA computing).

Kniha je určená tým pracovníkom vo výskume aj praxi, ako aj študentom vyššie uvedených odborov, ktorí potrebujú zvládnuť stochastické optimalizačné algoritmy pre riešenie svojich pracovných úloh. Kniha môže slúžiť aj pre všeobecné vzdelanie všetkých, ktorí chcú hlbšie preniknúť do problematiky pochopenia evolúcie a optimalizácie.

L'ubica Beňušková

Pod'akovanie

Autori knihy by sa touto cestou chceli poďakovať Dr. Danke Valkovej a Dr. Jozefovi Nosekovi za cenné pripomienky ohľadom objasnenia terminológie a princípov prenosu genetickej informácie v živých organizmoch. Ďalej autori knihy ďakujú recenzentom doc. G. Andrejkovej, doc. M. Brezovi a doc. J. Csontó za starostlivé prečítanie a posúdenie tejto knihy a opravenie množstva chýb a preklepov.

1. Úvodné poznámky

Rozvoj modernej informatiky je pozitívne poznamenaný skutočnosťou, že hľadá inšpiráciu v živej prírode, snaží sa formalizovať a implementovať paradigmy živej prírody pre návrh nových algoritmov, postupov a metód [1]. V tejto knihe sa budeme zaoberať metaforou prevzatou zo živej prírody, a to Darwinovou evolučnou teóriou. Táto metafora viedla ku vzniku evolučných algoritmov [1-14], ktoré patria v súčasnosti medzi aktuálnu problematiku informatiky a numerickej matematiky. Prekvapujúce na tejto novej netradičnej oblasti informatiky je, že okrem toho, že sa už stala organickou súčasťou modernej informatiky, stala sa tiež mostom medzi exaktnými prírodovedeckými oblasťami a vedami o živej prírode a určitou časťou humanitných vied. Informatici majú teraz v rukách mocný simulačný nástroj, pomocou ktorého môžu získať relevantné odpovede na rôzne dôležité otázky a problémy evolučnej biológie, psychológie, sociológie atď., ktoré boli donedávna prístupné len špekulatívnym diskusiam založeným hlavne na analógiách alebo domnienkach autorov. Z pohľadu teórie a histórie vedy ide o unikátny fenomén modernej vedy. "Mosty" medzi rôznymi vednými oblasťami vznikajú obvykle ako výsledok intenzívnych interdisciplinárnych snáh preklenúť dve rôzne vedné oblasti. V našom prípade išlo o pomerne "jednostranný akt" v rámci informatiky, aplikovali sa paradigmy živej prírody ako možné alternatívne algoritmy učenia a adaptácie, a až neskôr sa ukázalo, aký silný simulačný prostriedok má informatika k dispozícii na pochopenie a interpretáciu javov a procesov prebiehajúcich v živej prírode. Na tejto situácii je pre informatikov fascinujúce to, že sa dostali do centra vedeckých aktivít, kde v rámci mierne modifikovaného argumentačného a pojmového aparátu informatiky získavajú relevantné výsledky týkajúce sa evolučných procesov v živej prírode alebo procesov prebiehajúcich na elementárnej úrovni neurónov v mozgu pri jeho kognitívnych aktivitách.

Uvedieme niekoľko poznámok k metafore Darwinovej evolučnej teórie, ktorá obsahuje tieto tri zložky:

1. *Prirodzený výber*, t.j. proces v ktorom jedinci s veľkým fitness vstupujú do procesu reprodukcie s väčšou pravdepodobnosťou ako jedinci s malým fitness. Pod fitness rozumieme kvantitatívnu mieru schopnosti prežiť a vstupovať do reprodukčného procesu.
2. *Náhodný genetický drift*, v ktorom náhodné udalosti v živote jedincov ovplyvňujú populáciu. Takýmito udalosťami sú napr. náhodná mutácia genetického materiálu alebo náhodná smrť jedinca s veľkým fitness predtým, ako mal možnosť zúčastniť sa reprodukčného procesu. Náhodné efekty genetického driftu sú významné hlavne pre malé populácie.
3. *Reprodukčný proces*, v rámci ktorého sa z rodičov vytvárajú potomkovia. Genetická informácia potomkov sa vytvára vzájomnou výmenou genetickej informácie rodičov. Najčastejšie tento proces prebieha tak, že z genetickej informácie dvoch jedincov sa náhodne vyberú časti chromozómu - informácie, z ktorých sa potom zostaví genetická informácia nového jedinca - potomka (tzv. sexuálna reprodukcia). Tento proces sa inak volá križenie a vzhľadom na to, že sa vyskytuje u všetkých zložitejších organizmov, môžeme usúdiť, že podstatne zväčšuje rýchlosť a účinnosť evolúcie.

Evolučné algoritmy založené na metafore Darwinovej evolučnej teórie predstavujú nový netradičný prístup k hľadaniu optimálneho (alebo suboptimálneho) riešenia zložitých

optimalizačných problémov, ktoré nie sú riešiteľné klasickými technikami. Základným pojmom týchto algoritmov je populácia chromozómov, ktoré sú obvykle tvorené lineárnymi reťazcami symbolov. V týchto chromozómoch je zakódované aktuálne riešenie optimalizačného problému, chromozómy zodpovedajúce kvalitnému riešeniu sú ohodnotené veľkým fitness. Vstup do procesu reprodukcie je úmerný fitness chromozómov (chromozómy s veľkým fitness vstupujú s väčšou pravdepodobnosťou do reprodukcie). Táto reprodukcia obsahuje ešte aj určitý náhodný faktor (genetický drift) vyjadrený možnosťou náhodnej mutácie (v reťazci je náhodne vybraný symbol zamenený iným náhodne vybraným symbolom). Táto skutočnosť umožňuje evolúcii hľadať nové riešenia, ktoré sa v populácii ešte vôbec nevyskytli a môžu byť veľmi nádejné pre ďalšiu evolúciu populácie. Po určitom počte generácií sa začnú v populácii vyskytovať jedinci - chromozómy, ktoré reprezentujú vysokokvalitné riešenie daného optimalizačného problému.

Vyššie popísaná základná idea evolučných algoritmov je založená na formalizácii Darwinovej evolučnej teórie, ktorá sa prevzala do informatiky z biológie. V súčasnosti už existuje v informatike celá paleta evolučných algoritmov, ktoré sú založené na rôznej interpretácii Darwinovej evolučnej teórie, na zdôraznení niektorého partikulárneho aspektu atď. Dokonca vznikli evolučné algoritmy, ktoré svoju inšpiráciu nehľadali v živej prírode, ale v neživej prírode - vo fyzike. Dobrou ilustráciou evolučných algoritmov založených na paradigme inej ako Darwinova evolučná teória je metóda simulovaného žihania, ktorá vychádza z predstáv evolúcie termodynamických systémov [10].

Evolučné algoritmy v súčasnosti patria medzi základné nástroje modernej informatiky v prípadoch hľadania riešení v extrémne zložitých situáciách, kedy použitie štandardných deterministických metód založených na technikách úplného prehľadávania nemožno aplikovať (akoby sme hrali šach tak, že najprv zostrojíme všetky možné ťahy až do konca hry a z týchto vyberieme ten ťah, ktorý najrýchlejšie vedie k výhre). Ukazuje sa, že evolučná metafora je veľmi efektívnym prístupom k riešeniu týchto zložitých problémov najmä v prípadoch, keď nepotrebujeme optimálne riešenie problému, ale vystačíme aj s kvalitným suboptimálnym riešením. Základné vlastnosti týchto algoritmov môžeme zhrnúť takto:

1. Evolučné algoritmy patria medzi základné prostriedky modernej numerickej matematiky pre riešenie zložitých optimalizačných problémov. Používajú sa vtedy, ak hľadáme také globálne minimum, ktoré je obklopené množstvom lokálnych miním. Skutočnosť, že sa dajú takto použiť, je prekvapujúca (podobne ako pre neurónové siete vlastnosť, že sú univerzálnym aproximátorom funkcií), adaptáciou a modifikáciou všeobecných predstáv o Darwinovej evolučnej teórii sme dostali univerzálnu numerickú optimalizačnú metódu.
2. Evolučné algoritmy sa môžu vo všeobecnosti chápať ako abstrakcia a formalizácia Darwinovej evolučnej teórie. Pomocou evolučných algoritmov môžeme numericky simulovať darwinovskú evolúciu založenú na prirodzenom výbere a náhodnom genetickom drifte. Evolučné algoritmy poskytujú univerzálny algoritmus pre simuláciu evolúcie, v ktorom je potrebné modifikovať len spôsob určenia fitness chromozómu pomocou zodpovedajúceho reťazca symbolov (v biológii sa tento problém nazýva zobrazenie genotypu na fenotyp, kde lineárny reťazec chromozómu kóduje organizmus, ktorého schopnosť reprodukcie a prežitia je mierou fitness daného chromozómu). Táto skutočnosť otvára veľké možnosti pre informatikov zaoberajúcich sa genetickými algoritmami pre nové netradičné aplikácie v rámci evolučnej biológie, pri sledovaní evolučného vývoja tej-ktorej (aj sociálnej) vlastnosti, ktorá je zakódovaná v chromozóme.

Literatúra

- [1] J. Holland: *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor 1975.
- [2] L. Davis (ed): *Genetic Algorithms and Simulated Annealing*. Morgan Kaufman, Los Altos, CA 1987.
- [3] D. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA 1989.
- [4] M. Mitchel: *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, MA 1996.
- [5] T. Bäck: *Evolutionary Algorithms in Theory and Practice: Evolutionary Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford 1996.
- [6] T. Bäck, D.B. Fogel, and Z. Michalewicz (eds.): *Handbook of Evolutionary Computation*. Oxford University Press and Institute of Physics Publishing, Oxford 1997.
- [7] D. Fogel: *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ 1995.
- [8] J.R. Koza: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA 1992.
- [9] Z. Michalewicz: *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, Berlin 1992.
- [10] P.M.J. van Laarhoven and E.H.L. Aarts: *Simulated Annealing: Theory and Applications*. Reidel, Dordrecht, The Netherlands 1987.
- [11] F. Glover and M. Laguna: *Tabu Search*. Kluwer Academic Publishers, Dordrecht, The Netherlands 1997.
- [12] J. Pospíchal, V. Kvasnička: Genetické algoritmy: nic pro biology. *Pokroky matematiky, fyziky a astronomie* **40** (1995) 1-10.
- [13] V. Kvasnička, J. Pospíchal: Stochastické optimalizační metody v chemii. *Chemické Listy* **88** (1994) 484-490.
- [14] V. Kvasnička, J. Pospíchal: Metóda zakázaného hľadania (tabu search) pre riešenie zložitých optimalizačných úloh. *Československý časopis pro fyziku* **44** (1994) 247-253.

Významné konferencie venované evolučným algoritmom:

GECCO Genetic and EC Conference. Koná sa od r. 1999 každoročne v USA, vznikla spojením konferencií *Annual Genetic Programming Conference GP* (konala sa každoročne od r. 1996) a *International Conference on Genetic Algorithms ICGA* (konala sa v USA v nepárnych rokoch medzi r. 1985 a 1999).

FOGA: Foundations of Genetic Algorithms. Významný medzinárodný workshop zameraný na teoretické aspekty evolučného programovania, limitovaný približne 50 účastníkmi, koná sa od r. 1990 v párnych rokoch v USA.

PPSN: Parallel Problem Solving from Nature. Koná sa od r. 1990 v párnych rokoch v Európe.

ICEC: IEEE Conference on Evolutionary Computation. Koná sa od r. 1994 každoročne v USA.

SAB: International Conference on Simulation Of Adaptive Behavior. Koná sa od r. 1994 každoročne vo Francúzsku.

Okrem týchto konferencií sa ročne poriada celkovo cez 40 medzinárodných konferencií aspoň čiastočne zameraných na evolučné algoritmy.

Časopisy aspoň čiastočne zamerané na evolučné algoritmy:

Adaptive Behavior (MIT Press, vydáva sa od r. 1992)

Artificial Life (MIT Press, vydáva sa od r. 1993)

Autonomous Robots (Kluwer, vydáva sa od r. 1996)

BioSystems. Journal of Biological and Information Processing Sciences (Elsevier Science Publishers, vydáva sa od r. 1968)

Complex Systems (MIT Press, vydáva sa od r. 1987)

Evolutionary Computation (MIT Press, vydáva sa od r. 1993)

Evolutionary Optimization (Wiley, vydáva sa od r. 1998)

Genetic Programming and Evolvable Machines (Springer, vydáva sa od r. 2000)

Heuristics (Kluwer, vydáva sa od r. 1995)

IEEE Transactions on Evolutionary Computation (IEEE, vydáva sa od r. 1997)

Memetics (B. Edmonds, vydáva sa od r. 1997)

Soft Computing (Springer, vydáva sa od r. 1997)

2. Základné pojmy evolučných algoritmov

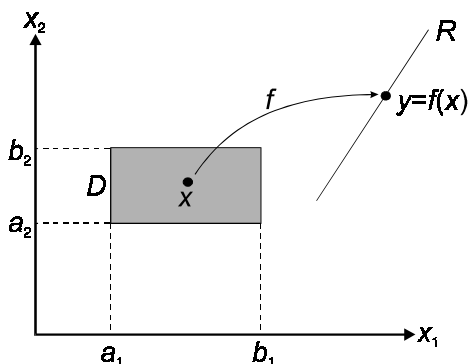
2.1 Optimalizačný problém

Nech funkcia

$$f : D \rightarrow R \quad (2.1a)$$

$$D = \prod_{i=1}^n [a_i, b_i] = [a_1, b_1] \times [a_2, b_2] \times \dots \times [a_n, b_n] \quad (2.1b)$$

zobrazuje n -rozmernú kocku D (karteziánsky produkt uzavretých intervalov $[a_i, b_i]$) na reálne číslo $y \in R$, (pozri obr. 2.1).



Obrázok 2.1. Schematické znázornenie zobrazenia f , ktoré priradí reálne číslo $y \in R$ každému n -rozmernému vektoru $x \in D$.

Táto funkcia je ohraničená dvoma podmienkami:

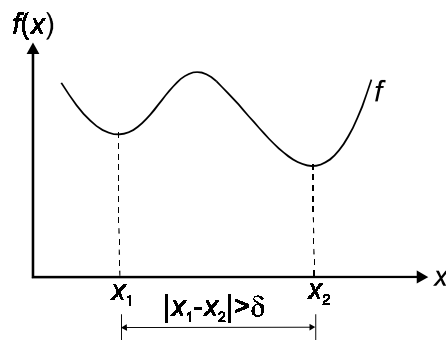
(1) Existuje taký algoritmus, ktorý funkciu f "vypočíta dostatočne rýchlo" s požadovanou presnosťou pre každé $x \in D$ (hovoríme, že funkcia f je *dobře vypočítateľná*).

(2) Pre každú dvojicu lokálnych miním $x_1, x_2 \in D$ je vzdialenosť $|x_1 - x_2|$ väčšia ako dané kladné číslo $\delta > 0$, $|x_1 - x_2| > \delta$ (pozri obr. 2.2). Podmienka zhora ohraničuje počet lokálnych miním funkcie f , ktoré sa vyskytujú na kocke D . Nie je možné, aby sa v ľubovoľnom okolí minima funkcie vyskytovalo iné minimum, pre určité malé okolie minima funkcie by vyššie uvedená podmienka $|x_1 - x_2| > \delta$ prestala platiť. 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

$$x_{opt} = \arg \min_{x \in D} f(x) \quad (2.2)$$

Nájdenie globálneho minima použitím klasických optimalizačných metód [1-3] (gradientových a negradientových) patrí medzi ťažko riešiteľné problémy pre funkcie, ktoré nie sú ohraničené ďalšími podmienkami (napr. že $f(x)$ je konvexná funkcia v oblasti D). Z týchto dôvodov sa v súčasnosti [4,5] pri riešení problému (2.2) často používajú tzv. *evolučné optimalizačné algoritmy*, ktoré poskytujú riešenia blízke globálnemu alebo s ním totožné.

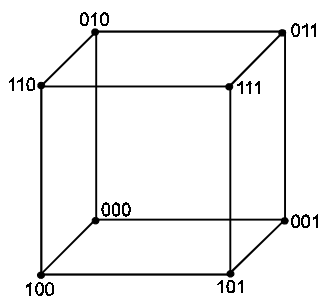


Obrázok 2.2. Ilustrácia druhej podmienky kladenej na funkciu f . Vzdialenosť medzi každou dvojicou miním x_1 a x_2 je väčšia ako predpísané kladné číslo δ , $|x_1 - x_2| > \delta$.

Úloha 2.1. Nech funkcia $f(x)$ má tvar

$$f(x) = e^{-0.01x^2} \sin(mx)$$

pre $m=2,5,10$ a $x \in [-10,10]$. Použitím programových systémov typu MAPLE [6] alebo MATHEMATICA [7] nakreslite priebeh týchto funkcií a nájdite niekoľko prvých najnižších miním (včítane globálneho). Zistite horné odhady konštánt δ (pozri podmienku 2 z kapitoly 2.1) funkcie $f(x)$ pre rôzne hodnoty parametrov m .



Obrázok 2.3. Geometrická interpretácia binárnych vektorov dĺžky $k=3$ pomocou vrcholov kocky. Dva susedné vrcholy odpovedajú binárnym vektorom, ktorých Hammingova vzdialenosť je jednotková (t.j. vektory sa líšia len v jednej polohe).

Binárna verzia funkcie (2.1) má tvar

$$f : \{0,1\}^k \rightarrow R \quad (2.3a)$$

Táto funkcia je definovaná nad množinou binárnych vektorov dĺžky k , zobrazenie f priradí každému binárnemu vektoru reálne číslo z množiny R

$$y = f(\alpha) \quad (2.3b)$$

Kardinalita množiny binárnych vektorov dĺžky k je určená vzťahom

$$|\{0,1\}^k| = 2^k \quad (2.4)$$

To znamená, že "dimenzia" priestoru všetkých možných binárnych vektorov dĺžky k rastie exponenciálne s dĺžkou k . Jednoduchá geometrická interpretácia binárnych vektorov dĺžky $k=3$ je znázornená na obr. 2.3.

Úloha 2.2. Nakreslite analóg kocky na obr. 2.3 pre $k=4$.

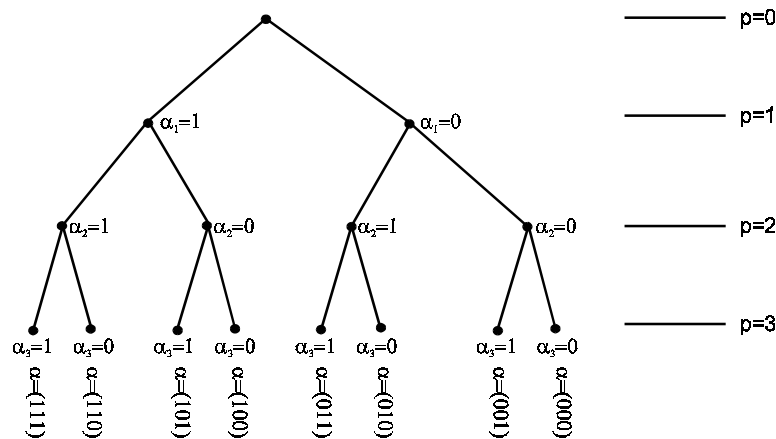
Úloha 2.3. Napíšte program (v jazyku Pascal alebo C) na systematické prehľadávanie všetkých binárnych vektorov dĺžky k pomocou metódy spätného prehľadávania (backtracking) [8] (pozri algoritmus 2.1).

```

procedure Backtrack_Searching(output :  $f_{opt}, \alpha_{opt}$ );
begin  $f_{opt} := \infty$ ;  $p := 1$ ;  $U_1 := 1$ ;
    while  $p \geq 1$  do
        if  $U_p \geq 0$  then
            begin  $\alpha_p := U_p$ ;  $U_p := U_p - 1$ ;
                if ( $p = k$ ) and ( $f(\alpha) < f_{opt}$ ) then
                    begin  $f_{opt} := f(\alpha)$ ;  $\alpha_{opt} := \alpha$  end else
                        if  $p < k$  then
                            begin  $p := p + 1$ ;  $U_p := 1$  end;
                        end else  $p := p - 1$ ;
            end;
    end;

```

Algoritmus 2.1. Implementácia metódy spätného prehľadávania pre riešenie optimalizačného problému (2.5). Priebeh algoritmu pre $k=3$ je znázornený na obrázku 2.4.



Obrázok 2.4. Diagramické znázornenie algoritmu 2.1 spätného prehľadávania pomocou binárneho stromu.

Analóg optimalizačného problému (2.2) pre binárne vektory má tvar

$$\alpha_{opt} = \arg \min_{\alpha \in \{0,1\}^k} f(\alpha) \quad (2.5)$$

Vo všeobecnosti sa globálne optimum α_{opt} nájde po preskúšaní všetkých možných binárnych vektorov dĺžky k . Obrazne povedané, problém (2.5) vyriešime tak, že sa určitým systematickým spôsobom pohybujeme po vrchoch k -rozmernej hyperkocky (pozri obr. 2.3) tak, že navštívime všetkých 2^k vrcholov. Algoritmicke tento prístup možno implementovať pomocou metódy spätného prehľadávania [8], pozri algoritmus 2.1. Čas CPU potrebný na riešenie optimalizačnej úlohy (2.5) je potom úmerný kardinalite priestoru riešení

$$t_{CPU} \propto 2^k \quad (2.6)$$

2.2 Binárna reprezentácia reálnej premennej

Nech sa binárny vektor α dĺžky k

$$\alpha = (\alpha_1 \alpha_2 \dots \alpha_k) \in \{0,1\}^k \quad (2.7)$$

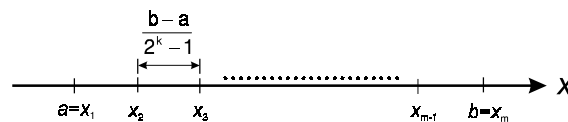
interpretuje ako celé číslo

$$\begin{aligned} \text{int}(\alpha) &= \sum_{i=1}^k \alpha_i 2^{k-i} = \\ &= \alpha_1 2^{k-1} + \alpha_2 2^{k-2} + \dots + \alpha_{k-1} 2 + \alpha_k \end{aligned} \quad (2.8)$$

K tomuto celému číslu jednoduchým spôsobom priradíme reálne číslo, ktoré sa môže chápať ako aproximácia reálneho čísla $x \in [a, b]$

$$x \approx \text{real}(\alpha) = a + \frac{b-a}{2^k-1} \text{int}(\alpha) \quad (2.9)$$

Táto konštrukcia racionálneho čísla $a \leq \text{real}(\alpha) \leq b$ z binárneho reťazca α dĺžky k sa formálne interpretuje ako "transformácia" binárnej reprezentácie na "reálnu" reprezentáciu, kde zostrojené racionálne číslo $\text{real}(\alpha)$ aproximuje požadované reálne číslo x s presnosťou $(b-a)/(2^k-1)$. Interval $[a, b]$ obsahuje $m=2^k$ bodov $x_1=a$, $x_2=a+(b-a)/(2^k-1)$, ..., $x_i=a+(i-1)(b-a)/(2^k-1)$, ..., $x_m=b$, pozri obr. 2.5 a tab. 2.1.



Obrázok 2.5. Reálna premenná x z uzavretého intervalu $[a, b]$ v binárnej reprezentácii sa aproximuje bodmi $x_1, x_2, \dots, x_m$, kde počet bodov $m=2^k$. Presnosť tejto aproximácie je určená dĺžkou binárnych reťazcov k , vzdialenosť medzi dvoma susednými bodmi je $(b-a)/(2^k-1)$.

Tabuľka 2.1. Reprezentácia čísel

No.	α	$\text{int}(\alpha)$	$\text{real}(\alpha)$	$\tilde{\alpha}$ (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

Inverzná transformácia k vzorcu (2.9) má tvar

$$\text{int}(\alpha) = \left\lceil \frac{x-a}{b-a} (2^k - 1) \right\rceil \quad (2.10)$$

kde symbol $\lceil x \rceil$ predstavuje celú časť reálneho čísla x (napr. $\lceil 1.1 \rceil = 1$ a $\lceil 1.9 \rceil = 1$). V dôsledku toho, že $a \leq x \leq b$, zlomok $(x-a)/(b-a)$ leží v uzavretom intervale $[0, 1]$, ak $\text{int}(\alpha)=0$, potom $x=a$; ak $\text{int}(\alpha)=2^k-1$, potom $x=b$.

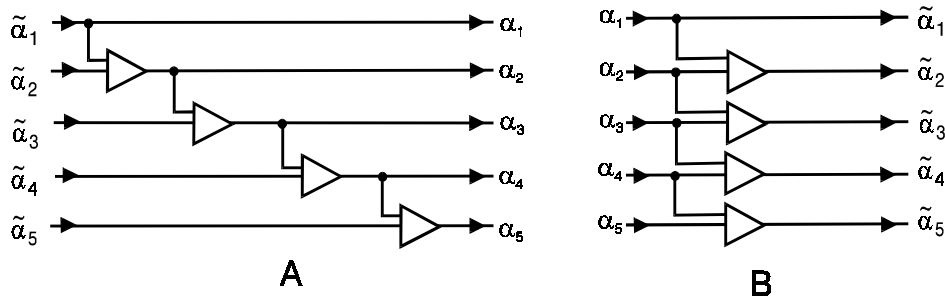
Vyššie uvedená binárna reprezentácia má jednu podstatnú nevýhodu. Dvojica binárnych reťazcov, ktoré sa líšia vo všetkých polohách bitových premenných, môže odpovedať dvom susedným celým číslam (pozri napr. tab. 2.1, kde komplementárne binárne reťazce $\alpha_1=(011)$ a $\alpha_2=(100)$ sa interpretujú ako celé čísla $\text{int}(\alpha_1)=4$, resp. $\text{int}(\alpha_2)=5$). Táto nevýhoda štandardného binárneho kódu sa odstraňuje použitím tzv. Grayovho kódu [9]. 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 (pozri tab. 2.1, kde napr. binárne reťazce $\tilde{\alpha}_1=(010)$ a $\tilde{\alpha}_2=(110)$, ktoré sa líšia len v prvej polohe reťazca, odpovedajú susedným celým číslam $\text{int}(\tilde{\alpha}_1)=4$ resp. $\text{int}(\tilde{\alpha}_2)=5$). Konštrukcia racionálneho čísla z intervalu $[a,b]$ (ktoré, ako už bolo povedané vyššie, aproximuje reálne číslo $a \leq x \leq b$) sa zakladá na tom, že binárne číslo v Grayovom kóde najprv pretransformujeme do štandardného kódu a až potom sa použije vzťah (2.9)

$$x \approx \text{real}(\tilde{\alpha}) = a + \frac{b-a}{2^k - 1} \text{int}(\tilde{\alpha}) \quad (2.11)$$

kde $\tilde{\alpha} = (\tilde{\alpha}_1, \tilde{\alpha}_2, \dots, \tilde{\alpha}_k)$ je binárny reťazec zostrojený pomocou Grayovho kódu a $\text{int}(\tilde{\alpha})$ je odpovedajúce celé číslo priradené reťazcu $\tilde{\alpha}$

$$\text{int}(\tilde{\alpha}) = \text{int}(\alpha) \quad (2.12)$$

kde sa jednotlivé komponenty reťazca $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_k)$ určia pomocou zložiek reťazca $\tilde{\alpha} = (\tilde{\alpha}_1, \tilde{\alpha}_2, \dots, \tilde{\alpha}_k)$ (pozri obr. 2.6)



Obrázok 2.6. Transformácie Grayovho kódu na štandardný kód (A, pozri (2.13)) resp. štandardného kódu na Grayov kód (B, pozri (2.14)) použitím algebraickej operácie "xor" reprezentovanej trojuholníkmi.

$$\begin{aligned} \alpha_1 &= \tilde{\alpha}_1 \\ \alpha_2 &= \tilde{\alpha}_1 \oplus \tilde{\alpha}_2 = \alpha_1 \oplus \tilde{\alpha}_2 \\ \alpha_3 &= \tilde{\alpha}_1 \oplus \tilde{\alpha}_2 \oplus \tilde{\alpha}_3 = \alpha_2 \oplus \tilde{\alpha}_3 \\ &\dots\dots\dots \\ \alpha_k &= \tilde{\alpha}_1 \oplus \tilde{\alpha}_2 \oplus \dots \oplus \tilde{\alpha}_k = \alpha_{k-1} \oplus \tilde{\alpha}_k \end{aligned} \quad (2.13)$$

kde binárna operácia "+" je definovaná ako "xor" súčet binárnych čísel, $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$ a $1 \oplus 1 = 0$. Tieto vzťahy (2.13) možno jednoducho verifikovať na príkladoch z tabuľky 2.1. Posledný stĺpec tejto tabuľky (Grayov kód) sa použitím (2.13) pretransformuje na druhý stĺpec (štandardný kód). Inverzná transformácia vzhľadom na (2.13) (t.j. konštrukcia Grayovho kódu zo štandardného kódu) sa jednoducho určí ako riešenia rovníc (2.14)

$$\begin{aligned}
\tilde{\alpha}_1 &= \alpha_1 \\
\tilde{\alpha}_2 &= \alpha_1 \oplus \alpha_2 \\
\tilde{\alpha}_3 &= \alpha_2 \oplus \alpha_3 \\
&\dots\dots\dots \\
\tilde{\alpha}_k &= \alpha_{k-1} \oplus \alpha_k
\end{aligned}
\tag{2.14}$$

Úloha 2.4. Napíšte program, ktorý transformuje binárne vektory dĺžky k na reálne čísla z intervalu $[-10,10]$, pričom sa využíva tak štandardné, ako aj Grayovo kódovanie. Súčasťou programu nech je aj inverzná transformácia vypočítaného reálneho čísla na binárny tvar. Tento zostrojený binárny vektor musí byť totožný s binárnym vektorom, ktorý vstupoval do programu. Zreprodukuje tabuľku 2.1 a zostrojte pomocou programu jej rozšírenú verziu pre $k=4$.

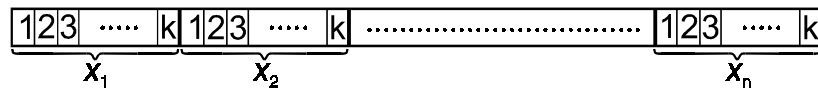
Úloha 2.5. Aká musí byť dĺžka binárnej reprezentácie premennej funkcie $f(x)$ z úlohy 2.1, ak chceme získať výsledky s presnosťou 10^{-2} ? Je výsledok kompatibilný so získanými hodnotami konštant δ z úlohy 2.1?

2.3 Transformácia spojitého optimalizačného problému na binárny optimalizačný problém

Našou hlavnou úlohou je riešiť spojitý optimalizačný problém (2.2) pre globálne minimum funkcie $f(x)$ nad oblasťou D . Pre ďalšie úvahy o použití evolučných algoritmov na riešenie tohto problému bude užitočné pretransformovať tento spojitý optimalizačný problém (2.2) na binárny optimalizačný problém (2.5). Predpokladajme, že každá z n premenných vektora $x \in D$ je vyjadrená v binárnej reprezentácii bitovým vektorom dĺžky k . To znamená, že vektor $x \in D$ je v binárnej reprezentácii vyjadrený bitovým vektorom dĺžky kn (pozri obr. 2.7). Nech funkcia f vyhovuje druhej podmienke z odseku 2.1 pre danú kladnú konštantu δ . Budeme predpokladať, že dĺžka binárnej reprezentácie k je zvolená tak, že platí

$$\delta \gg \frac{(b_i - a_i)}{(2^k - 1)} \quad (\text{pre } i = 1, 2, \dots, n) \tag{2.15}$$

Táto podmienka vyžaduje, aby minimálna vzdialenosť medzi dvoma minimami funkcie $f(x)$ nad oblasťou D bola omnoho väčšia ako "presnosť" binárnej reprezentácie pre každú premennú vektora $x \in D$.



Obrázok 2.7. Schematické znázornenie vektora $x \in D$ (obsahujúceho n zložiek) a jeho binárnej reprezentácie pomocou bitových vektorov dĺžky k . Výsledný bitový vektor má dĺžku kn .

Prechod od binárneho vektora $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_{kn}) \in \{0,1\}^{kn}$ k spojitému vektoru $x = (x_1, x_2, \dots, x_n) \in D$ sa môže formálne chápať ako transformácia

$$\Gamma : \{0,1\}^{kn} \rightarrow D \tag{2.16a}$$

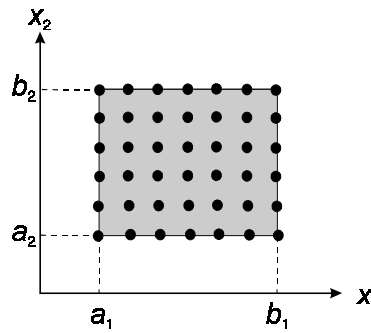
$$x = \Gamma(\alpha) \tag{2.16b}$$

ktorá zobrazuje množinu binárnych vektorov dĺžky kn na body - n -tice reálnych čísel z kocky D . Ináč povedané, konečná množina (2^{kn}) binárnych vektorov dĺžky kn je reprezentovaná pomocou zobrazenia Γ bodmi, ktoré môžu byť v oblasti D usporiadané do ortogonálnej mriežky (pozri obr. 2.8). Minimalizačný problém (2.2) pri použití binárnej reprezentácie n -rozmerných vektorov $\mathbf{x}$ sa teda realizuje na konečnej množine diskretných bodov. Označme $\tilde{\alpha}_{opt}$ binárny vektor dĺžky kn , ktorý bol získaný riešením optimalizačného problému (2.5), pričom funkcia f tohto problému je totožná s funkciou f v optimalizačnom probléme (2.2)

$$\tilde{\alpha}_{opt} = \arg \min_{\alpha \in \{0,1\}^{kn}} f(\Gamma(\alpha)) \quad (2.17a)$$

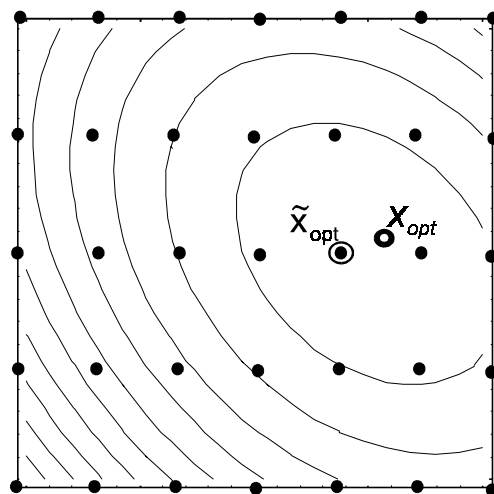
Vektor reálnych premenných priradený tomuto binárnemu vektoru je $\tilde{\mathbf{x}}_{opt} = \Gamma(\tilde{\alpha}_{opt})$. Budeme predpokladať, že presnosť binárnej reprezentácie (t.j. počet bitov k rezervovaných pre každú reálnu premennú) je taká, že vektor $\tilde{\mathbf{x}}_{opt}$ je blízky presnému riešeniu $\mathbf{x}_{opt}$ spojitého optimalizačného problému (2.2) funkcie f nad oblasťou D (pozri obr. 2.8).

$$\mathbf{x}_{opt} \approx \tilde{\mathbf{x}}_{opt} = \Gamma(\tilde{\alpha}_{opt}) \quad (2.17b)$$



Obrázok 2.8. Oblasť D je aproximovaná ortogonálnou mriežkou bodov, ktoré majú binárnu reprezentáciu vyjadrenú pomocou kn -rozmerných vektorov.

Presnosť riešenia optimalizačného problému (2.2) pri prechode zo spojitej reprezentácie do binárnej reprezentácie závisí od konštanty 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, pozri obr. 2.2) 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 byť pomerne veľká. Ortogonálna mriežka bodov nad oblasťou D , ktoré sú generované zvolenou binárnou reprezentáciou reálnych premenných, musí byť dostatočne jemná, aby sa postihli a odlišili blízko seba ležiace minimá funkcie f .



Obrázok 2.9. Vrstevnicový graf funkcie f nad oblasťou D , ktorá je v binárnej reprezentácii aproximovaná ortogonálnou mriežkou bodov. Presné minimum funkcie f nad oblasťou D je označené $\mathbf{x}_{opt}$, minimum funkcie f nad bodmi ortogonálnej mriežky je označené $\tilde{\mathbf{x}}_{opt}$. Vychádzajúc z podmienky, že parametre binárnej reprezentácie boli zvolené tak, aby vhodne aproximovali oblasť D (dĺžka k binárnych vektorov je dostatočná), potom vektor $\tilde{\mathbf{x}}_{opt}$ dobre aproximuje presné riešenie $\mathbf{x}_{opt}$.

Literatúra

- [1] L. Lukšan: *Metody s proměnnou metrikou*. Academia, Praha 1990.
- [2] M. Maňas: *Optimalizační metody*. SNTL, Praha 1979.
- [3] A. Brunovská: *Malá optimalizácia. Metódy, programy, príklady*. Alfa, Bratislava 1990.
- [4] T. Bäck: *Evolutionary Algorithms in Theory and Practice. Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, New York 1996.
- [5] D.B. Fogel: *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*. IEEE Press, New York 1995.
- [6] B. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, and S. Watt: *Maple V*. Springer Verlag, Berlin 1992.
- [7] S. Wolfram: *Mathematica. A System for Doing Mathematics by Computers*. Addison Wesley, Reading, MA 1993.
- [8] L. Kučera: *Kombinatorické algoritmy*. SNTL, Praha 1983.
- [9] R. Hamming: *Coding and Information Theory*. Prentice-Hall Inc., Englewood Cliffs, NJ 1980.

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 ako to riešenie, ktoré už bolo zaznamenané v predchádzajúcej histórii algoritmu. Z dôvodov kompatibility tohto algoritmu s evolučnými algoritmi uvedieme jeho implementáciu pre binárnu reprezentáciu vektorov - riešení (pozri algoritmus 3.1).

```
procedure Blind_Algorithm(input:  $t_{max}, k, n$ ; output:  $\alpha_{fin}, f_{fin}$ );  
begin  $f_{fin} := \infty$ ;  $t := 0$ ;  
  while  $t < t_{max}$  do  
    begin  $t := t + 1$ ;  
       $\alpha :=$  náhodne generovaný binárny  
        vektor dĺžky  $kn$ ;  
      if  $f(\Gamma(\alpha)) < f_{fin}$  then  
        begin  $\alpha_{fin} := \alpha$ ;  $f_{fin} := f(\Gamma(\alpha))$  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 jednotlivéj 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

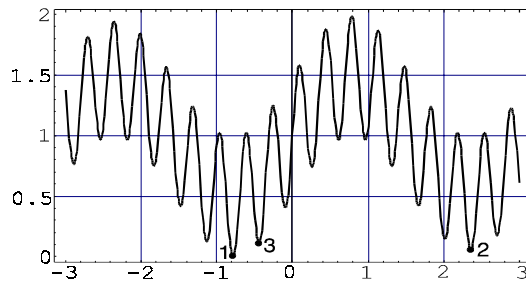
$$\lim_{k_{max} \rightarrow \infty} P(t_{max} | \alpha_{fin} = \alpha_{opt}) = 1 \quad (3.1)$$

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

Úloha 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: $x_1 = -0.7853024$, $f(x_1) = 5.69 \times 10^{-11}$ (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 $\delta = 1/4$. To znamená, že neexistuje taká dvojica dvoch susedných minim, ktorých vzdialenosť by bola menšia ako $\delta = 1/4$. V nasledujúcich úlohách sa táto funkcia bude často používať tak v jednorozmernej, 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 minim a lokálnych extrémov má táto funkcia?

Úloha 3.2. Zovšeobecnite program z úlohy 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 úlohy 3.1

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

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

Úloha 3.3. Zovšeobecnite program pre slepý algoritmus (pozri algoritmus 3.1) pre funkciu $f(x)$ z úlohy 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. Pri formalizácii horolezeckého algoritmu zavedieme niektoré základné pojmy, ktoré sú dôležité pre jeho jednoduchý popis. Operácia *mutácie* stochasticky transformuje binárny vektor α na nový binárny vektor α' , pričom stochastičnosť toho procesu je určená pravdepodobnosťou P_{mut}

$$\alpha' = O_{mut}(\alpha) \quad (3.2a)$$

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

$$\alpha = (\alpha_1, \alpha_2, \dots, \alpha_{kn}) \text{ a } \alpha' = (\alpha'_1, \alpha'_2, \dots, \alpha'_{kn}) \quad (3.2b)$$

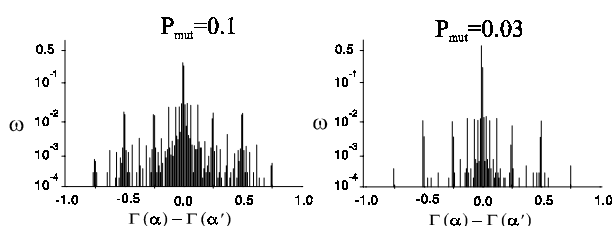
kde jednotlivé komponenty α' sú určené takto

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

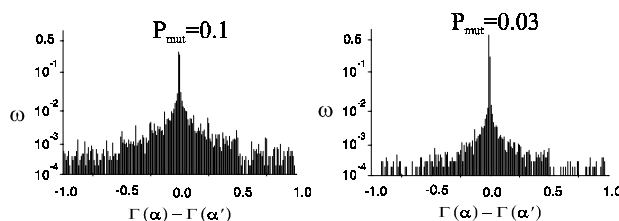
kde *random* je náhodné číslo z intervalu $[0, 1)$ generované s rovnomernou distribúciou (pozri algoritmus 3.2). Pravdepodobnosť P_{mut} určuje stochastičnosť operátora mutácie, v limitnom prípade ak $P_{mut} \rightarrow 0$, potom operátor O_{mut} nemení binárny vektor

$$\lim_{P_{mut} \rightarrow 0} O_{mut}(\alpha) = \alpha \quad (3.2d)$$

Štandardné binárne kódovanie



Grayovo binárne kódovanie



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 binárny 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 $-1 \leq \Gamma(\alpha) - \Gamma(\alpha') \leq 1$. 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 obrázku je vidieť, že mutácia pre binárne vektory so štandardným kódovaním vytvára nové binárne vektory s číselnými hodnotami, ktoré sú rozdelené diskretným 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 diskretné rozdelené okolo nuly. Táto "diskretnosť 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).

Úloha 3.4. Zreproduktujte grafy z obr. 3.1. Zvolte 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} dostanete nový binárny reťazec α' , $\alpha' = O_{mut}(\alpha)$. Pôsobenie operátora mutácie sa realizuje vzťahmi (3.2a-c). Zostrojíte 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 sa realizuje 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 $z \in [a-b, b-a]$ priradiť index podľa predpisu

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

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

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

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  $\omega_i := 0$ ;
time:=0
while time < timemax do
begin
time:=time+1;
 $\alpha :=$  náhodne generovaný binárny vektor;
 $\alpha' := O_{mut}(\alpha)$ ;
 $\Delta(\alpha, \alpha') := \Gamma(\alpha) - \Gamma(\alpha')$ ;
 $index := 1 + \left\lceil \frac{\Delta(\alpha, \alpha') - a + b}{\xi} \right\rceil$ ;

 $\omega_{index} := \omega_{index} + 1$ ;
end;
for i:=1 to N do  $\omega_i := \omega_i / N$ ;

```

Grafy z obr. 3.1 dostanete tak, že pomocou vhodného softvéru vynesiete 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 zvolené 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 početkrát, 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:  $\alpha$ ; output  $\alpha'$ );
begin for i:=1 to kn do
    if random <  $P_{mut}$  then
         $\alpha'_i := 1 - \alpha_i$  else  $\alpha'_i := \alpha_i$ ;
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 $U(\alpha)$ binárneho vektora α sa zostrojí pomocou vektorov $\alpha' = O_{mut}(\alpha)$

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

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

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

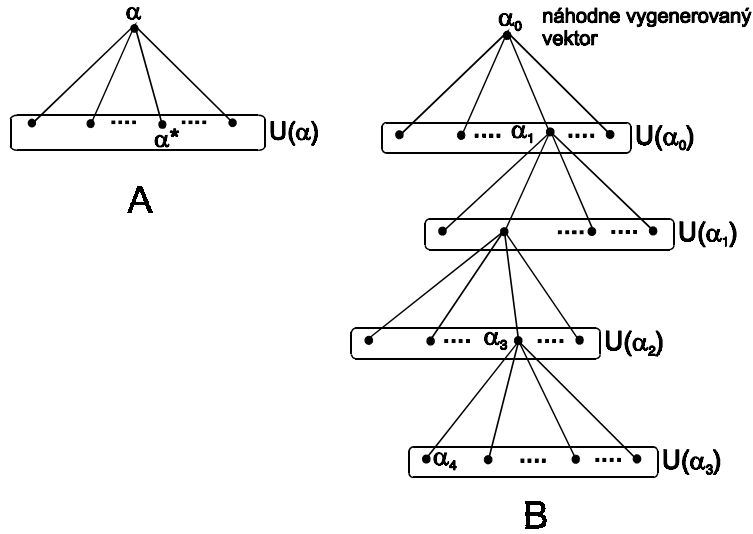
V horolezeckom algoritme sa takto získané riešenie α^* použije ako "stred" v ďalšom iteračnom kroku algoritmu (pozri obr. 3.2). Implementácia horolezeckého algoritmu v pseudopascalu je uvedená v algoritme 3.3.

Analógia 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 však kardinalita okolia $c_0 = |U(\alpha)|$ musí asymptoticky zväčšovať do nekonečna

$$\lim_{c_0 \rightarrow \infty} P(c_0 \mid \alpha_{fin} = \alpha_{opt}) = 1 \quad (3.5)$$

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

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 za 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:  $t_{max}, c_0, P_{mut}$ ; output:  $f_{fin}, \alpha_{fin}$ );
begin  $\alpha :=$  náhodne generovaný binárny vektor dĺžky  $kn$ ;
     $f_{fin} := \infty$ ;  $t := 0$ ;
    while  $t < t_{max}$  do
    begin  $t := t + 1$ ;
         $\alpha^* = \arg \min_{\alpha \in U(\alpha)} (\Gamma(\alpha))$ 

        if  $f(\Gamma(\alpha^*)) < f_{fin}$  then

        begin  $f_{fin} := f(\Gamma(\alpha^*))$ ;
             $\alpha_{fin} := \alpha^*$ 
        end;
         $\alpha := \alpha^*$ 
    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 reprezentácie 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} .

Úloha 3.5. Napíšte program pre horolezecký algoritmus (pozri algoritmus 3.3) pre funkciu $f(x)$ definovanú v úlohe 3.3 a jej n -rozmerné zovšeobecnenie (pozri úlohu 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 $n=2,3,4$. Zistíte, 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 výsledky uvedené v tejto tabuľke.

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 $U(\alpha)$. Pôvodná definícia okolia (3.3) využíva stochastický operátor mutácie O_{mut} , zo "stredy" 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 ($P_{mut} \approx 0$), potom nové stavy generované mutačným operátorom sú veľmi blízke pôvodnému stavu α (t.j. stredy okolia $U(\alpha)$). Opačne, ak sa pravdepodobnosť P_{mut} blíži k 1/2, potom okolie $U(\alpha)$ obsahuje binárne vektory, ktoré sú veľmi vzdialené od "stredy" α . Táto jednoduchá úvaha nás vedie k myšlienke konštrukcie okolia tak, že pre každú polohu binárneho vektora máme zadanú zvlášť pravdepodobnosť. Zavedieme dva nové koncepty, ktoré zaujímavým spôsobom umožňujú modifikovať horolezecký algoritmus

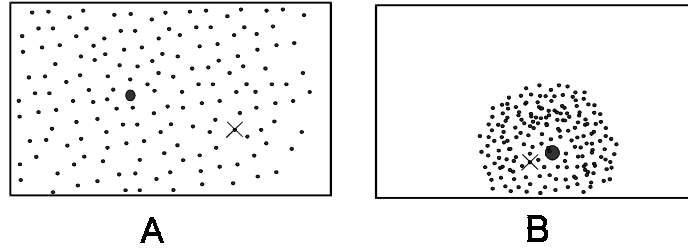
(1) *Pravdepodobnostný vektor* $w = (w_1, w_2, \dots, w_{kn}) \in [0,1]^{kn}$. Jeho jednotlivé zložky $0 \leq w_i \leq 1$ určujú pravdepodobnosti výskytu premennej '1' v danej pozícii. Napr. ak $w_i = 0(1)$, potom $\alpha_i = 0(1)$, pre $0 < w_i < 1$ je premenná α_i náhodne určená vzťahom

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

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

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

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



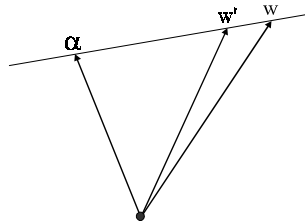
Obrázok 3.3. Schematické znázornenie okolia $U(\mathbf{w})$ pre daný pravdepodobnostný vektor $\mathbf{w}$. Za predpokladu, že všetky zložky $\mathbf{w}$ sú blízke $1/2$, potom binárne vektory okolia $U(\mathbf{w})$ sú "rozmiestnené" v celom priestore binárnych vektorov $\{0,1\}^{kn}$ (diagram A). V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

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

$$B(\mathbf{w}) = \arg \min_{\alpha \in U(\mathbf{w})} f(\Gamma(\alpha)) \quad (3.8)$$

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

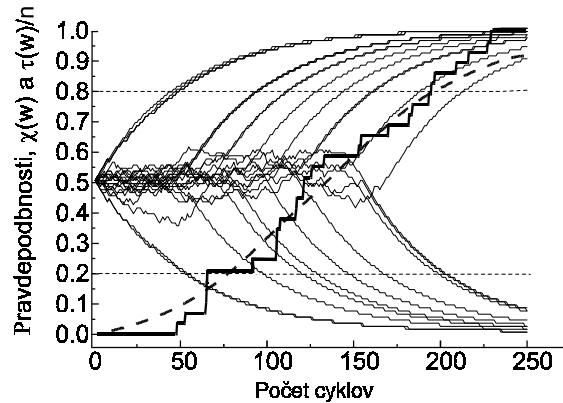
$$\mathbf{w} \leftarrow \mathbf{w} + \lambda \sum_{\alpha \in B(\mathbf{w})} (\alpha - \mathbf{w}) \quad (3.9)$$



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

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

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 sa vektory okolia generujú pomocou pravdepodobnostného vektora $\mathbf{w}$. Okrem toho, pravdepodobnostný vektor $\mathbf{w}$ sa systematicky obnovuje pomocou Hebbovho učenia, ktoré ho posúva smerom k najlepším riešeniam $B(\mathbf{w})$ z množiny riešení $U(\mathbf{w})$ generovanej pomocou pravdepodobnostného vektora $\mathbf{w}$, $B(\mathbf{w}) \subset U(\mathbf{w})$ (pozri algoritmus 3.4).



Obrázok 3.5. Priebeh jednotlivých pravdepodobností vzhľadom na počet iterácií 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 $\chi(\mathbf{w})$, schodová neprerušovaná čiara odpovedá veličine $\tau(\mathbf{w})/n$, kde n je počet premenných (v našom prípade $n=1$). Funkcie $\chi(\mathbf{w})$ a $\tau(\mathbf{w})$ sú definované vzťahmi (3.10) resp. (3.11), pričom $w_{\text{eff}}=0.2$.

```

procedure HCwL(input:timemax,c0,b,λ;output:αfin);
begin for i:=1 to n do wi:=0.5;
    time:=0;
    while time<timemax do
        begin time:=time+1;
            B(w):=arg minb f(Γ(a))
                    a∈U(w)
            w:=w+λ ∑α∈B(w) (α-w);
        end;
        αfin:= najlepšie riešenie z B(w);
    end;

```

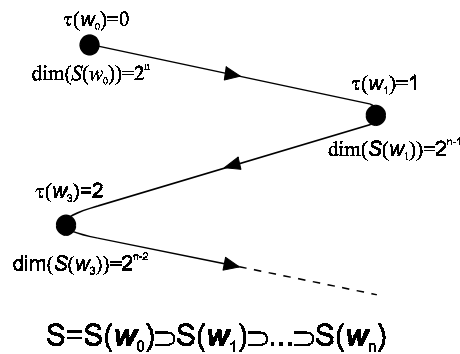
Algoritmus 3.4. Pseudopascalovská procedúra realizujúca horolezecký algoritmus s učením (HCwL, Hill Climbing with Learning). Algoritmus sa inicializuje tak, že pravdepodobnosti w_i sú rovné $1/2$. Vonkajší cyklus **while** sa opakuje $time_{\text{max}}$ iterácií. V rámci tohto cyklu sa zostrojí množina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

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

Pre počiatočný vektor pravdepodobností $\mathbf{w}^{(0)}=(1/2,1/2,\dots,1/2)$ 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 $1-\varepsilon$ (kde ε je malé kladné číslo), potom metóda je ukončená, pretože výsledný binárny vektor určený ako najlepšie riešenie z množiny $B(\mathbf{w})$ 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 $\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_n$, kde $\mathbf{w}_i$ je pravdepodobnostný vektor, v ktorom je i zložiek už fixovaných buď na 0 alebo 1. Na začiatku procesu má vektor $\mathbf{w}_0$ všetky zložky približne rovné $1/2$. To znamená, že odpovedajúci priestor riešení obsahuje všetkých možných 2^n binárnych vektorov dĺžky n , t.j. $S(\mathbf{w}_0)=\{0,1\}^n$. Pre prechodný pravdepodobnostný vektor $\mathbf{w}_i$ (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 $S(\mathbf{w}_i)$ označuje podpriestor zložený z vektorov s tromi zafixovanými zložkami, napr. $(\#\#011\#\#\#\#\#)$, kde i symbolov $\#$ je zamenených buď za 0 alebo za 1, platí $\dim(S(\mathbf{w}_i))=2^{n-i}$. Konečne, vektor $\mathbf{w}_n$ 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*

$$\tau(\mathbf{w}) = \text{počet zložiek } w_i \text{ pravdepodobnostného vektora } \mathbf{w}, \quad (3.11)$$

ktoré sú menšie ako w_{eff} alebo ako $(1-w_{\text{eff}})$,

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

Úloha 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 $f(\mathbf{x})$ bola 95% pravdepodobnosť získania globálneho minima. Pomocou programu zreprodukuje obr. 3.5, kde sú vynesené pravdepodobnosti v závislosti od počtu iteračných krokov. Experimentujte s funkciami $\chi(\mathbf{w})$ a $\tau(\mathbf{w})$, ktoré sú definované vzťahmi (3.10) resp. (3.11), pre určenie vhodného kritéria na zastavenie metódy. Vykreslite priebehy týchto funkcií v závislosti 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 početkrát. 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] naznačujú 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 dôležitú otázku, 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 algoritmickej trikov a heuristik než o metódu so solídnym teoretickým základom. Súčasne však musíme konštatovať, že patrí medzi numericky veľmi efektívne metódy riešenia globálnej optimalizácie nad diskretnými 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

$$S = \{t_1, t_2, \dots, t_p\} \quad (3.12)$$

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

$$t : \{0,1\}^{kn} \rightarrow \{0,1\}^{kn} \quad (3.13a)$$

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

$$t_i(\dots \alpha_i \dots) = (\dots 1 - \alpha_i \dots) \quad (3.13b)$$

pre $i=1,2,\dots,p=kn$. 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 $t_{i_1}, t_{i_2}, \dots, t_{i_n} \in S$, že "východiskový" vektor α_1 sa postupne pretransformuje na "konečný" vektor α_2

$$\alpha_1 = \beta'_1 \xrightarrow{t_{i_1}} \beta'_2 \xrightarrow{t_{i_2}} \dots \xrightarrow{t_{i_n}} \alpha_2 = \beta'_n \quad (3.14)$$

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

$$U(\alpha) = \{t\alpha, \forall t \in S\} \quad (3.15)$$

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 jednoduchšej 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 iteráciách. Zakázaný zoznam transformácií $T \subseteq S$, maximálnej kardinality s , $0 \leq |T| \leq s$, sa 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 iteráciách 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 iteráciách) je každé dodanie novej transformácie sprevádzané aj vylúčením "najstaršej" transformácie (dodanej práve pred s iteráciami); hovoríme, že zakázaný zoznam sa cyklicky obnovuje

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

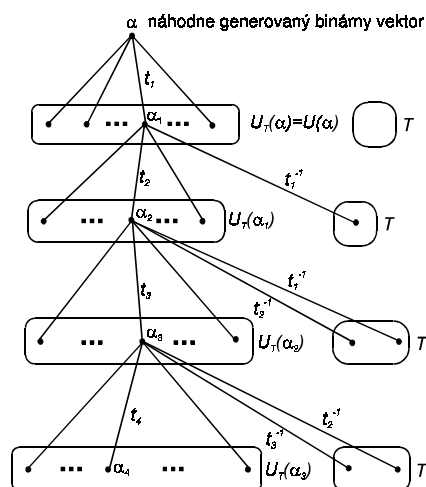
kde t^* je transformácia, ktorá vytvára lokálne minimálne riešenie, $\alpha^* = t^* \alpha$, a $\hat{t}$ je "najstaršia" transformácia zavedená do zakázaného zoznamu práve pred s iteráciami.

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 môže 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 $U_T(\alpha)$

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

ktorého kardinalita $p-s \leq |U_T(\alpha)| \leq p$, pričom $U_T(\alpha) = U(\alpha)$ pre $T = \emptyset$. Toto okolie obsahuje vektory $\alpha' \in \{0,1\}^{kn}$ 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 hľadania 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:timemax,s ; output: ffin,αfin);
begin α:=náhodne generovaný binárny vektor dĺžky kn;
    ffin: =∞; time:=0; T:=∅;
    while t<timemax do
        begin time:=time+1; ffin-loc: =∞;
            for t∈S do
                begin α': =tα;
                    if (t∉T and f(Γ(α'))<ffin-loc) or f(Γ(α'))<ffin then
                        begin α*: =α';
                            t*: =t;
                            ffin-loc: =f(Γ(α'))
                        end;
                    end;
                if ffin-loc<ffin then
                    begin ffin: =ffin-loc; αfin: =α* end;
                α:=α*;
                if |T|<s then T:=T∪{t*-1} else T:=(T∪{t*-1})\{t̂};
            end;
        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 na konštrukciu okolia $U(\alpha)$. 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 $f_{fin-loc}$. Vonkajší cyklus sa ukončí operáciou obnovy zakázaného zoznamu.

Možnosti ďalšej sofistikácie metódy zakázaného hľadania sa obšírne 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

$$f(\alpha') + \chi\omega(t) < f(\alpha^*) \quad (3.18)$$

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.

Úloha 3.7. Napíšte program pre metódu zakázaného hľadania. Program hľadá minima n -rozmernej funkcie (napr. špecifikovanej v úlohe 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 však pokrýva vlastnosti tejto metódy dostatočne všeobecne.

Základná úloha spočíva v riešení nasledujúceho optimalizačného problému

$$\alpha_{opt} = \arg \min_{\alpha \in \{0,1\}^k} f(\alpha) \quad (3.19)$$

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

$$P = \{\alpha_1, \alpha_2, \dots, \alpha_p\} \subseteq \{0,1\}^k \quad (3.20)$$

Každý prvok α z populácie P je ohodnotený funkčnou hodnotou $f(\alpha)$. 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

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

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

$$R = Q \cup Q' \quad (3.22)$$

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

Uvedieme niekoľko poznámok k realizácii tohto operátora. Najjednoduchší prístup spočíva v usporiadaní prvkov R podľa rastúcich funkčných hodnôt $f(\alpha)$, potom S je tvorená prvými $|Q|$ elementmi takto usporiadanej množiny R . Iný prístup je ten, že pre $|Q|$ náhodne vybraných dvojíc $\alpha_1, \alpha_2 \in R$ sa realizuje malý "turnaj"; ak platí $f(\alpha_1) < f(\alpha_2)$, potom sa riešenie α_1 presunie do množiny S (pozri algoritmus 3.6).

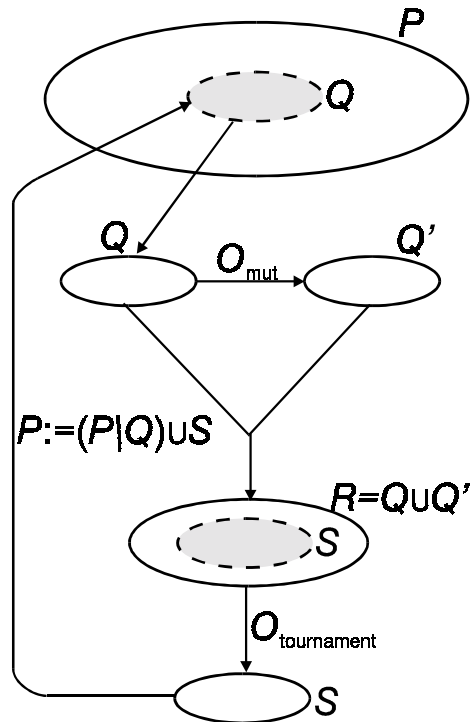
$$S = O_{tournament}(R) \quad (3.23)$$

Pre známu množinu S nasledovníkov obnovíme populáciu P takto

$$P \leftarrow (P \setminus Q) \cup S \quad (3.24)$$

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).

Úloha 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 . Porovnajme 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 podpopuláciu 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  $\alpha_1$  a  $\alpha_2$ ;
            R:=R\{ $\alpha_1$ ,  $\alpha_2$ };
            if  $f(\alpha_1) < f(\alpha_2)$  then S:=S∪{ $\alpha_1$ } else
                S:=S∪{ $\alpha_2$ };
            end;
        end;
    end;

```

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

```

procedure Evolutionary_Programming(output:  $\alpha_{opt}$ );
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:= $Q \cup Q'$ ; S:= $O_{tournament}(R)$ ; P:=(P\Q)  $\cup$  S;
            if konvergenčné kritériá sú splnené then
              stop_criterion:=true;
            end;
             $\alpha_{opt} = \arg \min_{\alpha \in P} f(\alpha)$ ;
      end;

```

Algoritmus 3.7. Implementácia evolučného programovania. Metóda sa inicializuje náhodným generovaním populácie P . Cyklus **while** sa opakuje tak dlho, pokiaľ neplatia podmienky konvergenzie (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. Král: *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 and 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 and D.B. Fogel: *How to Solve It: Modern Heuristics*. Springer Verlag, Berlin 1999.

4. Genetický algoritmus (GA)

4.1 Evolúcia v živej prírode

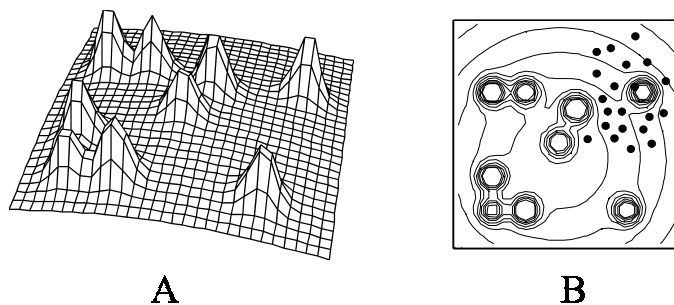
Genetický algoritmus patrí medzi základné stochastické optimalizačné algoritmy s výraznými evolučnými črtami. V súčasnosti je najčastejšie používaným evolučným optimalizačným algoritmom, so širokou paletou aplikácií od optimalizácie vysoko multimodálnych funkcií cez kombinatorické a grafovo-teoretické problémy až po aplikácie nazývané "umelý život".

V predchádzajúcej kapitole bol opísaný tzv. slepý algoritmus stochastickej optimalizácie funkcií. Aj keď je tento algoritmus extrémne jednoduchý, jeho hlavné ohraničenie spočíva v "slepom" náhodnom generovaní nového riešenia bez vzťahu k optimalizovanej funkcii. Je prakticky nepoužiteľný, čas potrebný pre nájdenie minima bude obrovský už pre jednoduché optimalizačné problémy. Metódy numerickej matematiky obchádzajú tento problém tak, že buď využívajú poznatky o tvare funkcie v danom bode (napr. gradientové metódy), alebo sú schopné pomocou predchádzajúcich riešení indikovať s veľkou efektívnosťou nádejný smer minimalizácie funkcie (napr. simplexová metóda). Ukazuje sa, že na základe analógie s evolučnými procesmi prebiehajúcimi v biologických systémoch existuje alternatívna možnosť, ako usmerniť náhodné generovanie bodov k hodnotám blízkym optimálnym. Práve táto analógia sa stala základom genetického algoritmu, ktorý vylepšuje čisto stochastický slepý algoritmus tak, že poskytuje v reálnom čase optimálne riešenie.

Darwinova teória evolúcie [1] sa zakladá na téze prirodzeného výberu, podľa ktorej prežívajú len najlepšie prispôbení jedinci populácie. Reprodukciou dvoch jedincov s vysokým fitness dostávame potomkov, ktorí budú s vysokou pravdepodobnosťou dobre prispôbení na úspešné prežitie. Pri podrobnej analýze (hlavne matematickej) sa ukazuje, že samotné pôsobenie reprodukcie nie je dostatočne efektívne na vznik dobre prispôbených jedincov s novými vlastnosťami, ktoré významne uľahčujú prežitie. Do evolúcie živej hmoty je nutné zapojiť aj tzv. mutácie. Tieto náhodným spôsobom ovplyvňujú (kladne alebo záporne) genetický materiál populácie jedincov.

Biologická evolúcia je progresívna zmena obsahu genetickej informácie (genotypu) populácie v priebehu mnohých generácií. Prv ako pristúpime k jej špecifikácii, zavedieme pojem fitness, ktorý hrá kľúčovú úlohu v našich úvahách o genetickom algoritme. V biológii je *fitness* definovaná ako relatívna schopnosť prežitia a reprodukcie genotypu v danom prostredí. Podobne sa chápe aj v umelom živote, je to kladné číslo priradené genetickej informácii reprezentujúcej organizmus (obvykle vyjadrenej pomocou bitového reťazca), ktoré reprezentuje jeho relatívnu úspešnosť plniť si v danom prostredí svoje úlohy (napr. zber potravy) a vstupovať do reprodukcie, t.j. tvoriť nové organizmy.

Pojem "*povrch fitness*" (fitness' surface/landscape) [2] podstatne uľahčuje heuristickú interpretáciu selekcie v populácii počas priebehu evolúcie. Základná idea povrchu fitness spočíva v grafickej reprezentácii fitness vzhľadom na zloženie genotypu organizmov v danom prostredí (pozri obr. 4.1A). Populáciu potom môžeme znázorniť ako "oblak" bodov na povrchu (pozri obr. 4.1B), každý bod je priradený organizmu z populácie. V priebehu mnohých generácií prirodzený výber spôsobuje pohyb populácie na povrchu do oblastí s väčšou hodnotou fitness. Ak je dominantná selekcia, potom sa tento pohyb uskutočňuje v smere gradientu, v ktorom fitness najrýchlejšie rastie. V opačnom prípade, ak je v evolúcii dominantná náhodnosť (napr. mutácie), pohyb bodov na povrchu fitness má stochastický charakter.



Obrázok 4.1. Znáznorenie 3-rozmerného povrchu fitness (A). Výška jeho jednotlivých vrcholov odpovedá hodnote fitness genetickej informácie (genotypu) organizmu z danej populácie. Vrstevnicový diagram (B) je priradený povrchu fitness, body reprezentujú organizmy v populácii.

Vo vyššom stupni abstrakcie [3] sa môže pojem biologický jedinec nahradiť pojmom chromozóm, ktorý reprezentuje lineárne usporiadaným spôsobom informačný obsah jedince (genotyp). Potom hovoríme o populácii chromozómov, ktoré sa reproduktujú s pravdepodobnosťou úmernou ich fitness, pričom integrálnou súčasťou tejto reprodukcie sú mutácie. Mutácie zanášajú do chromozómov novú informáciu, ktorá môže zvýšiť fitness chromozómov vznikajúcich reprodukciou pôvodných chromozómov. Nové chromozómy vytesňujú z populácie chromozómy s malou fitness. Tento základný reprodukčný cyklus sa v danej populácii chromozómov neustále opakuje. Po určitom čase sa teda v populácii s vysokou pravdepodobnosťou objavia chromozómy s novými vlastnosťami, ktoré podstatne zvyšujú ich fitness a vytesňujú pôvodné chromozómy bez týchto vlastností.

Pristúpme k formalizácii základných pojmov evolúcie tak, ako boli formulované v predchádzajúcej časti tejto kapitoly. Jedincov populácie reprezentujú chromozómy, čo sú lineárne reťazce symbolov. Potom populácia P je množina týchto chromozómov

$$P = \{\alpha_1, \alpha_2, \dots, \alpha_p\} \subseteq \{a, b, \dots\}^k \quad (4.1)$$

Populácia P obsahuje p chromozómov, ktoré sú realizované ako reťazce dĺžky k zložené zo znakov $a, b, \dots$. Obvykle sa tieto znaky rovnajú 0 a 1, chromozómy sú potom binárne reťazce, $P \subseteq \{0, 1\}^k$ (táto binárna konvencia sa bude používať v celej práci). Každý chromozóm $\alpha \in P$ je ohodnotený fitness, ktorá sa interpretuje ako zobrazenie chromozómov na kladné reálne čísla

$$F: P \rightarrow R_+ \quad (4.2)$$

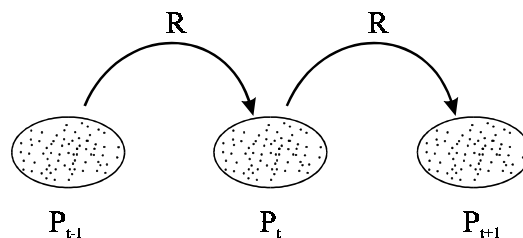
Proces reprodukcie chromozómov začína tým, že z populácie sa kvázinahodne vyberú dva chromozómy v závislosti od ich fitness (chromozómy s väčším fitness majú väčšiu pravdepodobnosť byť vybrané). Pod reprodukciou budeme rozumieť taký proces, v ktorom sa dva pôvodné chromozómy α_1 a α_2 reproduktujú (tvoria potomkov) na dva nové chromozómy α'_1 a α'_2

$$(\alpha'_1, \alpha'_2) = O_{repro}(\alpha_1, \alpha_2) \quad (4.3)$$

Konkrétny tvar tohto operátora reprodukcie nebudeme zatiaľ bližšie diskutovať. Ako ukážeme neskôr, obsahuje dve časti, a to *kříženie* a *mutáciu*. Tento operátor má stochastický charakter, operácie kříženia a mutácie sa vykonávajú len s určitou pravdepodobnosťou. Mutácia už bola špecifikovaná v kapitole 3.1, operáciu kříženia, ktorá znamená výmenu častí medzi dvoma chromozómami, budeme špecifikovať podrobnejšie neskôr.

Vo všeobecnej rovine sa môže evolúcia populácie chápať ako zmena populácie realizujúca sa v čase po jednotlivých krokoch (generáciách). Nech P_t je populácia chromozómov v čase t , potom v nasledujúcom čase $t+1$ je populácia P_{t+1} určená rekurentne pomocou "reprodukčného" R -operátora (pozri obr. 4.2)

$$P_{t+1} = R(P_t) \quad (4.4)$$



Obrázok 4.2. Evolúcia sa môže charakterizovať ako postupná rekurentná zmena populácie, formálne $P_{t+1}=R(P_t)$. R -operátor popisuje rekurentnú zmenu populácie z jednej generácie na druhú. Obsahuje niekoľko zložiek, akými sú výber chromozómov - rodičov, kríženie, mutáciu a návrat chromozómov - potomkov do populácie.

Prístup založený na všeobecnej rovnici (4.4) poskytuje veľmi všeobecný pohľad na evolúciu, avšak za cenu toho, že R -operátor nie je bližšie špecifikovaný. Našu pozornosť upriamime na jednoduchú realizáciu R -operátora tak, že obsahuje problém výberu chromozómov z populácie, reprodukciu (4.3) a návrat nových chromozómov do populácie. Dostaneme jednoduchý evolučný algoritmus, ktorý simuluje evolúciu chromozómov v populácii, pričom hnacou fitness tejto evolúcie je darwinovský prirodzený výber (pozri algoritmus 4.1 a obrázok 4.3). Diskutujme vzťah medzi horolezeckým algoritmom (algoritmus 3.3) a evolučným algoritmom (algoritmus 4.1). V čom sa tieto algoritmy odlišujú? Evolučný algoritmus sa zakladá na simultánnej optimalizácii celej populácie chromozómov, zatiaľ čo horolezecký algoritmus 3.3 optimalizuje náhodne len jeden objekt - chromozóm. Ak v procese reprodukcie chromozómov uvažujeme len mutácie, tak evolučný algoritmus je veľmi podobný horolezeckému algoritmu 3.3. Možnosť úniku z lokálneho minima pomocou (spočiatku málo výhodnej) mutácie v evolučnom algoritme je spojená s potenciálnou nevýhodou, vygenerovaný chromozóm môže mať menšiu fitness ako pôvodný chromozóm. Pri väčšom množstve chromozómov v populácii sa najskôr takýto chromozóm nezúčastňuje na reprodukčnom procese, no neskôr môže nastať situácia, že tieto chromozómy sa ukážu vhodnými pre ďalšiu evolúciu (optimalizáciu) systému. V evolučnom algoritme je lokálne hľadanie optimálneho riešenia horolezeckého algoritmu nahradené efektívnejším spôsobom simultánnej optimalizácie celej populácie chromozómov, pričom jednotlivé chromozómy si v rámci reprodukcie vymieňajú informáciu pomocou kríženia.

Prečo pri prezentácii evolučných algoritmov neustále zdôrazňujeme, že do reprodukčného procesu *vyberáme chromozómy náhodne*? Ak by sme do reprodukčného procesu vyberali len tie chromozómy, ktoré majú najväčšiu fitness, potom by sme s určitou pravdepodobnosťou podstatne ohraničili oblasť D , v ktorej hľadáme optimálne riešenie. Evolučný algoritmus by bol veľmi "oportunistický", za dočasne lepší výsledok by sme asi podstatne ohraničili celý proces evolúcie populácie chromozómov smerom k optimálnemu riešeniu, a tým sa dostali k horšiemu konečnému riešeniu. Preto evolučný algoritmus vyberá chromozómy do procesu reprodukcie *kvázináhodne*. Obrazne, s "básnickou licenciou" môžeme povedať, že v rámci evolučného algoritmu nevystupuje *deus ex machina*, ktorý vopred pozná výsledok evolúcie populácie chromozómov a môže zasahovať výberom chromozómov do procesu reprodukcie. Keby sme toto boli schopní realizovať, potom by sme

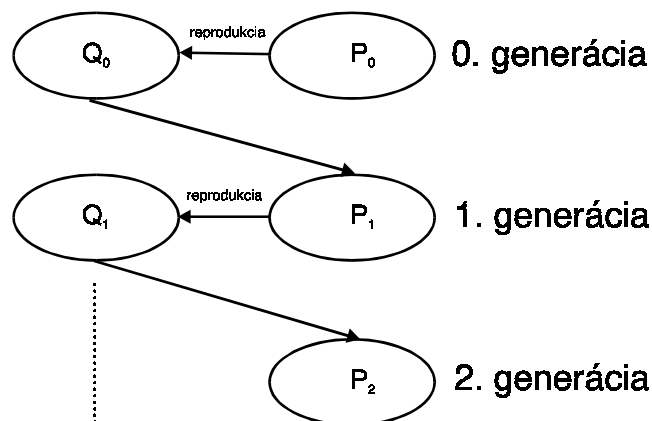
nepotrebovali evolučné algoritmy, pretože optimálne riešenie už poznáme. Všetky "čiastočne urýchľujúce" heuristiky sú nielen nedostatočné, no v konečnom dôsledku aj zavádzajúce, preto ich zabudovanie do evolučného algoritmu je vo väčšine prípadov veľmi problematické.

```

procedure Evolution(input  $t_{\max}$ ; output  $P_{\text{fin}}$ );
begin  $t := 0$ ;
       $P := \{\text{náhodne generovaná populácia chromozómov}\}$ ;
      while  $t < t_{\max}$  do
        begin  $t := t + 1$ ;  $Q := \emptyset$ ;
              while  $|Q| < |P|$  do
                begin kvázináhodne vyber dva rodičovské
                      chromozómy  $\alpha_1, \alpha_2 \in P$  s vysokým fitness;
                      if random  $< P_{\text{repro}}$  then  $(\alpha'_1, \alpha'_2) = O_{\text{repro}}(\alpha_1, \alpha_2)$ 
                      else  $(\alpha'_1, \alpha'_2) = (\alpha_1, \alpha_2)$ ;
                       $Q := Q \cup \{\alpha'_1, \alpha'_2\}$ ;
                end;
              end;  $P := Q$ ;
        end;  $P_{\text{fin}} := P$ ;
      end;

```

Algoritmus 4.1. Pseudopascalovská verzia algoritmu evolúcie v živej prírode založenej na Darwinovskom prirodzenom výbere. Algoritmus sa inicializuje náhodným generovaním populácie P . Vonkajší cyklus **while** sa opakuje $t_{\max}$ generácií (iterácií). Vo vnútornom cykle **while** sa zostrojuje nová populácia tak, že sa kvázináhodne vyberú dva chromozómy z populácie P (pravdepodobnosť ich výberu je úmerná ich fitness) a aplikuje sa na ne reprodukčný operátor (4.3). Tento proces reprodukcie sa realizuje s pravdepodobnosťou P_{repro} , v opačnom prípade sa vybrané rodičovské chromozómy jednoducho okopírujú do nových chromozómov - potomkov. Takto vzniknuté dva nové chromozómy sa uložia do novej populácie Q . Vnútorný cyklus **while** sa opakuje tak dlho, až je kardinalita novej populácie Q rovná kardinalite starej populácie P . Populácia P sa potom nahradí novou populáciou Q .



Obrázok 4.3. Diagramatická vizualizácia evolučného algoritmu 4.1. Z počiatočnej populácie P_0 sa aplikovaním reprodukčného procesu generuje nová populácia potomkov Q_0 , ktorá do ďalšej generácie vstupuje ako obnovená populácia P_1 . Tento proces sa iteratívne opakuje $t_{\max}$ -krát.

4.2 Základné pojmy genetického algoritmu - populácia a fitness chromozómov

Genetický algoritmus (GA) vynašiel v polovici 70-tych rokov Holland [3], ale podobne rozmýšľali aj iní [4]. V súčasnosti patrí genetický algoritmus medzi najpopulárnejšie evolučné optimalizačné algoritmy [5,6]. Základné princípy genetického algoritmu sú určitou špecifikáciou všeobecných princípov evolúcie a jej algoritmizácie, ktoré boli uvedené v kapitole 4.1. Pod chromozómom α budeme rozumieť binárny vektor fixnej dĺžky k

$$\alpha = (\alpha_1, \alpha_2, \dots, \alpha_k) \in \{0,1\}^k \quad (4.5)$$

Populácia P obsahuje chromozómy - binárne vektory α

$$P = \{\alpha_1, \alpha_2, \dots, \alpha_p\} \quad (4.6)$$

kde p je kardinalita populácie P , vyjadruje počet chromozómov v P , $p=|P|$. Nech f je účelová funkcia definovaná nad množinou binárnych vektorov dĺžky k

$$f : \{0,1\}^k \rightarrow \mathbb{R} \quad (4.7)$$

ktorá ohodnotí každý binárny vektor $\alpha \in \{0,1\}^k$ reálnym číslom. Našou úlohou bude hľadať globálne minimum tejto funkcie nad množinou $\{0,1\}^k$ (pozri (2.5))

$$\alpha_{opt} = \arg \min_{\alpha \in \{0,1\}^k} f(\alpha) \quad (4.8)$$

Funkcia f "reprezentuje prostredie", v ktorom existujú chromozómy populácie. Povedané pomocou biologickej terminológie, chromozóm α reprezentuje *genotyp* organizmu, zatiaľ čo funkčná hodnota $f(\alpha)$ reprezentuje jeho *fenotyp*. Mierou úspešnosti chromozómu je jeho funkčná hodnota. Pretože hľadáme minimum účelovej funkcie, chromozóm je tým úspešnejší, čím je jeho funkčná hodnota (genotyp) menšia. Na základe týchto úvah môžeme zaviesť ohodnotenie fitness chromozómov ako zobrazenie (4.2), ktoré vyhovuje nasledujúcej podmienke (pozri obr. 4.4)

$$\forall \alpha_1, \alpha_2 \in P : f(\alpha_1) \leq f(\alpha_2) \Rightarrow F(\alpha_1) \geq F(\alpha_2) \geq 0 \quad (4.9)$$

Z praktických dôvodov je vhodné, aby numerické hodnoty fitness boli z otvoreného intervalu $(0,1)$, preto sa zavádza tzv. *renormalizovaná fitness*

$$F'(\alpha) = \frac{F(\alpha)}{\sum_{\alpha \in P} F(\alpha')} \quad (4.10a)$$

$$\sum_{\alpha \in P} F'(\alpha) = 1 \quad (4.10b)$$

kde $0 < F'(\alpha) < 1$ pre každé $\alpha \in P$.

Akým konkrétnym spôsobom možno realizovať zobrazenie hodnôt účelovej funkcie na fitness tak, aby bola splnená podmienka (4.9)? Toto zobrazenie sa dá uskutočniť mnohými spôsobmi, v tejto kapitole ukážeme dva prístupy zostrojenia síl pomocou hodnôt účelovej funkcie. Prvý postup sa zakladá na jednoduchom lineárnom zobrazení funkčných hodnôt na fitness tak, že maximálnej (minimálnej) funkčnej hodnote je priradená minimálna (maximálna) fitness, ostatné funkčné hodnoty sú lineárne interpolované medzi týmito dvoma limitnými hodnotami

$$F(\alpha) = \frac{F_{max} - F_{min}}{f_{min} - f_{max}} f(\alpha) + \frac{f_{min} F_{min} - f_{max} F_{max}}{f_{min} - f_{max}} \quad (4.11a)$$

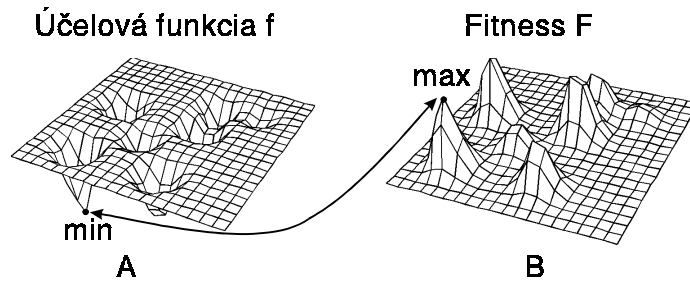
kde f_{max} (f_{min}) je maximálna (minimálna) hodnota účelovej funkcie pre chromozómy populácie P

$$f_{max} = \max_{\alpha \in P} f(\alpha), f_{min} = \min_{\alpha \in P} f(\alpha) \quad (4.11b)$$

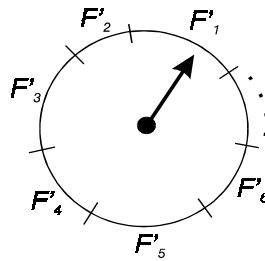
a F_{max} (F_{min}) je maximálna (minimálna) hodnota fitness, obvykle sa položí $F_{max}=1$ a $F_{min}=\varepsilon$, kde ε je malé kladné číslo (napr. $\varepsilon=0.01$). Dosadením týchto hraničných hodnôt fitness do (4.11a) dostaneme konečný výraz pre výpočet fitness chromozómov na základe ich hodnôt účelovej funkcie

$$F(\alpha) = \frac{1}{f_{min} - f_{max}} [(1-\varepsilon)f(\alpha) + f_{min}\varepsilon - f_{max}] \quad (4.11c)$$

Tento vzorec sa často používa v rôznych implementáciách genetického algoritmu v prípade, že hodnoty účelovej funkcie sa veľmi nemenia (napr. sú rovnakého rádu). V opačnom prípade, ak sa tieto funkcie menia vo veľkom intervale idúcim cez mnoho rádov funkčných hodnôt, vzorec (4.11c) priraduje skoro nulovú fitness tým chromozómom, ktoré odpovedajú funkčným hodnotám blízky maximálnej hodnote, a týmto sa neopodstatnene zosilňuje preferencia chromozómov s malou funkčnou hodnotou. Tento nedostatok sa odstráni nasledujúcim alternatívnym prístupom k určeniu síl chromozómov.



Obrázok 4.4. Diagram A znázorňuje účelovú funkciu v priestore všetkých možných binárnych vektorov, kde binárny vektor je formálne pretransformovaný na bod v dvojrozmernom priestore a tretí rozmer (výška) je daný hodnotou účelovej funkcie. Každému chromozómu z populácie P môžeme priradiť funkčnú hodnotu ležiacu na povrchu účelovej funkcie. Diagram B znázorňuje fitness binárnych vektorov - chromozómov. V dôsledku podmienky (4.9), tam kde máme na povrchu A minimum, musí byť na povrchu B maximum (pozri šípku).



Obrázok 4.5. Schematické vyjadrenie rulety pomocou jednotkovej kružnice, na ktorej sú vynesené renormalizované fitness. Obrazne povedané, čím väčšia je renormalizovaná fitness, tým väčší je oblúk priradený na kružnici. Ručička (šípka) sa tlmene točí okolo svojej osi (bodu) a chápe sa ako generátor náhodnej polohy na kružnici. Po jej zastavení ukazuje kvázináhodne vybraný chromozóm.

Nech $Q=(q_1, q_2, \dots, q_{|P|})$ je taká permutácia $|P|$ chromozómov z populácie P , ktorá usporiada hodnoty účelovej funkcie do neklesajúcej postupnosti

$$f(\alpha_{q_1}) \leq f(\alpha_{q_2}) \leq \dots \leq f(\alpha_{q_{|P|}}) \quad (4.12a)$$

Potom chromozómu α_{q_i} ($\alpha_{q_{|P|}}$) priradíme maximálnu (minimálnu) fitness $F_{max}=1$ ($F_{min}=\varepsilon$)

$$F(\alpha_{q_i}) = \frac{1}{1-|P|} [(1-\varepsilon)i + \varepsilon - |P|] \quad (\text{pre } i=1, 2, \dots, |P|) \quad (4.12b)$$

V tomto prípade je fitness chromozómov z populácie P rovnomerne rozložená v celom intervale $[\varepsilon, 1]$, kde ε je malé kladné číslo. Určitým nedostatkom tohto vzorca je, že ohodnocuje chromozómy s rovnakou funkčnou hodnotou rôznymi fitness, čo môže byť nevýhodné hlavne pre záver behu genetického algoritmu, keď je populácia pomerne homogénna. Z týchto dôvodov sa ohodnotenie vzorcom (4.12b) upravuje takto

$$f(\alpha_{q_{i-1}}) = f(\alpha_{q_i}) \Rightarrow F(\alpha_{q_i}) = F(\alpha_{q_{i-1}}) \quad (\text{pre } i=2, 3, \dots, |P|) \quad (4.12c)$$

Potom sú chromozómy s rovnakou hodnotou účelovej funkcie ohodnotené rovnakou fitness.

Úloha 4.1. V kapitole 4.2 sú opísané dve rôzne metódy transformácie hodnôt účelovej funkcie na fitness tak, aby bola splnená podmienka (4.9). Navrhňte iné alternatívne spôsoby tejto transformácie a vykonajte ich interpretáciu v tom zmysle, ako sa líšia od obidvoch pôvodných prístupov.

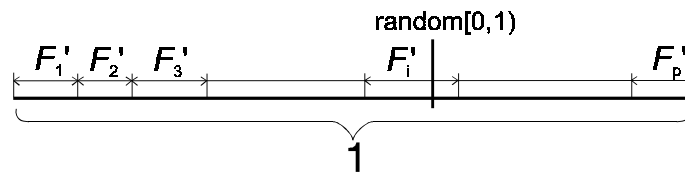
Pomocou renormalizovaných fitness môžeme jednoducho realizovať kvázináhodnosť výberu chromozómov (čo patrí medzi základné imperatívy evolučných algoritmov) pomocou pojmu ruleta [5] (pozri obr. 4.5). Chromozóm s indexom i sa vyberie (kvázináhodne), ak pre náhodne generované číslo random z intervalu $[0, 1)$ s rovnomernou distribúciou pravdepodobnosti platí

$$\sum_{k=1}^{i-1} F'_k \leq \text{random} < \sum_{k=1}^i F'_k \quad (4.13)$$

Tento vzorec má jednoduchú grafickú interpretáciu znázornenú na obr. 4.6 (pozri tiež algoritmus 4.2).

```
function Roulette_Wheel:integer;
begin
  aux:=random; bound_lower:=0; bound_upper:=F'_1;
  i:=0; criterion:=false;
  while (i<|P|) and (not criterion) do
    begin
      i:=i+1;
      criterion:=(bound_lower<=aux) and (aux<bound_upper);
      bound_lower:=bound_upper;
      bound_upper:=bound_upper+F'_{i+1};
    end;
  Roulette_wheel:=i;
end;
```

Algoritmus 4.2. Jednoduchá implementácia rulety pre náhodný výber chromozómov podľa ich fitness. V cykle **while** sa postupne (rekurentne) realizujú súčty renormalizovaných fitness (pozri rov. (4.11) a obr. 4.5). Sofistikovanejší prístup k implementácii rulety je založený na tom, že pred aktiváciami rulety sa spočítajú dolné a horné hranice podintervalov, na ktoré je rozdelený interval $[0, 1]$. Potom hľadáme, napr. metódou binárneho vyhľadávania [7], do ktorého intervalu premenná aux patrí.



Obrázok 4.6. Jednotková úsečka je rozdelená na úseky podľa veľkostí renormalizovaných fitness chromozómov z populácie P . Náhodne generované číslo z intervalu $[0,1)$ predstavuje polohu na úsečke (reprezentované hrubou vertikálnou čiarou) a podľa danej polohy určuje odpovedajúci chromozóm. Z konštrukcie tejto realizácie "rulety" vyplýva, že čím väčšia je fitness chromozómu, tým väčšia je pravdepodobnosť jeho výberu.

Úloha 4.2. Algoritmus 4.2 implementuje ruletu podľa vzťahu (4.13) tak, že sa postupne hľadá interval $[bound_lower, bound_upper]$, ktorý obsahuje náhodne vygenerované číslo $0 < aux \leq 1$. Podstatne efektívnejšia je metóda implementácie rulety pomocou binárneho vyhľadávania [7,14] (porovnaj s detskou hrou určenia prirodzeného čísla menšieho alebo rovného ako 100, dobrý hráč sa pýta oponenta či je dané číslo väčšie (menšie) tak, že neustále delí nádejné intervaly na polovice). Navrhните algoritmus rulety pomocou binárneho vyhľadávania, vykonajte jeho implementáciu.

4.3 Kríženie chromozómov

Pre chromozómy z populácie P sa definujú dve základné operácie: operácia mutácie a operácia kríženia. Prvá operácia (mutácia) už bola podrobne diskutovaná v rámci horolezeckého algoritmu (pozri kapitolu 3.1), preto sa teraz budeme podrobne venovať len operácii kríženia. Majme dva chromozómy $\alpha, \beta \in \{0,1\}^k$, operácia kríženia sa bude formálne interpretovať ako operátor - zobrazenie, ktorý tejto dvojici chromozómov priradí dva nové chromozómy s rovnakou dĺžkou ako pôvodné

$$(\alpha', \beta') = O_{cross}(\alpha, \beta) \quad (4.14)$$

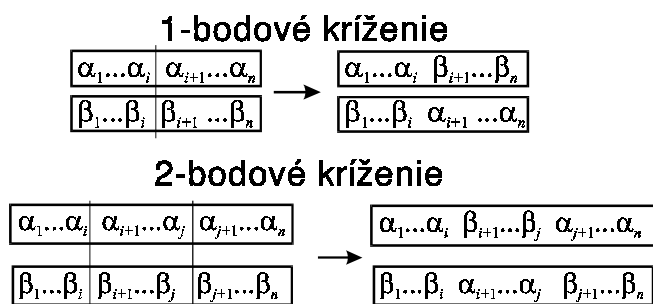
kde sú rôzne spôsoby realizácie operátora O_{cross} znázornené na obr. 4.7 (pozri algoritmus 4.3). Tento operátor má tiež stochastický charakter podobne ako operátor mutácie. Táto stochastičnosť je založená na tom, že pri aplikovaní operátora kríženia na dvojicu chromozómov z populácie sa generuje náhodne tzv. bod (alebo body) kríženia a potom je už aplikácia kríženia úplne deterministická.

```

procedure Crossover(input:  $\alpha, \beta$ ; output:  $\alpha', \beta'$ );
begin cross_point := 1 + random(k-1);
    for i := 1 to cross_point do
        begin  $\alpha'_i := \alpha_i$ ;  $\beta'_i := \beta_i$  end;
    for i := cross_point + 1 to k do
        begin  $\alpha'_i := \beta_i$ ;  $\beta'_i := \alpha_i$  end;
end;

```

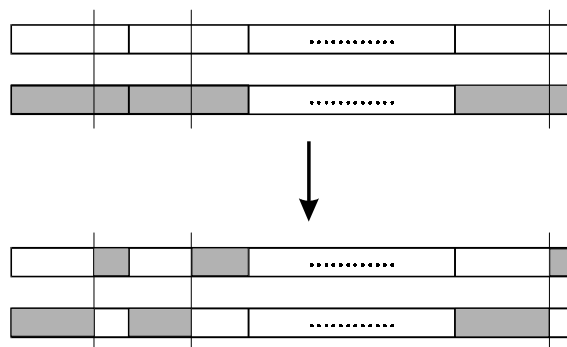
Algoritmus 4.3. Jednoduchá pseudopascalovská implementácia kríženia. Premenná *cross-point* je náhodne generovaný bod kríženia. Po tomto bod sa chromozómy α a β jednoducho okopírujú do nových chromozómov α' resp. β' . Za týmto bodom až do konca chromozómu sa do α' kopíruje zvyšok β a do chromozómu β' sa kopíruje zvyšok α . Funkcia $random(k-1)$ generuje náhodné celé čísla z intervalu $[0, k-2]$.



Obrázok 4.7. Diagramatické vyjadrenie operácie kríženia nad dvoma chromozómami. Vertikálna čiara znázorňuje tzv. bod kríženia. Ilustrujú sa dve verzie kríženia, v 1-bodovom krížení sa náhodne vygeneruje bod kríženia a chromozómy si vymenia svoje časti za týmto bodom. V 2-bodovom krížení sa náhodne vygenerujú dva rôzne body kríženia, chromozómy si vymenia časti medzi nimi.

V prípade, že chromozóm kóduje n reálnych premenných pomocou binárnych reťazcov dĺžky k (celková dĺžka chromozómu je kn), operácia kríženia sa môže aplikovať zvlášť pre každý binárny podreťazec reprezentujúci jednotlivú premennú účelovej funkcie (pozri obr. 4.8).

Úloha 4.3. Rozšírte algoritmus 4.3 (realizujúci 1-bodové kríženie binárneho reťazca dĺžky k) tak, aby sa realizovalo kríženie znázornené na obr. 4.8, kde sa chromozóm skladá z n binárnych podreťazcov dĺžky k .



Obrázok 4.8. Schematické znázornenie kríženia dvoch chromozómov dĺžky kn , kde k je dĺžka binárnej reprezentácie reálnej premennej a n je počet premenných účelovej funkcie. Operácia kríženia prebieha v každom podreťazci zvlášť pre náhodne generované body kríženia (vertikálne čiary v hornej časti diagramu).

Pomocou operátorov mutácie a kríženia môžeme pristúpiť ku konštrukcii operátora reprodukcie (4.3). Nech $\alpha, \beta \in P$ sú dva chromozómy z populácie P , pomocou operátora reprodukcie zostrojíme z týchto dvoch "rodičovských" chromozómov dva nové chromozómy "potomkov", formálne (pozri (4.3)) $(\alpha', \beta') = O_{repro}(\alpha, \beta)$. V prvom kroku zostrojíme pomocou kríženia dva nové chromozómy

$$(\hat{\alpha}, \hat{\beta}) = O_{cross}(\alpha, \beta) \quad (4.15)$$

V nasledujúcom kroku sa tieto dva nové chromozómy modifikujú mutáciou na výsledné chromozómy - potomkov

$$\alpha' = O_{mut}(\hat{\alpha}) \quad \text{a} \quad \beta' = O_{mut}(\hat{\beta}) \quad (4.16)$$

Implementácia reprodukčného operátora sa vyjadrí pomocou algoritmu 4.4.

```
procedure Reproduction(input: $\alpha, \beta$ ; output: $\alpha', \beta'$ );
begin Crossover( $\alpha, \beta, \hat{\alpha}, \hat{\beta}$ );
      Mutation_Bin( $\hat{\alpha}, \alpha'$ );
      Mutation_Bin( $\hat{\beta}, \beta'$ );
end;
```

Algoritmus 4.4. V prvom kroku sa aktivuje procedúra križenia (pozri algoritmus 4.3), výstupné parametre tejto procedúry $\hat{\alpha}$ a $\hat{\beta}$ sa modifikujú pomocou mutácie (pozri algoritmus 3.2).

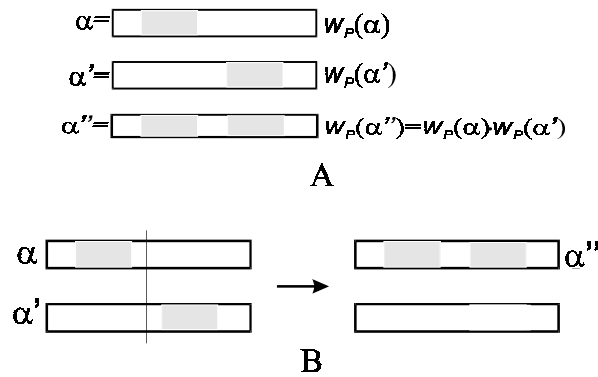
Križenie patrí medzi základné črty genetického algoritmu; ak ho odstránime, potom sa genetický algoritmus zredukuje na niečo blízke horolezeckému algoritmu. Používanie križenia odlišuje genetický algoritmus od ostatných stochastických evolučných algoritmov, ktoré v rámci populácie objektov - riešení - chromozómov tiež používajú reprodukciu založenú na ich fitness a aplikujú mutácie na jednotlivé objekty. Medzi odborníkmi zaoberajúcimi sa aplikáciami genetického algoritmu panuje presvedčenie, že efektívnosť týchto ostatných evolučných algoritmov je vo všeobecnosti menšia ako efektívnosť genetického algoritmu hlavne z toho dôvodu, že nepoužívajú križenie. Toto presvedčenie je čiastočne založené na osobných skúsenostiach s aplikáciami genetického algoritmu a na analógiách s evolúciou biologických objektov, ktoré vyústili do formulácie slávnej Hollandovej vety [3] tvoriacej teoretický základ genetického algoritmu.

Pohlavná reprodukcia je v prírode všeobecne využívaná hlavne medzi vyspelejšími biologickými druhmi našej planéty. V porovnaní s reprodukčnými technikami, ktoré vyžadujú len jedného rodiča, pohlavná reprodukcia je neobyčajne zložitý proces produkcie potomkov. Druh využívajúci pohlavnú reprodukciu musí byť podstatne zložitejší ako druh, ktorý ju nevyužíva. Títo jedinci sa musia odlišovať určitými podstatnými znakmi a musia tiež vynaložiť určitý čas a energiu, aby sa vzájomne stretli k reprodukčnému procesu. Pretože pohlavná reprodukcia jednoznačne zvíťazila v aréne prirodzeného výberu, musia existovať vážne dôvody, prečo tento komplikovaný spôsob reprodukcie patrí medzi základné črty, ktoré odlišujú nižšie druhy od vyšších druhov.

Jeden z hlavných dôvodov (podporený dôkladnou matematickou analýzou) výhodnosti križenia je fakt, že pohlavná reprodukcia umožňuje rýchlu výmenu výhodných vlastností, ktoré podstatne zvyšujú fitness jedincov - potomkov, čo je len veľmi ťažko dosiahnuteľné pomocou mutácií.

Vykonáme abstrakciu tohto problému (pozri obr. 4.9). Nech α a α' sú dva chromozómy a každý z nich obsahuje nejaký podreťazec, ktorý podstatne zvyšuje fitness chromozómu. Nech sú tieto reťazce na takých pozíciách, že keby boli na jednom chromozóme, tak by sa ani čiastočne neprekrývali. Nech pravdepodobnosť výskytu chromozómu s jedným z týchto dvoch výhodných podreťazcov v populácii je označená $w_P(\alpha)$ resp. $w_P(\alpha')$. Potom pravdepodobnosť $w_P(\alpha'')$ výskytu chromozómu α'' , ktorý súčasne obsahuje podreťazce z chromozómov α a α' , sa približne rovná súčinu týchto pravdepodobností, $w_P(\alpha'') = w_P(\alpha) \times w_P(\alpha')$. Položme *ad-hoc* $w_P(\alpha) = 10^{-6}$ a $w_P(\alpha') = 10^{-6}$, potom $w_P(\alpha'') = 10^{-12}$. Môžeme teda konštatovať, že súčasný výskyt podreťazcov z chromozómov α a α' v jednom

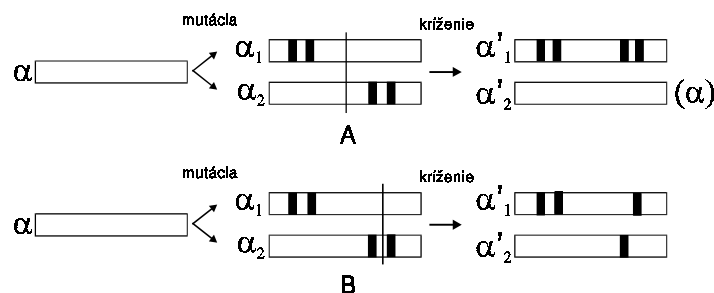
chromozóme α'' len v dôsledku mutácií je vysoko nepravdepodobný jav. Situácia sa však dramaticky zmení, ak pripustíme kríženie medzi chromozómami. Výskyt chromozómu α'' v populácii sa stáva vysoko pravdepodobným javom za predpokladu, že chromozómy α a α' sa v populácii už vyskytujú. Pohlavné kríženie (alebo v abstraktnej rovine kríženie medzi chromozómami) je efektívny spôsob ako prekonať pravdepodobnostnú bariéru tvorby chromozómov s vysokým informačným obsahom, a teda aj s vysokým fitness. Ak nezapojíme do prirodzeného výberu pohlavné kríženie, rýchlosť evolúcie sa podstatne zníži tak, že vznik zložitejších druhov v prírode je len málo pravdepodobný.



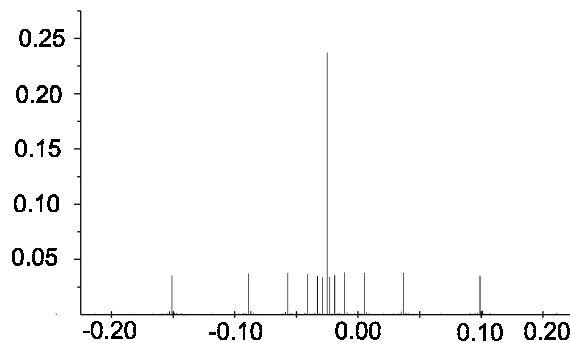
Obrázok 4.9. Schematické znázornenie významu kríženia v genetickom algoritme. Diagram A znázorňuje chromozómy α a α' , ktoré obsahujú dva rôzne neprekrývajúce sa podreťazce, tieto podreťazce zvyšujú fitness chromozómov. Pravdepodobnosť výskytu týchto chromozómov je označená $w_P(\alpha)$ a $w_P(\alpha')$. Chromozóm α'' obsahuje obidva podreťazce, pravdepodobnosť jeho výskytu je zhruba rovná súčinu pravdepodobností výskytu chromozómov α a α' , $w_P(\alpha'') = w_P(\alpha) \cdot w_P(\alpha')$. Diagram B ukazuje, že použitím kríženia môžeme zostrojiť chromozóm α'' , pravdepodobnosť tejto konštrukcie je podstatne vyššia ako pravdepodobnosť súčasného výskytu dvoch podreťazcov $w_P(\alpha'')$.

Vyššie uvedené úvahy o význame kríženia v genetickom algoritme budeme konkretizovať pre chromozómy reprezentované binárnymi reťazcami. Vykonajme nasledujúci jednoduchý výpočtový experiment: Nech α je binárny reťazec - templát, špeciálnou operáciou mutácie $O_{mut}^{(i)}$ tento chromozóm pretransformujeme na dva nové chromozómy, $\alpha_1 = O_{mut}^{(i)}(\alpha)$ a $\alpha_2 = O_{mut}^{(i)}(\alpha)$. Operátor mutácie $O_{mut}^{(i)}$ mutuje - preklopí práve i bitov v reťazci α . Stochastičnosť tohto operátora spočíva v tom, že bity, v ktorých dochádza k preklopeniu, sa náhodne vybrali, ich počet je určený parametrom i . Aplikovaním operácie kríženia na tieto dva chromozómy dostaneme $(\alpha'_1, \alpha'_2) = O_{cross}(\alpha_1, \alpha_2)$. Definujme si nasledujúce dva výrazy: $\Delta_1 = \text{real}(\alpha) - \text{real}(\alpha'_1)$ a $\Delta_2 = \text{real}(\alpha) - \text{real}(\alpha'_2)$, ktoré vyjadrujú rozdiel medzi reálnou hodnotou templátu α a chromozómu vzniklého mutáciou a krížením. V prípade $\Delta_i = 0$ (pre $i=1,2$) operácia kríženia "opravila" mutáciou zmenený chromozóm α_i na pôvodný templát. V opačnom prípade, ak $\Delta_i \neq 0$, kríženie produkuje nové chromozómy, ktoré sú rôzne od templátu (pozri obr. 4.10). Na obr. 4.11 je uvedený graf znázorňujúci distribúciu hodnôt diferencií Δ pre mutáciu špecifikovanú tak, že zmení v reťazci práve jeden bit. Na tomto grafe vidíme významné maximum odpovedajúce $\Delta=0$, t.j. kríženie opravuje mutované reťazce na pôvodný templát. Ako ukazujú naše modelové výpočty, distribúcia

pravdepodobnosti tohto maxima pre $\Delta=0$ rýchlo klesá s rastúcim počtom zmenených bitov, pre štyri zmenené bity je táto pravdepodobnosť takmer nulová. Ako interpretovať tieto pozorovania? Operácia kríženia je veľmi efektívnou vtedy, keď do procesu kríženia vstupujú chromozómy, ktoré majú veľké neprekrývajúce sa časti z "cieľového" chromozómu - templátu. V tomto prípade operácia kríženia produkuje nové chromozómy, ktoré tieto rôzne časti integrujú do jedného chromozómu - potomka veľmi blízkeho (alebo totožného) s cieľovým templátom. V opačnom prípade, ak dvojica chromozómov vstupujúca do kríženia neobsahuje takéto časti, operácia kríženia pôsobí podobne ako mutácia, produkuje dvojicu chromozómov, ktorých fitness je v podstate podobný fitness chromozómov vstupujúcich do kríženia.



Obrázok 4.10. Dva rôzne prípady pôsobenia kríženia na dva chromozómy, ktoré vznikli mutáciou (v tomto prípade zmenili svoje hodnoty práve dva bity) templátu α . V prípade A bol bod kríženia náhodne vybraný tak, že krížením chromozómov sme dostali jedného potomka, ktorý je totožný s templátom α . V prípade B bol bod kríženia vybraný tak, že ani jeden z potomkov kríženia nie je totožný s templátom.

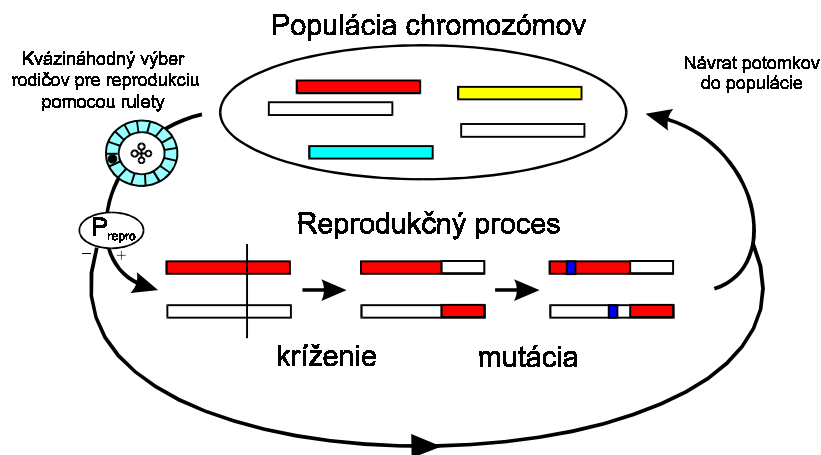


Obrázok 4.11. Distribúcia pravdepodobnosti výskytu rozdielov Δ medzi templátom (náhodne generovaným) dĺžky 10 a chromozómami z neho tvorenými mutáciou (jeden bit sa zmení) a potom skríženými, pričom reálne čísla priradené chromozómom sú z intervalu $(0,1)$. Distribúcia má významné maximum pre $\Delta=0$, t.j. kríženie významne "opravuje" zmutované chromozómy do tvaru pôvodného templátu.

Úloha 4.4. Napíšte program, pomocou ktorého budete schopní zreprodukovat' graf na obr. 4.11. Preštudujte rôzne dĺžky binárneho reťazca a rôzny počet zmenených bitov v binárnych reťazcoch α_1 a α_2 .

4.4 Implementácia genetického algoritmu [5]

Genetické algoritmy sa v súčasnosti využívajú v najrozličnejších vedecko-technických aplikáciách [8-13]. Konkrétnu implementáciu genetického algoritmu dostaneme zo všeobecného evolučného algoritmu (pozri algoritmus 4.1), ak je operátor reprodukcie O_{repro} špecifikovaný tak, že obsahuje dve časti: časť kríženia a časť mutácie (pozri algoritmus 4.5). Výber chromozómov do reprodukčného procesu je realizovaný pomocou rulety (pozri algoritmus 4.2). Pseudopascalovský kód genetického algoritmu je vyjadrený algoritmom 4.5 (pozri obr. 4.12).



Obrázok 4.12. Schematické znázornenie genetického algoritmu (pozri algoritmus 4.5). Pomocou rulety sa vyberie dvojica chromozómov, ktorá vstupuje do procesu reprodukcie s pravdepodobnosťou P_{repro} , pričom reprodukčný proces sa skladá z kríženia a mutácie.

Problém zastavenia genetických algoritmov v týchto implementáciách sa zatiaľ rieši tak, že počet generácií je zhora ohraničený číslom t_{max} , čo je však veľmi zjednodušený pohľad na tento dôležitý aspekt genetických algoritmov. Prediskutujeme dve metódy ukončenia genetického algoritmu.

Prvá metóda spočíva v tom, že v každom iteračnom kroku - generácii nájdeme v populácii chromozóm s najväčšou hodnotou fitness, označíme ho $\tilde{\alpha}_{opt}$. Nech $0 \leq w(\tilde{\alpha}_{opt}) \leq 1$ je podiel chromozómov populácie, ktoré sa rovnajú $\tilde{\alpha}_{opt}$. Potom sa genetický algoritmus môže ukončiť, ak podiel $w(\tilde{\alpha}_{opt}) > w_{max}$, kde w_{max} je prahová hodnota podielu (obvykle 0.8-0.95). To znamená, že populácia obsahuje $100 \times w(\tilde{\alpha}_{opt})$ % chromozómov, ktoré sú identické s chromozómom $\tilde{\alpha}_{opt}$, čiže je už malá pravdepodobnosť, že v rámci ďalších generácií genetického algoritmu vzniknú chromozómy s ešte väčšou fitness, ako má chromozóm $\tilde{\alpha}_{opt}$.

Druhý prístup k zastaveniu genetického algoritmu je založený na výpočte analógu pravdepodobnostného vektora $\mathbf{w} = (w_1, w_2, \dots, w_k)$ známeho z horolezeckého algoritmu s učením (pozri kapitolu 3.2). Nech populácia P v i -tom kroku (generácii) obsahuje chromozómy $P = (\alpha_1, \alpha_2, \dots, \alpha_p)$, kde $\alpha_i = (\alpha_1^{(i)}, \alpha_2^{(i)}, \dots, \alpha_k^{(i)})$. Potom sú zložky pravdepodobnostného vektora určené takto

$$w_j = \frac{1}{p} \sum_{i=1}^p \alpha_j^{(i)} \quad (\text{pre } j=1,2,\dots,k) \quad (4.17)$$

ktoré vyjadrujú "pravdepodobnosť" výskytu "1" v j -tej polohe vo všetkých chromozómoch populácie P . V prípade, že $w_j=0$ (alebo $w_j=1$), potom $100 \times w_j$ % chromozómov z populácie P má v j -tej polohe 0 (1). Parameter usporiadania (pozri rovnicu (3.10)) definovaný pre takto určený pravdepodobnostný vektor má tvar

$$\chi(w) = \frac{4}{k} \sum_{j=1}^k (w_j - 0.5)^2 \quad (4.18)$$

Tento parameter s funkčnými hodnotami z intervalu $[0,1]$ má tú vlastnosť, že pre náhodne generovanú populáciu P (prvé iteračné kroky genetického algoritmu) sa blíži k nule (je malé kladné číslo). Tak ako evolúcia populácie pokračuje, hodnota parametru rastie a asymptoticky sa blíži k jednotke. Preto môžeme zaviesť prahovú hodnotu parametra usporiadania χ_{crit} ; potom ak platí $\chi(w) > \chi_{crit}$, genetický algoritmus sa ukončí. V tomto prípade populácia obsahuje chromozómy, ktoré v každej polohe majú s veľkou frekvenciou buď "0" alebo "1", takže môžeme povedať, že populácia s veľkou prevahou obsahuje rovnaké chromozómy, a preto je malá pravdepodobnosť, že v rámci pokračovania genetického algoritmu dostaneme nové lepšie riešenie.

```

procedure Genetic_Algorithm(input  $t_{max}$ ; output  $\alpha_{opt}$ );
begin  $t:=0$ ; stop_criterion:=false;
       $P:=\{\text{náhodne generovaná populácia chromozómov}\}$ ;
      while ( $t < t_{max}$ ) and (not stop_criterion) do
        begin  $t:=t+1$ ;  $Q:=\emptyset$ ;
              každý chromozóm je ohodnotený fitness;
              while  $|Q| < |P|$  do
                begin vyber ruletou dva chromozómy  $\alpha_1, \alpha_2 \in P$ ;
                      if random  $< P_{repro}$  then
                        Reproduction( $\alpha_1, \alpha_2, \alpha'_1, \alpha'_2$ ) else
                          begin  $\alpha'_1 := \alpha_1$ ;  $\alpha'_2 := \alpha_2$  end;
                         $Q := Q \cup \{\alpha'_1, \alpha'_2\}$ ;
                      end;  $P := Q$ ;
                      if konvergenčné kritérium je splnené
                        then stop_criterion:=true;
                end;
               $\alpha_{opt} := \text{najlepší chromozóm z } P$ ;
        end;
      end;

```

Algoritmus 4.5. Pseudopascalovská implementácia genetického algoritmu, ktorá vznikla z evolučného algoritmu 4.1 tak, že sa špecifikuje reprodukčný operátor O_{repro} algoritmom 4.5, t.j. obsahuje kríženie a mutáciu. Vonkajší cyklus **while** riadi výmenu generácií - epoch algoritmu, ktorých maximálny počet je určený vstupným parametrom t_{max} . Vnútny cyklus **while** generuje novú populáciu Q tak, že sa opakovane pomocou rulety vyhladajú dva chromozómy a tieto s pravdepodobnosťou P_{repro} vstupujú do reprodukčného procesu (v opačnom prípade sa vybrané chromozómy jednoducho skopírujú do novej populácie Q). Výstup z genetického algoritmu je α_{opt} , určený ako chromozóm s najväčším fitness z poslednej populácie. Z konštrukcie novej populácie Q vyplýva, že populácia P musí mať párny počet chromozómov. Genetický algoritmus obsahuje dve pravdepodobnosti, a to pravdepodobnosť reprodukcie P_{repro} a pravdepodobnosť 1-bitovej mutácie P_{mut} , tieto možno považovať spolu s kardinalitou populácie P za základné parametre algoritmu.

Úloha 4.5. Napíšte program pre genetický algoritmus podľa algoritmu 4.5, pričom účelová funkcia obsahuje n premenných z intervalu $[a,b]$ a tieto reálne premenné sú v binárnej reprezentácii aproximované reťazcami dĺžky k . V binárnej reprezentácii sa má používať tak štandardné, ako aj Grayovo kódovanie. Vykonaajte tieto štúdie:

(1) Účelová funkcia z úlohy 3.2 pre $n=1,2,3,4$ (pozri úlohu 3.2).

(2) Účelová funkcia z úlohy 3.3 pre $n=1,2,3,4$ (pozri úlohu 3.3).

Pre obe tieto funkcie porovnajte efektívnosť algoritmu tak pre Grayovo kódovanie, ako aj pre štandardné kódovanie binárnej reprezentácie reálnych čísel. Fitness chromozómov z populácie sa bude počítať vzorcami (4.11) resp. (4.12). Pokúste sa optimalizovať pravdepodobnosti P_{mut} a P_{repro} pre rôzne veľkosti populácie, spôsoby kódovania a dĺžku k binárnej reprezentácie.

Úloha 4.6. Implementujte do genetického algoritmu z úlohy 4.5 kritéria pre zastavenie genetického algoritmu z kapitoly 4.4. Vykreslite priebeh pravdepodobností w_i definovaných vzťahom (4.17) vzhľadom na počet epoch (iteračných krokov) genetického algoritmu. Výsledky porovnajte s obr. 3.5.

4.5 Teória genetického algoritmu

Väčšina teoretických prác venovaná genetickému algoritmu [3,15] je zameraná na význam tzv. schém, ktoré určujú celé triedy podobných binárnych reťazcov dĺžky k (chromozómov) z množiny $\{0,1\}^k$. Schéma je definovaná ako reťazec dĺžky k zostrojený nad množinou symbolov $\{0,1,*\}$, formálne schéma $\sigma \in \{0,1,*\}^k$, kde "*" je tzv. voľný symbol (wildcard). Hovoríme, že reťazec $\alpha \in \{0,1\}^k$ je príkladom schémy $\sigma \in \{0,1,*\}^k$, formálne $\alpha \in \sigma$, ak v každej polohe schémy σ s iným ako voľným symbolom je hodnota znaku reťazca α totožná so znakom v σ . Formálne

$$\alpha \in \sigma \Leftrightarrow (\forall i \in \{1,2,\dots,k\}: \sigma_i \in \{0,1\} \Rightarrow \sigma_i = \alpha_i) \quad (4.19)$$

Zavedený pojem môžeme jednoducho ilustrovať schémou $\sigma=(11**0**)$; príkladom uvedenej schémy je binárny reťazec dĺžky $\alpha=(1101011)$, prvé dva voľné symboly sú nahradené podreťazcom 01, zatiaľ čo posledné dva voľné symboly podreťazcom 11.

Množina $I(\sigma)$ obsahuje všetky možné príklady α schémy σ ,

$$I(\sigma) = \{\alpha; \alpha \in \sigma\} \quad (4.20)$$

Pre schému $\sigma=(11**0**)$ obsahuje množina $I(\sigma)$ 16 binárnych reťazcov, ktoré môžu byť zostrojené tak, že voľné symboly "*" systematicky nahradíme "0" alebo "1",

$$I(\sigma) = \{(1100000), (1100001), (1100010), \dots, (1111011)\} \quad (4.21)$$

Rád schémy σ , označený $o(\sigma)$, opisuje počet symbolov "0" alebo "1" v σ

$$o(\sigma) = \left| \left\{ i; \sigma_i \in \{0,1\} \right\} \right| \quad (4.22)$$

tento vzorec určujúci rád schémy σ má jednoduchú interpretáciu, rád je definovaný ako kardinalita množiny indexov zložiek schémy σ , ktoré nie sú rovné voľnému symbolu. Pre $\sigma=(11**0**)$ platí $o(\sigma)=3$, pretože schéma obsahuje len tri "nevoľné" symboly. Dĺžka Δ schémy σ sa určí ako rozdiel medzi indexom posledného binárneho symbolu v σ a prvým binárnym symbolom v σ

$$\Delta(\sigma) = \max\{i; \sigma_i \in \{0,1\}\} - \min\{i; \sigma_i \in \{0,1\}\} \quad (4.23)$$

Pre $\sigma=(11**0**)$ platí $\Delta(\sigma)=5-1=4$. Veličiny rád a dĺžka schémy budú užitočné pre náš výpočet pravdepodobnosti prežitia schémy σ v populácii P pôsobením mutácie alebo kríženia.

Pravdepodobnosť prežitia schémy σ po aplikovaní mutácie (určenej s pravdepodobnosťou P_{mut} jednobitovej mutácie) je určená vzťahom

$$(1-P_{mut})^{o(\sigma)} \quad (4.24a)$$

kde pravdepodobnosť prežitia jedného bitu je $(1-P_{mut})$. Pretože schéma σ obsahuje práve $o(\sigma)$ binárnych symbolov ($k-o(\sigma)$ voľných symbolov), potom elementárna pravdepodobnosť prežitia jedného bitu musí byť umocnená na exponent $o(\sigma)$, čím sme dostali (4.24a). V mnohých praktických aplikáciách genetického algoritmu je pravdepodobnosť P_{mut} veľmi malá, preto (4.24a) môžeme zjednodušiť takto

$$1-o(\sigma)P_{mut} \quad (4.24b)$$

Výpočet pravdepodobnosti prežitia schémy σ v populácii P po aplikácii kríženia je podstatne zložitejší ako predošlý postup. Schéma σ zaniká aplikovaním kríženia s pravdepodobnosťou $P_{cross} \Delta(\sigma)/(k-1)$. Okrem toho, každé kríženie s chromozómom $\alpha \in P$, ktorý je príkladom schémy σ , $\alpha \in \sigma$, automaticky garantuje zachovanie tejto schémy. Nech $N(\sigma)=|I(\sigma) \cap P|$ popisuje počet príkladov schémy σ v populácii P , potom len podiel $1-N(\sigma)/|P|$ z populácie P môže spôsobovať deštrukciu schémy σ krížením. Minimálna pravdepodobnosť prežitia schémy σ v populácii P je určená vzťahom

$$1-P_{cross} \frac{\Delta(\sigma)}{k-1} \left(1-\frac{N(\sigma)}{|P|}\right) \quad (4.25a)$$

Tento vzorec sa dá podstatne zjednodušiť, ak sa zanedbá súčinový člen, čím dostaneme

$$1-P_{cross} \frac{\Delta(\sigma)}{k-1} \quad (4.25b)$$

Upriamime svoju pozornosť na problém výberu chromozómov vstupujúcich do reprodukcie. Zavedieme nasledujúce dve veličiny, *strednú fitness schémy σ*

$$F(\sigma) = \frac{1}{N(\sigma)} \sum_{\alpha \in I(\sigma) \cap P} F(\alpha) \quad (4.26)$$

a strednú fitness chromozómov z populácie P

$$\bar{F} = \frac{1}{|P|} \sum_{\alpha \in P} F(\alpha) \quad (4.27)$$

Pri prechode z generácie t na nasledujúcu generáciu $t+1$ v genetickom algoritme, ktorý obsahuje len výber chromozómov podľa rulety a reprodukčný proces sa úplne ignoruje, nastane nasledujúca zmena počtu výskytu schémy σ v populácii

$$N_{t+1}(\sigma) = N_t(\sigma) \frac{F_t(\sigma)}{\bar{F}_t} \quad (4.28)$$

kde dolný index popisuje index generácie populácie, pre ktorú sa dané veličiny počítali.

Nech σ je schéma, ktorej stredná hodnota fitness vzhľadom na strednú hodnotu chromozómov je určená vzťahom

$$F_t(\sigma) = \bar{F}_t \times (1+c) \quad (4.29)$$

kde konštanta c je malé kladné alebo záporné číslo. Dosadením tohto vzťahu do (4.28) a opakovaným použitím výsledného vzorca $N_{t+1}(\sigma) = N_t(\sigma) \times (1 + c)$ t -krát pre počiatočný stav $t=0$ dostaneme "idealizovanú" rovnicu

$$F_t(\sigma) = F_0(\sigma) \times (1 + c)^t \quad (4.30)$$

To znamená, že pre proporcionálny výber (realizovaný ruletou) stredná hodnota síl schémy σ *exponenciálne rastie* (klesá) v priebehu mnohých generácií genetického algoritmu pre $1+c>1$ (resp. $1+c<1$).

Kombinovaním rovnice (4.28), ktorá vyjadruje zmenu výskytu schémy σ v dôsledku proporcionálneho výberu, s pravdepodobnosťami prežitia schémy σ v dôsledku mutácie a kríženia dostaneme výsledný vzťah pre odhad počtu príkladov schémy pre generáciu $t+1$ populácie P za predpokladu, že reprodukčný proces obsahuje mutáciu, kríženie a výber pomocou rulety

$$N_{t+1}(\sigma) \approx N_t(\sigma) \frac{F_t(\sigma)}{\bar{F}_t} \left[1 - P_{cross} \frac{\Delta_t(\sigma)}{k-1} \times \left(1 - \frac{N_t(\sigma)}{|P|} \right) \right] \times (1 - P_{mut})^{o_t(\sigma)} \quad (4.31a)$$

Keď sa zanedbajú "krížové" súčiny v hranatej zátvorke, zjednodušená verzia má tvar

$$N_{t+1}(\sigma) \approx N_t(\sigma) \frac{F_t(\sigma)}{\bar{F}_t} \left[1 - P_{cross} \frac{\Delta_t(\sigma)}{k-1} - P_{mut} o_t(\sigma) \right] \quad (4.31b)$$

Veta (Schema Theorem [3]). Nech pre populáciu P_t platí, že $N_t(\sigma)$ je počet chromozómov, ktoré obsahujú schému σ , $F_t(\sigma)$ je stredná hodnota fitness chromozómov obsahujúcich schému σ a $\bar{F}_t$ je stredná hodnota fitness všetkých chromozómov. Potom očakávaný počet chromozómov v nasledujúcej populácii, ktoré obsahujú schému σ , je zdola ohraničený nerovnosťou

$$N_{t+1}(\sigma) \geq N_t(\sigma) \frac{F_t(\sigma)}{\bar{F}_t} (1 - P_{zánik}) \quad (4.32)$$

kde $P_{zánik}$ je pravdepodobnosť zániku schémy v dôsledku mutácie a kríženia (pozri rovnicu (4.33)).

Táto veta hovorí, že ak platí $F_t(\sigma) > \bar{F}_t$ ($F_t(\sigma) < \bar{F}_t$), potom schéma σ sa vyskytuje v nasledujúcej generácii v chromozómoch s rastúcou (klesajúcou) frekvenciou. Tieto jednoduché závery sú založené na predpoklade, že pravdepodobnosť zániku

$$P_{zánik} = P_{cross} \frac{\Delta_t(\sigma)}{k-1} - P_{mut} o_t(\sigma) \quad (4.33)$$

je malé číslo. Toto je obvykle splnené pre krátke schémy (s malým $\Delta(\sigma)$), nazývané stavebné kamene (building blocks). Možno konštatovať, že táto veta tvorí jeden z teoretických základov genetického algoritmu.

Veta uvedená vyššie predstavuje jeden z hlavných teoretických výsledkov genetického algoritmu, ktorý dosiahol Holland v r. 1975 [3]. Naznačuje mechanizmus práce tohto algoritmu predovšetkým v prvých iteračných krokoch, kde schémy - stavebné kamene hrajú dôležitú úlohu. Teoretické aspekty genetického algoritmu, ktoré súvisia s jeho schopnosťou

nájsť globálne minimum účelovej funkcie aproximovanej reálnymi hodnotami z ortogonálnej mriežky bodov (pozri obr. 2.8), sa študujú až v poslednej dobe [16-18]. Uvedieme vetu, ktorá špecifikuje, za akých podmienok je genetický algoritmus potenciálne schopný poskytnúť globálne minimum s jednotkovou pravdepodobnosťou.

Veta. Nech je v genetickom algoritme postupnosť populácií $P_0, P_1, \dots$ monotónna, t.j. pre každú generáciu t platí

$$\max_{\alpha \in P_{t+1}} F(\alpha) \geq \max_{\alpha \in P_t} F(\alpha) \quad (4.34)$$

potom genetický algoritmus asymptoticky pre $t \rightarrow \infty$ dosiahne globálne optimum

$$\alpha_{opt} = \lim_{t \rightarrow \infty} \left(\arg \min_{\alpha \in P_t} F(\alpha) \right) \quad (4.35)$$

Ako dosiahnuť monotónnosť genetického algoritmu? Toto sa môže jednoducho dosiahnuť pomocou tzv. *elitizmu*, ktorý je založený na jednoduchej myšlienke, že najlepšie riešenie z populácie P_t je automaticky prenesené aj do nasledujúcej populácie P_{t+1} . Ináč povedané, najlepšie riešenie v populácii P_t nemá možnosť byť náhodne vynechané výberom realizovaným ruletou.

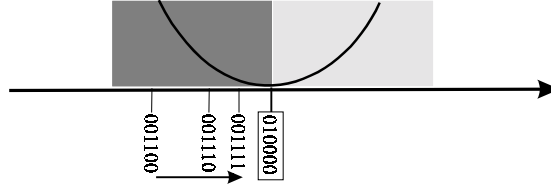
4.6 Hammingova bariéra

Uvažujme genetický algoritmus v ktorom sú reálne premenné účelovej funkcie binárne kódované štandardným spôsobom (pozri kapitolu 2.2). Dva chromozómy $\alpha=(00100\dots0)$ a $\beta=(000111\dots1)$ reprezentujú dve susedné celé čísla (ktorým sa priradia dve "susedné" reálne premenné). Čo je zaujímavé na týchto chromozómoch, je skutočnosť, že aj keď reprezentujú dve "susedné" čísla, ich Hammingova vzdialenosť (t.j. počet rôznych bitových premenných) je veľká (v limitnom prípade sa môže rovnať až dĺžke binárneho reťazca). Predpokladajme, že optimálne riešenie odpovedá binárnemu reťazcu $\alpha=(00100\dots0)$ a že populácia obsahuje chromozómy, ktoré sú blízke alebo totožné s binárnym reťazcom $\beta=(000111\dots1)$. Aká môže byť v tomto prípade evolúcia chromozómov populácie? Chromozómy populácie $(00011\dots\dots)$ sa vyvíjajú smerom k optimálnemu riešeniu tak, že po určitom počte iteračných krokov bude populácia obsahovať v prevažnej miere chromozómy $\beta=(000111\dots1)$. Pravdepodobnosť zmeny $(00011\dots\dots)$ mutáciou na $(00100\dots\dots)$ je veľmi malá, preto sa v populácii budú s najväčšou pravdepodobnosťou vyskytovať len chromozómy odpovedajúce schéme $(00011\dots\dots)$. To znamená, že evolúcia populácie chromozómov, ktorá bola inicializovaná chromozómami typu $(00011\dots\dots)$, má len veľmi malú možnosť byť smerovaná do výslednej populácie, ktorá by obsahovala optimálne riešenie $\alpha=(00100\dots0)$, a s najväčšou pravdepodobnosťou skončí pri riešení $\beta=(000111\dots1)$. Táto skutočnosť sa v genetickom algoritme nazýva Hammingova bariéra a predstavuje vážne ohrozenie aplikovateľnosti genetického algoritmu za predpokladu, že sa používa štandardné kódovanie binárnych reťazcov.

Ako sa vyhnúť Hammingovej bariére v genetickom algoritme? Tento problém sa môže riešiť dvoma prístupmi. Prvý prístup je založený na použití ďalšieho operátora transformácie

chromozómov, a to tzv. *inverzného operátora*. Tento operátor zmení binárny reťazec $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_k)$ na iný binárny reťazec $\beta = (\beta_1, \beta_2, \dots, \beta_k)$

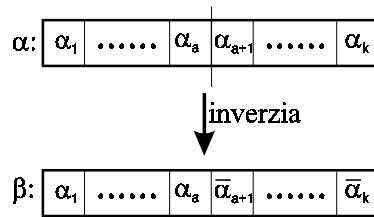
$$\beta = O_{inv}(\alpha) \quad (4.36)$$



Obrázok 4.13. Znáznornenie evolúcie riešenia typu 0011** k presnému riešeniu 010000. V priebehu evolúcie genetického algoritmu bude populácia obsahovať stále viac a viac chromozómov 001111, ktoré sú blízke optimálnemu riešeniu 010000.

Táto operácia je definovaná tak, že pre náhodne vygenerované celé číslo $1 < a < k$ (bod inverzie) sú zložky binárneho vektora pre indexy $i > a$ zamenené za ich komplementy (pozri obr. 4.14 a algoritmus 4.6)

$$\beta_i = \begin{cases} \alpha_i & (\text{pre } i \in [1, a]) \\ 1 - \alpha_i & (\text{pre } i \in [a+1, k]) \end{cases} \quad (4.37)$$



Obrázok 4.14. Schematické znázornenie operácie inverzie. Pre náhodne vybraný bod inverzie a vykonáme inverziu binárneho vektora α tak, že od bodu $a+1$ až do konca binárneho vektora nahradíme jeho zložky komplementami, $\bar{\alpha}_i = 1 - \alpha_i$.

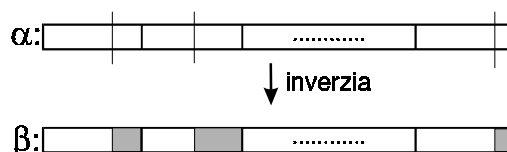
```

procedure Inversion(input: $\alpha$ ; output: $\beta$ );
begin if random <  $P_{inv}$  then
    begin  $a := 1 + \text{random}(k-1)$ ;
        for  $i := 1$  to  $a$  do  $\beta_i := \alpha_i$ ;
        for  $i := a+1$  to  $k$  do  $\beta_i := 1 - \alpha_i$ ;
    end else for  $i := 1$  to  $k$  do  $\beta_i := \alpha_i$ ;
end;

```

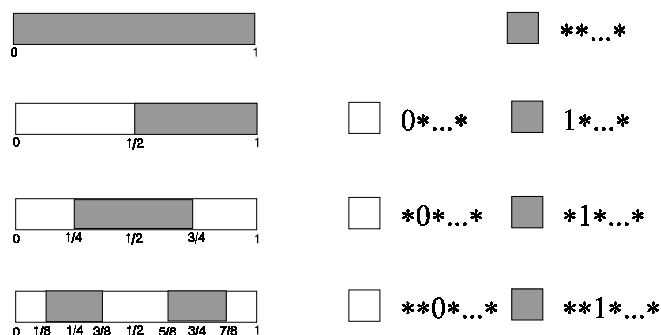
Algoritmus 4.6. Pseudopascalovská implementácia inverzie binárneho vektora dĺžky k . Pravdepodobnosť P_{inv} (malé kladné číslo, napr. $P_{inv} = 0.01-0.001$) určuje, či sa vykoná inverzia alebo nie. Funkcia random($k-1$) generuje náhodné celé čísla z intervalu $[0, k-2]$, random je náhodne generované číslo z intervalu $[0, 1)$.

Stochastický operátor inverzie odstraňuje vznikajúce Hammingove bariéry a dáva možnosť genetickému algoritmu evolúciou zostrojiť populáciu, ktorá obsahuje chromozómy zodpovedajúce optimálnemu riešeniu. Kde implementovať operátor inverzie v genetickom algoritme? Najjednoduchšie je ho prezentovať ako časť reprodukčného operátora. Algoritmus 4.4 bude rozšírený o dve operácie inverzie, ktoré sa vykonajú s pravdepodobnosťou P_{inv} . V prípade, že chromozóm binárne reprezentuje n reálnych premenných účelovej funkcie, pričom každá premenná je kódovaná binárnym reťazcom dĺžky k , operácia inverzie sa vykoná zvlášť pre každý podreťazec odpovedajúci jednotlivej premennej účelovej funkcie (pozri obr. 4.15).



Obrázok 4.15. Realizácia operácie inverzie pre binárny vektor reprezentujúci n reálnych premenných, pre každý podreťazec sa vykoná operácie inverzie zvlášť.

Druhý prístup na odstránenie Hammingovej bariéry je založený na použití Grayovho kódu pre binárnu reprezentáciu reálnych premenných (pozri kapitolu 2.2). Delenie jednotkového intervalu $[0,1]$ na podintervaly v rámci Grayovho kódovania binárných reťazcov je znázornené na obr. 4.16 a obr. 4.17. V tomto prípade je dôležité, aby susedné intervaly boli kódované binárnymi reťazcami, ktoré nie sú diametrálne odlišné medzi sebou, ako to môže byť vtedy, keď je použité štandardné kódovanie. Pre lepšie pochopenie tejto myšlienky sa vráťme k obr. 4.13, kde je znázornený priebeh účelovej funkcie s globálnym minimom v bode $1/4$, ktorý je binárne reprezentovaný 0100 . V štandardnom kódovaní sú intervaly $[1/8, 1/4]$ a $[1/4, 3/8]$ reprezentované reťazcami $001\dots$ resp. $010\dots$, pričom optimálne riešenie má tvar $001111\dots$. Vidíme, že keď sa blížíme k optimálnemu riešeniu z pravého intervalu, potom v určitom momente musí nastať dramatická zmena zloženia binárneho reťazca, a to z $010000\dots$ na $001111\dots$. Genetický algoritmus s normálne nastavenou pravdepodobnosťou mutácie ($P_{mut}=0.001 - 0.01$) nemá možnosť prekonať túto "Hammingovu bariéru". V prípade, ak použijeme Grayovo kódovanie, sa situácia podstatne zlepši v tom zmysle, že nedochádza k tak dramatickým zmenám zloženia chromozómov pri prechode z jedného podintervalu do druhého (pozri obr. 4.17).



Obrázok 4.16. Postupné delenie intervalu $[0,1]$ pre binárnu reprezentáciu pomocou Grayovho kódovania pre rôzne schémy. V pravej časti obrázku sú uvedené oblasti, ktoré odpovedajú uvedeným schémam.

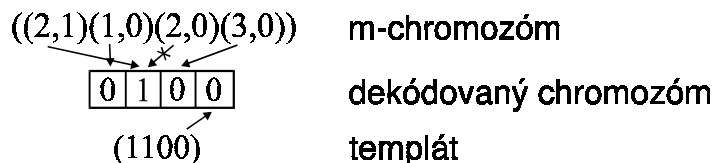
0	1/8	2/8	3/8	4/8	5/8	6/8	7/8	1
000..	001..	011..	010..	110..	111..	101..	100..	

Obrázok 4.17. Delenie intervalu [0,1] na osem podintervalov použitím Grayovho kódovania binárnych reťazcov.

Teraz musíme odpovedať na otázku, či je lepšie štandardné kódovanie alebo Grayovo kódovanie binárnych reťazcov v genetickom algoritme. Dať odpoveď na túto otázku nie je jednoduché. V štandardnej reprezentácii existuje rýchla transformácia binárneho reťazca na reálne čísla, avšak za cenu potenciálnej existencie Hammingových bariér. V prvej časti tejto podkapitoly sme ukázali, že tento problém štandardnej reprezentácie možno odstrániť zavedením operátora inverzie do genetického algoritmu ako súčasti reprodukčného procesu. Grayova reprezentácia odstraňuje problém s Hammingovou bariérou, táto je jednoducho odstránená tým, že v tomto kódovaní sú susedné intervaly reálnych čísel kódované podobnými binárnymi reťazcami (na rozdiel od štandardného kódovania). Použitie Grayovho kódovania v genetickom algoritme naráža na problém numerickej náročnosti. Na rozdiel od štandardného kódovania Grayovo kódovanie pri konštrukcii reálnych čísel požaduje ešte medzitransformáciu binárneho reťazca do štandardného kódovania. Táto medzitransformácia nám môže podstatne predĺžiť čas CPU genetického algoritmu, pretože transformácia binárneho reťazca na reálne číslo patrí medzi najčastejšie volanú procedúru algoritmu, čiže už malé zvýšenie jej časovej náročnosti môže mať dramatický dôsledok na celkový čas CPU.

4.7 "Messy"-chromozómy (M-chromozómy)

V r. 1989 sa Goldberg so spolupracovníkmi [19,20] pokúšal prekonať niektoré problémy genetického algoritmu pri hľadaní globálneho minima tzv. *deceptívnych účelových funkcií* (multimodálne funkcie s mnohými minimami, ale len s jedným globálnym minimom, ktoré je oddelené od ostatných miním bariérou) pomocou "messy" genetického algoritmu. Hlavná myšlienka tejto modifikácie genetického algoritmu spočíva okrem iného v použití m-chromozómov, ktoré majú premenlivú dĺžku a ich zložky sú špecifikované súčasne indexom aj hodnotou. Na rozdiel od štandardných chromozómov, špecifikovaných ako binárne vektory fixnej dĺžky, m-chromozómy môžu mať premenlivú dĺžku a byť tak "preurčené", ako aj "podurčené". Táto flexibilita m-chromozómov obvykle podstatne zvyšuje efektivitu a robustnosť evolučných algoritmov, ktoré ich využívajú [19,20].



Obrázok 4.18. Dekódovanie m-chromozómu pomocou templátu. Poloha špecifikovaná indexom je ohodnotená binárnou hodnotou, ktorá sprevádza špecifikovaný index. Keď už daná poloha bola špecifikovaná v predchádzajúcom kroku dekodovania, potom sa aktuálna špecifikácia ignoruje. V prípade, že niektorá poloha nie je špecifikovaná m-chromozómom, potom sa pre špecifikáciu použije príslušná hodnota z templátu.

M-chromozóm premenlivej dĺžky je definovaný takto: Nech $Q = \{1, 2, \dots, k\} \times \{0, 1\}$ je množina obsahujúca usporiadané dvojice (μ, v) , kde $\mu \in \{1, 2, \dots, k\}$ je index a $v \in \{0, 1\}$ je

binárna hodnota. M-chromozóm dĺžky l je definovaný takto

$$\chi = ((\mu_1, v_1), (\mu_2, v_2), \dots, (\mu_l, v_l)) \in Q^l \quad (4.38)$$

Tento m-chromozóm je dekódovaný na jednoduchý binárny vektor dĺžky k nasledujúcim postupom (pozri obr. 4.18): ak ideme postupne zľava doprava, prvá dvojica (μ_1, v_1) špecifikuje hodnotu v_1 v polohe μ_1 , $\alpha_{\mu_1} = v_1$. Ak $\mu_1 \neq \mu_2$, druhá dvojica (μ_2, v_2) špecifikuje hodnotu v_2 v polohe μ_2 , $\alpha_{\mu_2} = v_2$. Vo všeobecnosti platí

$$\alpha_{\mu_i} = v_i \quad (\text{ak } \forall j \leq i-1: \alpha_j \neq \alpha_i) \quad (4.39)$$

To znamená, že problém preurčenia sa rieši pomocou pravidla, že určujúca je tá dvojica, ktorá sa vyskytuje prvá, ak ideme cez m-chromozóm zľava doprava. Problém podurčenia (t.j. niektorá poloha v zostrojenom binárnom vektore nie je špecifikovaná) sa rieši pomocou templátu

$$t = (t_1, t_2, \dots, t_k) \in \{0,1\}^k \quad (4.40)$$

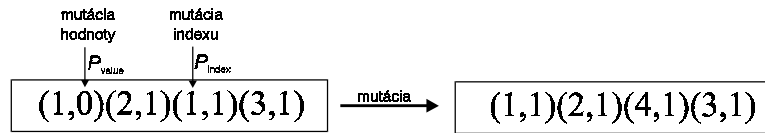
Ak p -ta poloha nie je špecifikovaná (4.40), t.j. $p \neq \alpha_i$ pre $\forall i \leq l$, potom táto poloha je určená p -tou polohou v templáte t

$$\alpha_p = t_p \quad (\text{pre každú nešpecifikovanú polohu}) \quad (4.41)$$

Problém dekódovania m-chromozómu χ pre daný templát t môže byť vo všeobecnosti chápaný ako zobrazenie $G_t: Q^l \rightarrow \{0,1\}^k$

$$\alpha = G_t(\chi) \quad (4.42)$$

Pre lepšie pochopenie vyššie uvedeného procesu dekódovania m-chromozómu pozri obr. 4.19, kde je uvedený jednoduchý ilustračný príklad.



Obrázok 4.19. Mutácia m-chromozómu má dve časti, mutáciu indexu a mutáciu binárnej hodnoty. Každá takáto mutácia sa vykonáva s pravdepodobnosťou P_{index} resp. P_{value} .

M-chromozóm χ je transformovaný na iný m-chromozóm χ' operáciou mutácie (pozri obr. 4.19)

$$\chi' = O_{mut}(\chi) \quad (4.43)$$

kde $\chi = ((\mu_1, v_1), (\mu_2, v_2), \dots, (\mu_l, v_l))$ a $\chi' = ((\mu'_1, v'_1), (\mu'_2, v'_2), \dots, (\mu'_l, v'_l))$. Stochastický operátor mutácie O_{mut} sa realizuje tak, že idúc zľava doprava postupne náhodne meníme usporiadané dvojice, t.j. indexy a binárne hodnoty

$$\mu'_i = \begin{cases} \hat{\mu}_i & (\text{ak } random < P_{index}) \\ \mu_i & (\text{ostatné prípady}) \end{cases} \quad (4.44a)$$

kde $1 \leq \hat{\mu}_i \leq k$ je náhodné celé číslo s rovnomernou distribúciou pravdepodobnosti, pričom $\hat{\mu}_i \neq \mu_i$. Pravdepodobnosť P_{index} určuje stochastičnosť zmeny indexovej časti m-chromozómu. Podobným spôsobom sa mení aj binárna časť m-chromozómu

$$v'_i = \begin{cases} 1 - v_i & (\text{ak } random < P_{value}) \\ v_i & (\text{ostatné prípady}) \end{cases} \quad (4.44b)$$

Pravdepodobnosť P_{value} určuje stochastičnosť zmeny binárnej časti m-chromozómu. Operáciu mutácie ilustruje príklad na obr. 4.19. Pseudopascalovská implementácia mutácie je uvedená v algoritme 4.7.

```

procedure M_Mutation(input:  $\chi$ ; output:  $\chi'$ );
begin for i:=1 to l do
    begin  $\mu'_i := \mu_i$ ;  $v'_i := v_i$ ;
        if random <  $P_{index}$  then  $\mu'_i := 1 + \text{random}(k)$ ;
        if random <  $P_{value}$  then  $v'_i := 1 - v_i$ ;
    end;
end;

```

Algoritmus 4.7. Procedúra mutácie m-chromozómu. Funkcia random(k) generuje náhodné celé čísla z intervalu [0,k-1], random je náhodne generované číslo z intervalu [0,1).

Operácia kríženia dvoch rodičovských m-chromozómov $\chi = ((\mu_1, v_1), (\mu_2, v_2), \dots, (\mu_{l_\chi}, v_{l_\chi}))$ a $\kappa = ((\xi_1, \zeta_1), (\xi_2, \zeta_2), \dots, (\xi_{l_\kappa}, \zeta_{l_\kappa}))$ poskytuje dva nové m-chromozómy - potomkov $\chi' = ((\mu'_1, v'_1), (\mu'_2, v'_2), \dots, (\mu'_{l'_\chi}, v'_{l'_\chi}))$ a $\kappa' = ((\xi'_1, \zeta'_1), (\xi'_2, \zeta'_2), \dots, (\xi'_{l'_\kappa}, \zeta'_{l'_\kappa}))$

$$(\chi', \kappa') = O_{cross}(\chi, \kappa) \quad (4.45)$$

Táto operácia sa realizuje tak, že v prvom kroku sa náhodne generuje "bod kríženia" $1 < a < \min\{l_\chi, l_\kappa\}$ a potom sa tvoria nové m-chromozómy tak, že až po "bod kríženia" a sa jednoducho okopírujú rodičovské chromozómy, za týmto bodom si nové m-chromozómy vymenia zvyšné časti (pozri obr. 4.20 a algoritmus 4.8).



Obrázok 4.20. Operácia kríženia dvoch m-chromozómov. Tieto si vymenia svoje časti nasledujúce za náhodne generovaným bodom kríženia. Operácia kríženia vo všeobecnosti mení dĺžku m-chromozómov.

Pomocou procedúr M_Mutation a M_Crossover (pozri algoritmy 4.7 a 4.8) môžeme jednoducho modifikovať algoritmus 4.4 (procedúra Reproduction) realizujúci reprodukciu dvoch rodičovských chromozómov na dva nové chromozómy - potomkov, stačí vymeniť procedúry pre mutáciu a kríženie za podobné procedúry pre m-chromozómy.

```

procedure M_Crossover(input: $\chi, \kappa$ ; output: $\chi', \kappa'$ );
begin a:=1+random(min( $l_\chi, l_\kappa$ )-1);
    for i:=1 to a do
        begin  $\mu'_i := \mu_i$ ;  $v'_i := v_i$ ;
             $\xi'_i := \xi_i$ ;  $\zeta'_i := \zeta_i$ ;
        end;
     $l_{\chi'} := l_\kappa$ ;  $l_{\kappa'} := l_\chi$ ;
    for i:=a+1 to  $l_{\chi'}$  do begin  $\mu'_i := \xi_i$ ;  $v'_i := \zeta_i$  end;
    for i:=a+1 to  $l_{\kappa'}$  do begin  $\xi'_i := \mu_i$ ;  $\zeta'_i := v_i$  end;
end;

```

Algoritmus 4.8. Procedúra križenia dvoch rodičovských m-chromozómov χ a κ na dva nové m-chromozómy - potomkov χ' a κ' .

```

procedure M_Genetic_Algorithm(input:timemax; output: $\alpha_{opt}$ );
begin time:=0;
    stop_criterion:=false;
    P:={náhodne generovaná populácia m-chromozómov};
    t:=náhodne generovaný binárny vektor - templát;
    while (time<timemax) and (non stop_criterion) do
        begin time:=time+1;
            Q:= $\emptyset$ ;
            každý m-chromozóm je ohodnotený fitness;
            while |Q|<|P| do
                begin vyber pomocou Roulette_Wheel dva
                    m-chromozómy  $\chi, \kappa \in P$ ;
                    if random<Prepro then
                        Reproduction( $\chi, \kappa, \chi', \kappa'$ ) else
                            begin  $\chi' := \chi$ ;  $\kappa' := \kappa$  end;
                        Q:=Q $\cup$ { $\chi', \kappa'$ };
                    end;
                P:=Q; t:=arg min $\chi \in P$  f(Gt( $\chi$ ));
                if konvergenčné kritériá sú splnené then
                    stop_criterion:=true;
            end;
             $\alpha_{opt} := t$ ;
        end;
end;

```

Algoritmus 4.9. Pseudopascalovská implementácia genetického algoritmu (pozri algoritmus 4.5) pre m-chromozómy. Algoritmus sa inicializuje náhodným generovaním populácie m-chromozómov a binárneho vektora - templátu. Templát sa obnovuje v každom iteračnom kroku pomocou binárnej interpretácie najlepšieho chromozómu z predchádzajúcej iterácie. Na záver algoritmu sa templát použije ako výstupné riešenie získané genetickým algoritmom s m-chromozómami.

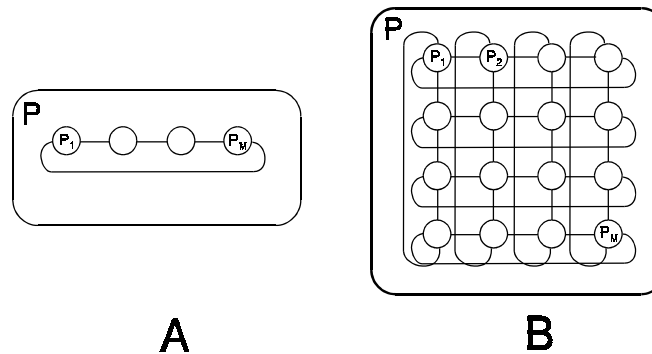
Základný problém, ktorý sa musí riešiť pri použití m-chromozómov v evolučných algoritmoch, je konštrukcia templátu t potrebného pre dekodovanie m-chromozómov do tvaru binárnych vektorov (pozri (4.40) a obr. 4.18). Pri inicializácii evolučného algoritmu sa templát t náhodne generuje ako binárny vektor, zatiaľ čo v neskorších fázach algoritmu je templát určený ako binárny vektor odpovedajúci najlepšiemu m-chromozómu z

predchádzajúcej iterácie. Na záver algoritmu sa templát použije ako najlepšie výstupné riešenie. Po týchto jednoduchých úvahách môžeme pristúpiť k modifikácii genetického algoritmu 4.5 pre m-chromozómy (pozri algoritmus 4.9).

Úloha 4.7. Implementujte genetický algoritmus podľa algoritmu 4.5, pričom namiesto chromozómov sa používajú "messy"-chromozómy z kapitoly 4.7. Porovnajte účinnosť takto modifikovaného "messy" genetického algoritmu s obyčajnou verziou genetického algoritmu.

4.8 Paralelný genetický algoritmus

Genetický algoritmus uvedený v kapitole 4.4 je sekvenčný algoritmus. V literatúre existuje mnoho verzií tzv. *paralelného genetického algoritmu* (PGA) [21], v ktorých je populácia rozdelená na podpopulácie, pričom evolúcia prebieha "nezávisle" nad týmito podpopuláciami, t.j. dochádza medzi nimi k občasnej stochastickej interakcii, v rámci ktorej si vymenia chromozómy. Obvykle sa tieto paralelné verzie uvádzajú (a tiež aj aplikujú) ako spôsob diverzifikácie a intenzifikácie genetického algoritmu.



Obrázok 4.21. Schematické znázornenie (disjunktného) rozkladu populácie P na podpopulácie $P_1, P_2, \dots, P_M$, $P = P_1 \cup P_2 \cup \dots \cup P_M$ a $P_i \cap P_j = \emptyset$, pre $i \neq j$. Diagram A znázorňuje rozklad, kde každá podpopulácia má práve dvoch susedov, "pravého" a "ľavého". Diagram B znázorňuje rozklad populácie, kde každá podpopulácia má práve štyroch susedov, "pravého" a "ľavého" a "dolného" a "horného". Zvolené "toroidovské" topológie 1D (diagram A) a 2D (diagram B) sa zvolili tak, aby neexistovali "okrajové" efekty, t.j. každá podpopulácia mala rovnaký počet susedov.

```

procedure One_generation_GA(input  $P_{ini}$ ; output  $P_{fin}$ );
begin  $P_{fin} := \emptyset$ ;
    while  $|P_{fin}| < |P_{ini}|$  do
        begin vyber ruletou dva chromozómy  $\alpha_1, \alpha_2 \in P_{ini}$ ;
            if random <  $P_{repro}$  then
                Reproduction( $\alpha_1, \alpha_2, \alpha'_1, \alpha'_2$ ) else
                    begin  $\alpha'_1 := \alpha_1$ ;  $\alpha'_2 := \alpha_2$  end;
                 $P_{fin} := P_{fin} \cup \{\alpha'_1, \alpha'_2\}$ ;
            end;
    end;

```

Algoritmus 4.10. Procedúra pre realizáciu jedného kroku genetického algoritmu (pozri algoritmus 4.11). Vstupná populácia P_{ini} sa zmení na výstupnú populáciu P_{fin} .

Nech je populácia P rozdelená na vzájomne disjunktné podpopulácie

$$P = P_1 \cup P_2 \cup \dots \cup P_M \quad (4.46)$$

Obvykle sa k tomuto rozkladu definuje ešte tzv. topológia, ktorá opisuje susedov danej podpopulácie. Formálne sa táto topológia rozkladu definuje pomocou neorientovaného súvislého grafu, ktorý obsahuje M vrcholov - podpopulácií. Ak sú dva vrcholy spojené hranou, potom hovoríme, že podpopulácie sú susedné (pozri obr. 4.21). V prípade, že neuvažujeme susednosť medzi podpopuláciami, budeme formálne predpokladať, že graf je úplný, t.j. každý vrchol susedí s ostatnými vrcholmi.

```

procedure Parallel_GA(input  $t_{\max}$ ; output  $\alpha_{\text{opt}}$ );
begin  $t:=0$ ; stop_criterion:=false;
  for  $i:=1$  to  $M$  do
     $P_i:=\{\text{náhodne generovaná podpopulácia chromozómov}\}$ ;
    while ( $t < t_{\max}$ ) and (not stop_criterion) do
      begin  $t:=t+1$ ;
        for  $i:=1$  to  $M$  do
          begin One_generation_GA( $P_i, P'_i$ );
             $P_i:=P'_i$ ;
          end;
          if random  $< P_{\text{exch}}$  then
            begin náhodne sú vybrané dve
              podpopulácie  $P_a$  a  $P_b$ ;
              exchange( $P_a, P_b$ );
            end;
          if konvergenčné kritéria sú splnené then
            stop_criterion:=true;
          end;
         $\alpha_{\text{opt}}:=\text{najlepší chromozóm všetkých podpopulácií}$ 
           $\{P_1, P_2, \dots, P_M\}$ ;
      end;
     $\alpha_{\text{opt}}:=\text{najlepší chromozóm všetkých podpopulácií}$ 
       $\{P_1, P_2, \dots, P_M\}$ ;
  end;

```

Algoritmus 4.11. Pseudopascalovská implementácia paralelného genetického algoritmu. Algoritmus sa inicializuje náhodnou generáciou podpopulácií $P_1, P_2, \dots, P_M$. Vonkajší cyklus **while** sa opakuje predpísaný počet generácií ($t_{\max}$) alebo pokiaľ boolovská premenná stop_criterion prestane mať hodnotu "false". Vo vnútri tohto cyklu sa zvlášť pre každú podpopuláciu realizuje proces jednej generácie, v ktorom sa aktuálna podpopulácia reprodukčným procesom (pozri algoritmus 4.10) zmení na novú podpopuláciu. Stochasticky sa realizuje (s pravdepodobnosťou P_{exch}) výmena chromozómov medzi dvoma náhodne vybranými podpopuláciami. V tomto bode existujú veľké odchýlky medzi rôznymi implementáciami paralelného genetického algoritmu. V prípade, že rozklad populácie na podpopulácie je "štruktúrovaný" (pozri obr. 4.21), potom sú podpopulácie P_a a P_b susedné. Aktivácia procedúry exchange(P_a, P_b) má za výsledok, že podpopulácie si vymenia chromozómy.

Jednoduchá verzia paralelného genetického algoritmu spočíva v tom, že pre každú podpopuláciu prebieha nezávislá evolúcia, pričom s určitou malou pravdepodobnosťou si podpopulácie vymenia chromozómy (pozri algoritmus 4.11). Výmena chromozómov je v algoritme 4.11 implementovaná pomocou procedúry exchange(P_a, P_b). Výsledok jej aktivácie nie je bližšie popísaný, ako príklad uvedieme túto možnú realizáciu: Podpopulácie P_a a P_b sa náhodne vybrali tak, aby boli susedné. V populácii P_a kvázi náhodne vyberieme najlepší chromozóm, potom tento vytesní v populácii P_b kvázi náhodne vybraný najslabší chromozóm. Akt výmeny chromozómov medzi dvoma podpopuláciami je úplne stochastický a uskutočňuje sa s pravdepodobnosťou P_{exch} .

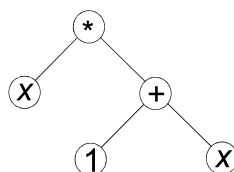
Literatúra

- [1] Ch. Darwin: *Darwinův původ druhů v ilustracích*. Panorama, Praha 1989.
- [2] S. Wright: Evolution of Mendelian Populations. *Genetics* **16** (1931) 97-157.
- [3] J.H. Holland: *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI 1975.
- [4] D.B. Fogel (ed.): *Evolutionary Computation: The Fossil Record*. IEEE, New York 1998.
- [5] D.E. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley, Reading, MA 1989.
- [6] M. Mitchell: *An introduction to Genetic Algorithms*. MIT Press, Boston, MA 1996.
- [7] N. Wirth: *Systematické programovanie*. Alfa, Bratislava 1981.
- [8] D. Dasgupta and Z. Michalewicz (eds.): *Evolutionary Algorithms in Engineering Applications*. Springer-Verlag, Berlin 1997.
- [9] M.A.C. Gill and A.Y. Zomaya: *Obstacle Avoidance in Multi-Robot Systems: Experiments in Parallel Genetic Algorithms*. World Scientific, Singapore 1998.
- [10] R.L. Haupt and S.E. Haupt: *Practical Genetic Algorithms*. John Wiley & Sons, New York 1998.
- [11] K.F. Man, K.S. Tang, S. Kwong, and W.A. Halang: *Genetic Algorithms for Control and Signal Processing*. Springer-Verlag, London 1997.
- [12] K.F. Man, K.S. Tang, S. Kwong, and W.A. Halang: *Genetic Algorithms: Concepts and Designs*. Springer-Verlag, London 1999.
- [13] C.L. Karr and K.M. Freeman (eds.): *Industrial Applications of Genetic Algorithms*. CRC Press, Boca Raton, FL 1999.
- [14] N. Wirth: *Algoritmy a štruktúry údajov*. Alfa, Bratislava 1988.
- [15] T. Bäck: *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York 1996.
- [16] R.F. Hartl: *A Global Convergence Proof for a Class of GAs*. Technische Universität Wien 1990.
- [17] A.E. Eiben, E.H.L. Aarts, and K.M. Van Hee: Global Convergence of GAs. An Infinite Markov Chain Analysis. In H.P. Schwefel and R. Männer (eds.): *Parallel Problem Solving from Nature I. Lecture Notes in Computer Science*, Vol. 496, Springer Verlag, Berlin 1991.
- [18] G. Rudolph: *Convergence Properties of Evolutionary Algorithms*. Verlag Kovac, Hamburg 1997.
- [19] D.E. Goldberg, B. Korb, and K. Deb: Messy Genetic Algorithms. Motivations, Analysis, and First Results. *Complex Systems* **3** (1989) 493-530.
- [20] D.E. Goldberg, K. Deb, and B. Korb: Messy Genetic Algorithms Revisited. Nonuniform Size and Scale. *Complex Systems* **4** (1990) 415-444.
- [21] E. Cantu-Paz: *A Survey of Parallel GAs*. M.Sc. Thesis, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign 1997. Dostupné na anonymnom ftp serveri <ftp-illigal.ge.uiuc.edu/pub/papers/Il ligals/97003.ps.Z>.

5. Genetické programovanie (GP)

5.1 Úvodné poznámky

Na prelome 80-tych a 90-tych rokov americký informatik John Koza [1-3] (Stanford University) navrhol originálnu modifikáciu genetického algoritmu, ktorú nazval *genetické programovanie*. V tomto prístupe sú chromozómy - znakové reťazce nahradené zložitejšími štruktúrami - funkciami. Čo sa rozumie pod pojmom "funkcia"? V najjednoduchšej verzii genetického programovania sa funkcie rovnajú výrazom obsahujúcim premenné, konštanty, základné aritmetické operácie a elementárne funkcie. Jednoduchá funkcia " $x*(1+x)$ " je reprezentovaná pomocou syntaktického stromu (parse tree)



kde horný vrchol (priradený operácii násobenia) sa nazýva koreň stromu. Interpretácia tohto syntaktického stromu je jednoduchá, pri postupe zdola-nahor, od koncových vrcholov ku koreňu stromu sa postupne vykonáva výpočet funkcie " $x*(1+x)$ ". Prvý problém, ktorý musíme riešiť v rámci genetického programovania, je spôsob vyhodnotenia chromozómu - stromu. Toto vyhodnotenie sa formálne chápe ako funkcia $y=t(x)$ priradujúca nezávislej premennej (vstup) x závislú premennú (výstup) y . V našom príklade je táto funkcia určená výrazom $t(x)=x*(1+x)$. Nech $A=\{(x_i, y_i); i=1, 2, \dots, p\}$ je tréningová množina obsahujúca p bodov. Naším cieľom je nájsť takú funkciu $t(x)$ (reprezentovanú syntaktickým stromom), ktorá minimalizuje rozdiel (jeho druhú mocninu alebo absolútnu hodnotu) vypočítaných a zadáných hodnôt y z tréningovej množiny. Ak je súčet týchto rozdielov cez všetky dvojice vypočítaných a zadáných hodnôt nulový, funkcia $t(x)$ presne vystihuje body z tréningovej množiny. Nájsť takú funkciu sa nám najskôr nepodarí, no môžeme sa pokúsiť nájsť takú funkciu, pre ktorú bude tento súčet minimálny. Hľadanú funkciu vyjadrujeme pomocou syntaktických stromov obsahujúcich predpísané typy vrcholov. Takto formulovaná úloha je veľmi blízka regresnej analýze, kde pre danú tréningovú množinu bodov a pre daný typ funkcie hľadáme také parametre - koeficienty funkcie, ktoré minimalizujú účelovú funkciu. V tomto prípade je "priestor" funkcií podstatne ohraničený, ich tvar sa mení len v dôsledku zmeny parametrov funkcie, ktorej "funkcionálny" tvar je fixný. John Koza nazval takýto zovšeobecnený prístup k regresii *symbolická regresia*.

Symbolická regresia je len jedna možná interpretácia genetického programovania. Iný pohľad na genetické programovanie je chápanie funkcie $t(x)$ ako "programu" pre správanie sa nejakého objektu v prostredí. Tak napríklad pohyb robota v prostredí môže byť určený funkciou $y=t(x)$, kde x sú vstupné informácie (napr. z televíznej kamery) o jeho blízkom okolí a y sú príkazy pre jeho motorické zariadenie. Naším cieľom je zostrojiť takú funkciu $t(x)$, ktorá čo najlepšie popisuje pohyb robota (umelého živočicha) a jeho úspešnosť v prostredí. V tomto prípade už nemôžeme hovoriť o tréningovej množine A ako o množine bodov, ktorá je vopred zadaná. Úspešnosť danej funkcie $t(x)$ je daná úspešnosťou robota v prostredí po určitom vopred danom počte krokov (napr. počtom kúskov potravy, ktoré nazbieral pri pohybe v prostredí po n krokoch). Vyššie uvedený postup pre ohodnotenie funkcie $f(x)$ je vhodný pre riešenie širokej triedy problémov adaptácie pomocou genetického

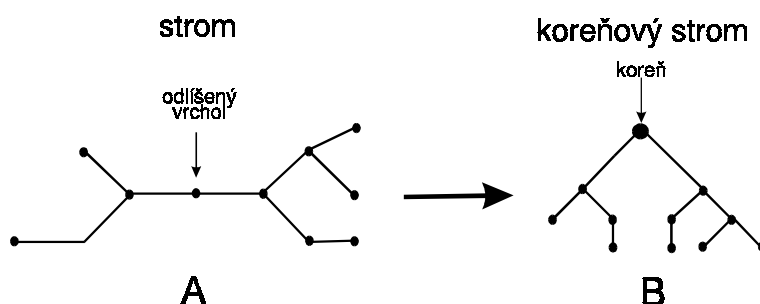
programovania. Môžeme teda konštatovať, že prístup genetického programovania je vhodný na riešenie dvoch tried problémov, a to symbolickej regresie a adaptácie (alebo učenia) [4-8].

Pre úspešnú a hlavne efektívnu implementáciu genetického programovania hrá významnú úlohu metóda kódovania stromových štruktúr. John Koza implementoval genetické programovanie v jazyku LISP (funkcionálny logicky orientovaný jazyk), ktorý má účinné programovacie prostriedky pre kódovanie stromov a manipuláciu s nimi. Napriek jednoduchosti napísaných programov v LISP-e je ich efektívnosť neobyčajne malá, reálne aplikácie vyžadujú pracovné stanice riscovského typu. Na prvý pohľad sa zdá, že genetické programovanie je tak zložitá aplikácia, že a-priori vyžaduje pracovné stanice riscovského typu a na počítačoch triedy PC nie je (alebo je len veľmi obtiažne) realizovateľná. V ďalšej časti tejto kapitoly ukážeme jednoduchý spôsob kódovania stromových štruktúr pomocou tzv. Readovho lineárneho kódu a naznačíme jeho implementáciu v pseudopascal. Jeho výhodou je, že namiesto funkcionálnych jazykov umožňuje implementovať genetické programovanie pomocou štandardných procedurálnych jazykov (Pascal, C/C++, Fortran, atď.) na počítačoch triedy PC, pričom výsledný kód je dostatočne efektívny, takže sú ľahko implementovateľné aj netriviálne aplikácie.

5.2 Koreňové stromy a Readov lineárny kód

Nech $G=(V,E)$ je *strom* (súvislý acyklický graf [9]), kde $V=\{v_1, v_2, \dots, v_p\}$ je neprázdna množina vrcholov a $E=\{e_1, e_2, \dots, e_q\}$ je množina hran (pozri obr. 5.1). Kardinality týchto dvoch množín spĺňajú podmienku

$$|V| = |E| + 1 \quad (5.1)$$



Obrázok 5.1. Diagramatická reprezentácia stromu (A) a koreňového stromu (B). Ak sa jeden vrchol v strome odlišuje od ostatných vrcholov, potom sa strom nazýva *koreňový strom* a odlišný vrchol sa nazýva *koreň*.

Nech $v \in V$ je vrchol stromu G , valencia (stupeň) tohto vrcholu, označená $val(v)$, je nezáporné celé číslo, ktoré určuje počet hrán incidentných s vrcholom v . Súčet valencií všetkých vrcholov spĺňa podmienku [9]

$$\sum_{v \in V} val(v) = 2q = 2(p-1) \quad (5.2)$$

Koreňový strom je strom, ktorý má jeden vrchol špeciálne odlišný od ostatných vrcholov, tento vrchol sa nazýva *koreň* (pozri obr. 5.1, diagram B). Koreňový strom je formálne určený ako usporiadaná trojica

$$T = (V, E, v) \quad (5.3)$$

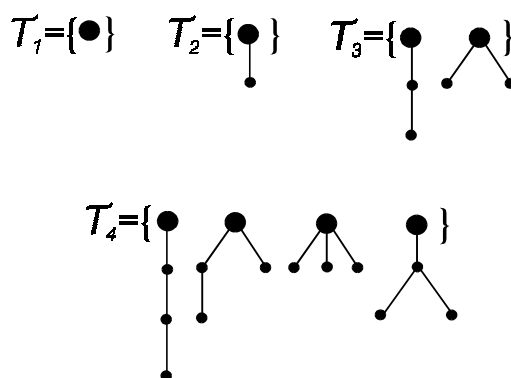
kde $v \in V$ je koreň. Dva koreňové stromy $T = (V, E, v)$ a $T' = (V', E', v')$ sú izomorfné [9] vtedy a len vtedy, ak existuje také jedno-jednoznačné zobrazenie $\phi: V \rightarrow V'$ ($T \approx T'$), ktoré zachováva susednosť vrcholov a zobrazuje koreň stromu T na koreň stromu T' , t.j. $\phi(v) = v'$.

Úloha 5.1. Dokážte vzťahy (5.1) a (5.2).

Nech $\mathcal{T}_i$ je množina obsahujúca všetky možné koreňové stromy, ktoré sú zložené z i vrcholov (pozri obr. 5.2). Zjednotenie týchto množín

$$\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \dots \quad (5.4)$$

obsahuje všetky možné neizomorfné koreňové stromy.



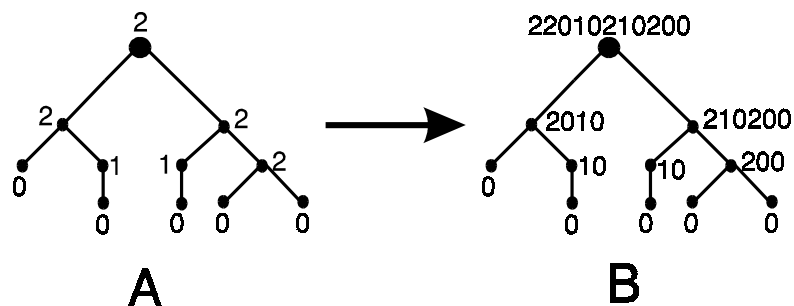
Obrázok 5.2. Prvé štyri množiny $\mathcal{T}_1 - \mathcal{T}_4$, ktoré obsahujú všetky možné neizomorfné koreňové stromy s počtom vrcholov 1 až 4.

Metóda Readovho lineárneho kódovania koreňových stromov sa často používa v teórii grafov na konštruktívnu enumeráciu stromových štruktúr [10,11]. V poslednej dobe sa tiež použila na kódovanie stromov pre potreby takej ich konštrukcie, aby mali požadované vlastnosti [12].

Readov kód pre koreňové stromy je reťazec (sekvencia) celých čísel, ktoré sú priradené buď valencii koreňa alebo valencii zníženej o jednotku pre ostatné vrcholy (pozri obr. 5.3). Lineárny kód koreňového stromu T bude označený $\text{code}(T)$; ak strom T obsahuje jeden vrchol (pozri koreňový strom na obr. 5.2 patriaci do množiny $\mathcal{T}_1$), potom jeho kód je $\text{code}(T) = '0'$. Študujme koreňový strom obsahujúci dva alebo viac vrcholov a predpokladajme, že jeho koreň je incidentný s r hranami, odstránením koreňa zo stromu dostaneme tzv. stromy prvej generácie. Tieto stromy prvej generácie sa znovu uvažujú ako koreňové stromy, koreň je identifikovaný s vrcholom, ktorý bol susedný s koreňom pôvodného stromu. Nech sú vzniknuté koreňové stromy označené $T_1, T_2, \dots, T_r$, potom Readov kód pôvodného stromu T je definovaný vzťahom

$$\text{code}(T) = 'r' + \text{code}(T_1) + \text{code}(T_2) + \dots + \text{code}(T_r) \quad (5.5)$$

kde '+' reprezentuje operáciu zretiazovania. Ak pravá strana (5.5) obsahuje koreňový strom obsahujúci dva alebo viac vrcholov, potom je jeho kód určený vzorcom, ktorý je podobným vzorcu (5.5); táto rekurzívna procedúra sa zastaví vtedy, keď vyskytujúce sa koreňové stromy obsahujú len jeden vrchol (ich kódy sú jednoducho určené reťazcami '0'). Výsledný kód $\text{code}(T)$ má tvar reťazca - postupnosti $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_p)$, jeho dĺžka je označená $|\alpha| = p$.



Obrázok 5.3. Konštrukcia Readovho lineárneho kódu. Diagram A znázorňuje koreňový strom, na ktorom je každý vrchol ohodnotený celým nezáporným číslom, ktoré je buď valenciou vrcholu alebo valenciou zníženou o jednotku pre ostatné vrcholy. Výsledný Readov kód sa zostrojí (diagram B) tak, že idúc postupne zdola nahor spájame ohodnotenie daného vrcholu a kódy nižších vrcholov. Koreňový strom je reprezentovaný kódom, ktorý je priradený koreňu.

Úloha 5.2. Zostrojte Readov kód pre niekoľko náhodne vybraných koreňových stromov.

Základný problém pre efektívnosť Readovho kódu je tzv. analýza (parsing) kódu, ktorá spočíva v jednoznačnom preklade kódu do tvaru, v ktorom sú vrcholy a hrany grafu identifikované. Na toto sa použije jednoduchý *back-track* algoritmus, ktorý zostrojí postupne tzv. *sled* obsahujúci všetky vrcholy koreňového stromu (spanning walk). Tento sled navštívi každú hranu dvakrát (v opačných smeroch), pričom počiatočný a konečný bod tohto sledu je koreň (pozri algoritmus 5.1 a obr. 5.4).

```

procedure Parsing_Code(input:  $\alpha$ ; output:  $V, E$ );
begin  $E := \emptyset$ ;  $V := \{1\}$ ;
       $index_0 := 1$ ;  $branch_0 := \alpha_1$ ;  $d := 0$ ;  $i := 1$ ;
      while  $d \geq 0$  do
        if  $branch_d > 0$  then
          begin  $branch_d := branch_d - 1$ ;  $i := i + 1$ ;  $d := d + 1$ ;
                 $branch_d := \alpha_i$ ;  $index_d := i$ ;
                 $V := V \cup \{index_d\}$ ;
                 $E := E \cup \{[index_{d-1}, index_d]\}$ ;
          end else  $d := d - 1$ ;
      end;

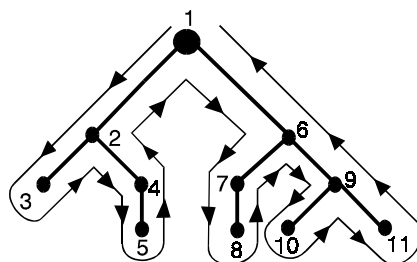
```

Algoritmus 5.1. Pseudopascalovská implementácia analýzy Readovho lineárneho kódu, ktorý vyhovuje podmienkam (5.6a-b), t.j. k analyzovanému kódu existuje koreňový strom. Výstupné parametre V a E sú vrcholová, resp. hranová množina. Hĺbka algoritmu "back-track" je určená celočíselnou premennou d . Algoritmus postupne navštívi všetky vrcholy koreňového stromu (pozri obr. 5.4). Po ukončení algoritmu obsahujú množiny V a E všetky vrcholy, resp. hrany, pričom koreň je indexovaný 1.

Teraz uvedieme niekoľko poznámok k jednoznačnosti Readovho kódu. Ako vyplýva z vyššie uvedenej konštrukcie kódu, táto sa môže realizovať mnohými spôsobmi, t.j. koreňový strom T je reprezentovaný mnohými kódmi. Ináč povedané, dva rôzne kódy môžu reprezentovať koreňové stromy, ktoré sú izomorfné. Toto ohraňenie Readovho kódu sa jednoducho odstráni tak, že vzorec (5.5) sa aplikuje takým spôsobom, že príslušné "podkódy" sa usporiadajú pred ich zretazením do nerastúcej alebo neklesajúcej postupnosti.

Týmto jednoduchým spôsobom získame tzv. *kanonický Readov kód*, ktorý má tú vlastnosť, že ak dva kanonické kódy sú rôzne, potom odpovedajúce koreňové stromy nie sú izomorfné. V našich ďalších úvahách nebudeme používať kanonický Readov kód, pre potreby genetického programovania je úplne postačujúci obyčajný Readov kód.

Úloha 5.3. Pre koreňové stromy z úlohy 5.2 indexujte vrcholy pomocou algoritmu 5.1.



Obrázok 5.4. Konštrukcia "back-track" sledu obsahujúceho všetky vrcholy koreňového stromu, ktorý začína a končí v koreni stromu. Vrcholy sú postupne indexované tak, ako sú v slede navštevované.

Obráťme teraz našu pozornosť na problém, za akých podmienok je postupnosť p nezáporných celých čísel priradená Readovmu kódu. Takáto postupnosť sa nazýva *grafová*, ak existuje koreňový strom obsahujúci p vrcholov, ktorého Readov kód je totožný s postupnosťou.

Veta 5.1. [10,11]. Nutné a postačujúce podmienky k tomu, aby postupnosť nezáporných celých čísel $(\alpha_1, \alpha_2, \dots, \alpha_p) \in \{0, 1, 2, \dots\}^p$ bola grafová (t.j. existuje koreňový strom s rovnakým kódom) majú tvar

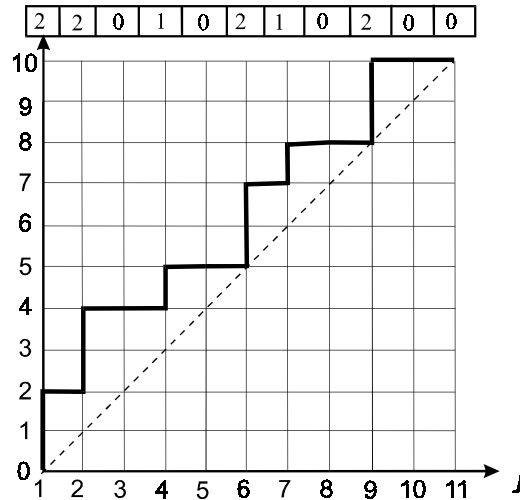
$$\sum_{i=1}^j \alpha_i \geq j \quad (j = 1, 2, \dots, p-1) \quad (5.6a)$$

$$\sum_{i=1}^p \alpha_i = p-1 \quad (5.6b)$$

Druhú vlastnosť (5.6b) možno jednoducho dokázať použitím vzťahu (5.2). Táto veta má veľký význam pre náhodné generovanie koreňových stromov a pre hľadanie podstromov v koreňových stromoch, čo sú dôležité operácie pre jednoduchú implementáciu mutácie a kríženia nad stromami.

Úloha 5.4. Pomocou obr. 5.5 náhodne vygenerujte niekoľko kódov a priradte im koreňové stromy.

Úloha 5.5. Náhodne vygenerujte kódy pomocou nerovností (5.6a-b) a verifikujte ich pomocou grafu na obr. 5.5.



Obrázok 5.5. Diagramatické znázornenie podmienok (5.6a-b), jednotlivé zložky kódu (22010210200) sa interpretujú ako dĺžky vertikálnych častí hrubej lomenej čiary. Prerušovaná čiara vyjadruje nerovnosti (5.6a-b), hrubá čiara musí ležať nad touto čiarou.

Definujeme rekurentne tieto veličiny

$$d_1 = p - 1 \quad (5.7a)$$

$$d_j = d_{j-1} - \alpha_{j-1} \quad (j = 2, 3, \dots, p) \quad (5.7b)$$

ktoré ohraničujú zhora zložky kódu $(\alpha_1, \alpha_2, \dots, \alpha_p)$

$$1 \leq \alpha_j \leq d_j \quad (\text{if } d_j = p - j) \quad (5.8a)$$

$$0 \leq \alpha_j \leq d_j \quad (\text{if } d_j < p - j) \quad (5.8b)$$

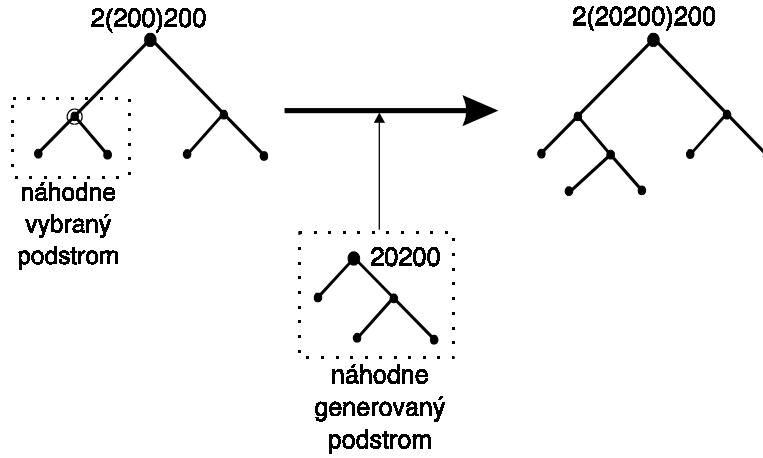
pre $j=1, 2, \dots, p$. Tieto ohraničujúce podmienky sú veľmi výhodné pre náhodnú generáciu Readových kódov tak, aby boli splnené podmienky (5.6a-b) (pozri obr. 5.5). Implementácia náhodnej generácie Readových lineárnych kódov sa uskutoční pomocou algoritmu 5.2.

```

procedure Generation_Code(output:p, $\alpha$ ) ;
begin p:=pmin+random(pmax-pmin+1) ;
       $\alpha_1$ :=1+random(p-1) ; d1:=p-1;
      for j:=2 to p-1 do
        begin dj:=dj-1- $\alpha_{j-1}$ ;
              if dj=p-j then  $\alpha_j$ :=1+random(dj)
              else  $\alpha_j$ :=random(dj+1) ;
        end;
       $\alpha_p$ :=0;
end;

```

Algoritmus 5.2. Pseudopascalovská implementácia náhodného generovania Readovho lineárneho kódu dĺžky p , ktorý je automaticky grafový, t.j. vyhovuje podmienkam (5.6a-b). Funkcia `random(I)` je náhodný generátor celých čísel z uzavretého intervalu $[0, I-1]$. Algoritmus je založený na podmienkach (5.7-8). Dĺžka generovaného kódu je ohraničená podmienkami $p_{min} \leq p \leq p_{max}$.



Obrázok 5.6. Ilustratívny príklad mutácie koreňového stromu. V prvom kroku je náhodne vybraný nekoreňový vrchol, ktorý je zakrúžkovaný na ľavom koreňovom strome. Príslušný koreňový podstrom je nahradený iným náhodne generovaným koreňovým podstromom. Počiatočný a výsledný kód príslušných koreňových stromov sú uvedené pri koreňových vrchoch.

Pre možnosť použitia kódovania koreňových stromov pomocou Readovho prístupu v rámci evolučných algoritmov je potrebné implementovať v rámci tohto prístupu operácie mutácie a kríženia [12-14]. Operácia mutácie priradí koreňovému stromu stochasticky iný koreňový strom, pričom oba stromy môžu mať rôzny počet vrcholov (počty vrcholov sú obvykle ohraničené intervalom $[p_{min}, p_{max}]$). Ak sú starý a nový lineárny kód označené α a α' , potom mutáciu formálne píšeme ako operátor transformujúci reťazec α na reťazec α'

$$\alpha' = O_{mut}(\alpha) \quad (5.9)$$

kde $|\alpha|, |\alpha'| \in [p_{min}, p_{max}]$. Podľa Kozu [1] sa mutácia koreňového stromu realizuje tak, že v prvom kroku sa náhodne vyberie nekoreňový vrchol a príslušný podstrom sa nahradí iným náhodne generovaným podstromom (pozri obr. 5.6).

Alternatívna reprezentácia procesu mutácie sa môže formulovať nasledujúcim spôsobom. Nech je kód α rozdelený na tri podkódy, $\alpha = (\alpha_1, \alpha_2, \alpha_3)$, kde podkód α_2 odpovedá nejakému podstromu stromu vyjadrenému kódom α (pozri obr. 5.6 a 5.7). Podkód α_2 sa nahradí novým náhodne generovaným podkódom α'_2 , potom dostaneme nový kód interpretovaný ako mutácia kódu α , $\alpha' = (\alpha_1, \alpha'_2, \alpha_3)$. Dĺžka podkódov α_2 a α'_2 je ohraničená nerovnosťami

$$p_{min} - |\alpha| + |\alpha_2| \leq |\alpha'_2| \leq p_{max} - |\alpha| + |\alpha_2| \quad (5.10)$$

Podkód α_2 kódu α sa určí tak, že najprv náhodne určíme tzv. mutačný bod $2 \leq \tau \leq |\alpha| - 1$, ktorý špecifikuje polohu v kóde α , od ktorej začína podkód α_2 . Ako efektívne stanovíme dĺžku podkódu α_2 ? Na tento účel je vhodné použiť nerovnosti (5.6a-b); hovoríme, že podkód α_2 má dĺžku r , ak sú splnené nasledujúce podmienky

$$\sum_{i=1}^j \alpha_{i+\tau-1} \geq j \quad (j = 1, 2, \dots, r-1) \quad (5.11a)$$

$$\sum_{i=1}^r \alpha_{i+\tau-1} = r-1 \quad (5.11b)$$

Úloha 5.6. Pre kódy vygenerované v úlohe 5.4 určte podkódy pomocou nerovností (5.11). Vykonajte grafickú interpretáciu podkódu.

Pseudopascalovská implementácia mutácie Readovho lineárneho kódu je uvedená v algoritme 5.3.

```

procedure Mutation(input:p,  $\alpha$ ; output:p',  $\alpha'$ );
begin  $\tau := 2 + \text{random}(p-1)$ ;
      r:=dĺžka podkódu  $\alpha_2$  kódu  $\alpha$ , ktorý začína
        v bode  $\tau$ ;
      r':= $p_{\min} - p + r + \text{random}(p_{\max} - p_{\min} + 1)$ ;
      Generation_Code(r',  $\alpha'_2$ );
       $\alpha' = (\alpha_1, \alpha'_2, \alpha_3)$ ; p' :=  $p - r + r'$ ;
end;

```

Algoritmus 5.3. Implementácia mutačného procesu aplikovaného na kód α , pričom výsledný kód je α' . Algoritmus sa inicializuje náhodným generovaním mutačného bodu τ , dĺžka podkódu, ktorý začína v bode τ je vyjadrená premennou r . Dĺžka nového náhodne generovaného podkódu je označená r' .

```

procedure Crossover(input:p,  $\tilde{p}$ ,  $\alpha$ ,  $\beta$ ; output p',  $\tilde{p}'$ ,  $\alpha'$ ,  $\beta'$ );
begin  $\tau := 2 + \text{random}(p-1)$ ;  $\tilde{\tau} := 2 + \text{random}(\tilde{p}-1)$ 
       $\alpha' = (\alpha_1, \alpha_2, \alpha_3)$ ;  $\beta = (\beta_1, \beta_2, \beta_3)$ ;
      p' :=  $p - |\alpha_2| + |\beta_2|$ ;  $\tilde{p}' := \tilde{p} - |\beta_2| + |\alpha_2|$ ;
end;

```

Algoritmus 5.4. Implementácia operátora kríženia dvoch rodičovských lineárnych kódov α a β , ktoré sú modifikované na nové lineárne kódy - potomkov α' a β' . Náhodný výber bodov kríženia τ a $\tilde{\tau}$ sa realizuje tak, že dĺžky nových lineárnych kódov vyhovujú nerovnostiam $p_{\min} \leq |\alpha'| \leq p_{\max}$ a $p_{\min} \leq |\beta'| \leq p_{\max}$.

Nech α a β sú dva (rodičovské) lineárne kódy koreňových stromov, aplikovaním operácie kríženia dostaneme dva nové lineárne kódy (potomkov) α' a β'

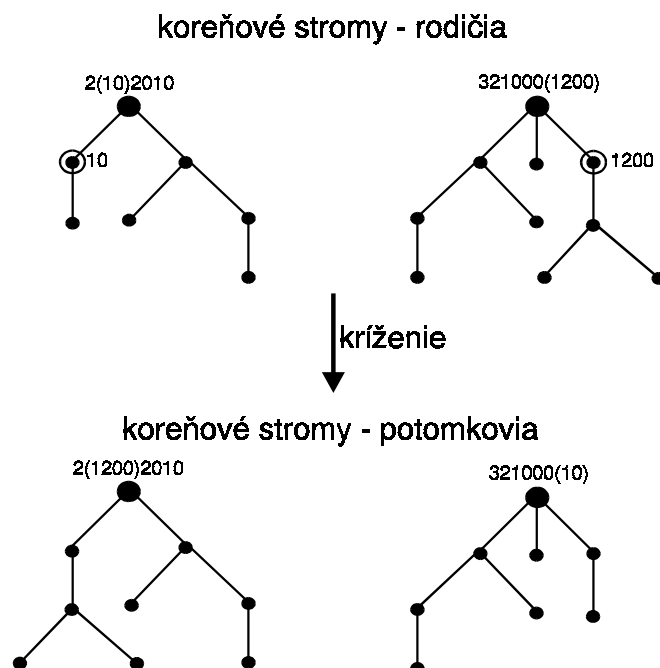
$$(\alpha', \beta') = O_{\text{cross}}(\alpha, \beta) \quad (5.12)$$

Táto operácia sa realizuje podľa Kozu [1,2] tak, že v rodičovských kódoch - chromozómoch α a β náhodne vyberieme dva body kríženia a príslušné podkódy, ktoré začínajú v týchto bodoch, sa vymenia (pozri obr. 5.7).

Alternatívna reprezentácia operátora kríženia je nasledujúca: nech rodičovské kódy α a β koreňových stromov majú tvar $\alpha = (\alpha_1, \alpha_2, \alpha_3)$, resp. $\beta = (\beta_1, \beta_2, \beta_3)$, kde počiatočné body podkódov α_2 a β_2 boli náhodne generované. Aplikácia operácie kríženia na rodičovské kódy α a β spočíva vo výmene podkódov α_2 and β_2 , potomkovia majú potom tvar

$$\alpha' = (\alpha_1, \beta_2, \alpha_3) \text{ a } \beta' = (\beta_1, \alpha_2, \beta_3) \quad (5.13)$$

Dĺžka potomkov je určená vzťahmi $|\alpha'| = |\alpha| - |\alpha_2| + |\beta_2|$ a $|\beta'| = |\beta| - |\beta_2| + |\alpha_2|$, pričom tieto dĺžky sú ohraničené nerovnosťami $p_{\min} \leq |\alpha'| \leq p_{\max}$, resp. $p_{\min} \leq |\beta'| \leq p_{\max}$. Pseudopascalovská implementácia operátora kríženia je uvedená v algoritme 5.4.



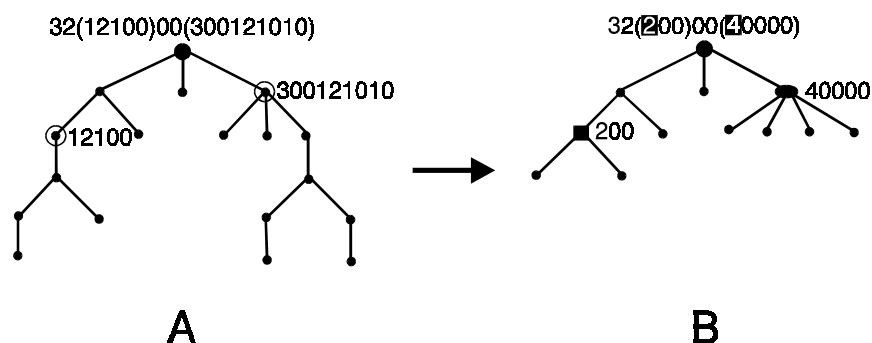
Obrázok 5.7. Ilustratívny príklad operácie kríženia dvoch rodičovských koreňových stromov, ktoré sú modifikované na dva nové koreňové stromy - potomkov. Zakrúžkované vrcholy označujú náhodne generované body kríženia. Operácia kríženia spočíva v tom, že podstromy sa medzi koreňovými stromami vymenia.

Posledný problém, ktorý budeme študovať v tejto kapitole, je tzv. kompresia Readovho kódu, ktorá úzko súvisí s prístupom ADF (Automatically Defined Function) používaným Kozom na zvýšenie efektívnosti genetického programovania [2]. Nech je koreňový strom T reprezentovaný kódom α . Ak vyberieme v tomto strome podstrom, potom kód α môže byť vyjadrený rozkladom $\alpha = (\alpha_1, \alpha_2, \alpha_3)$, kde α_2 odpovedá kódu vybraného podstromu (pozri obr. 5.8). Proces kompresie spočíva v substitúcii podkódu α_2 tzv. komprimovaným kódom typu $(n00...0)$, kde n je počet koncových vrcholov v podstrome (alebo počet zložiek '0' v podkóde α_2).

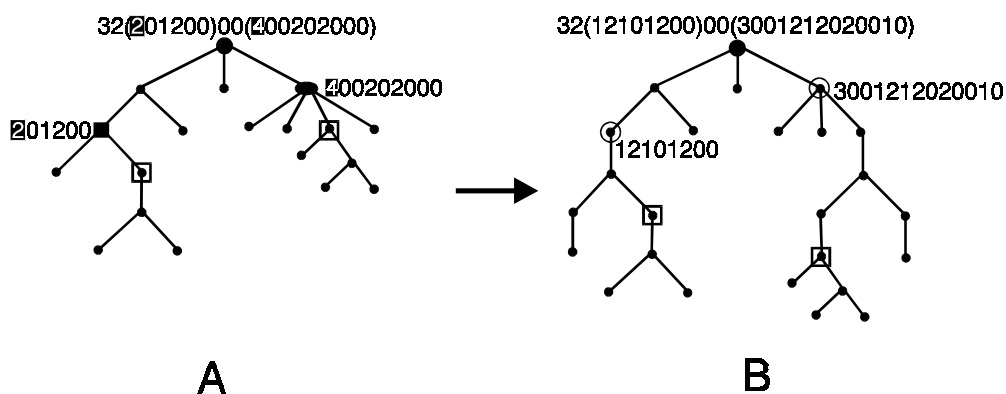
$$\alpha = (\alpha_1, \alpha_2, \alpha_3) \xrightarrow{\text{kompresia}} \alpha_{\text{compress}} = (\alpha_1, \alpha_{2,\text{compress}} = (n00...0), \alpha_3) \quad (5.14)$$

Ako realizovať inverzný proces ku kompresii? Nech kód β obsahuje vrchol interpretovaný ako "komprimovaný" vrchol. Potom sa kód β môže dekomprimovať tak, že odpovedajúci podstrom priradený komprimovanému vrcholu sa nahradí koreňovým podstromom, ktorý indukuje daný komprimovaný vrchol, pričom niektoré koncové vrcholy sa môžu substituovať podstromami (pozri obr. 5.9).

Úloha 5.7. Podrobne preštudujte príklady kompresie a dekompresie znázornené na obrázkoch 5.8 a 5.9. Navrhnite iné prípady kompresie a dekompresie.



Obrázok 5.8. Ilustračný príklad kompresie Readovho lineárneho kódu. Koreňový strom reprezentovaný diagramom A sa komprimuje na menší koreňový strom reprezentovaný diagramom B. Tento proces sa vykoná tak, že vybrané podstromy (ich koreňové vrcholy sú zakrúžkované) v diagrame A sa nahradia novými vrcholmi (štvorcovým alebo oválnym), ktoré sú incidentné len s koncovými vrcholmi, pričom ich počet je rovnaký ako počet koncových vrcholov v pôvodných podstromoch.



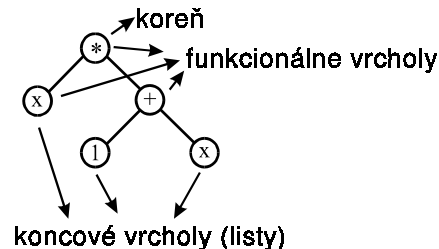
Obrázok 5.9. Ilustratívny príklad dekompresie Readovho lineárneho kódu. Koreňový strom A obsahuje dva komprimované vrcholy reprezentované štvorcovým a oválnym vrcholom, pričom vrcholy označené štvorcom (pôvodne koncové) sú nahradené reťazcami 1200, resp. 20200. Ak sú komprimované vrcholy "dekomprimované" odpovedajúcimi koreňovými podstromami (pozri obr. 5.8), potom sa pôvodný strom A dekomprimuje na väčší strom B, ktorý už obsahuje len štandardné vrcholy.

5.3 Symbolická regresia

Symbolická regresia patrí medzi základné aplikácie genetického programovania. Ako už bolo spomenuté v úvodnej kapitole 5.1, symbolická regresia spočíva v hľadaní takej funkcie reprezentovanej syntaktickým stromom s predpísanými operáciami, ktorá čo najlepšie aproximuje údaje z tréningovej množiny.

Čo je syntaktický strom? Pod syntaktickým stromom t budeme rozumieť koreňový strom T , ktorého vrcholy sú ohodnotené symbolmi aritmetických (alebo iných) operácií a celý strom sa môže ohodnotiť reálnym číslom reprezentujúcim hodnotu funkcie, ktorá je priradená

stromu pre danú vstupnú hodnotu nezávislej premennej (alebo nezávislých premenných). Vrcholy v syntaktických stromoch klasifikujeme takto (pozri obr. 5.10):



Obrázok 5.10. Klasifikácia vrcholov syntaktického stromu na funkcionálne a koncové (listy).

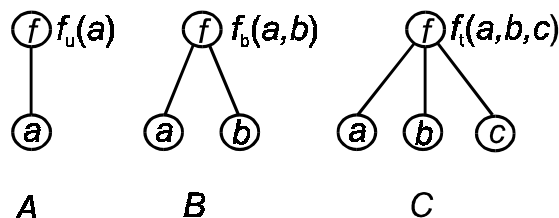
- (1) *Koncové vrcholy* odpovedajú *nezávislým premenným* $x, y, \dots$ alebo *nezáporným celočíselným konštantám* $0, 1, 2, \dots$
- (2) *Funkcionálne vrcholy* odpovedajú *jednoduchým operáciám*, ktoré sú *unárne, binárne, ternárne, ...*

Formálne môžeme syntaktický strom vyjadriť ako usporiadanú dvojicu

$$t = (T, \phi) \quad (5.15)$$

kde T je koreňový strom určujúci štruktúru syntaktického stromu t a ϕ je zobrazenie vrcholov koreňového stromu na "aritmetické" symboly, premenné alebo konštanty

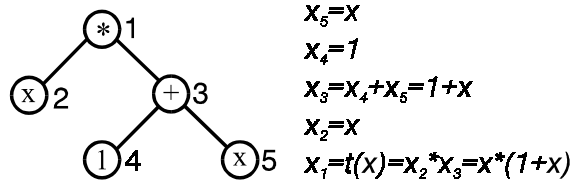
$$\phi : V \rightarrow \{*, +, -, \dots, x, y, \dots, 1, 2, \dots\} \quad (5.16)$$



Obrázok 5.11. Klasifikácia funkcionálnych vrcholov na unárne (A), binárne (B) a ternárne (C). Unárny vrchol má jedného predchodcu, ktorého hodnota a sa zobrazuje na funkčnú hodnotu $f_u(a)$. Binárny vrchol má dvoch predchodcov, ich funkčné hodnoty a a b sú zobrazené na $f_b(a, b)$. Ternárny vrchol má troch predchodcov, hodnoty a , b a c sú zobrazené na $f_t(a, b, c)$. Vo všeobecnosti platí, že binárne, unárne, ... vrcholy nie sú "komutatívne", t.j. napr. pre binárny vrchol platí $f_t(a, b) \neq f_t(b, a)$.

Každému syntaktickému stromu t priradíme funkciu $t(x, y, \dots)$, kde $x, y, \dots$ sú nezávislé premenné, nasledujúcim rekurzívnym spôsobom: Nech je každý vrchol syntaktického stromu označený indexom, toto indexovanie je realizovateľné pomocou algoritmu 5.1 (pozri obr. 5.12). Potom sa i -temu vrcholu môže predpísať "aktivita" x_i tak, že koncové vrcholy pre tieto aktivity sú rovné ohodnoteniam týchto vrcholov (t.j. konštantám alebo nezávislým premenným), pri postupe zdola nahor tieto aktivity postupne počítame tak, že na základe topológie syntaktického stromu počítame aktivity tých vrcholov, ktoré majú predchodcov s už známymi hodnotami aktivít. Tento proces končí v koreni stromu, jeho "aktivita" sa rovná hodnote funkcie $t(x, y, \dots)$ priradenej syntaktickému stromu t (pozri obr. 5.12)

$$x_1 = t(x, y, \dots) \quad (5.17)$$



Obrázok 5.12. Ilustratívny príklad interpretácie jednoduchého syntaktického stromu, jednotlivé "aktivity" sa počítajú "dopredným šírením" po strome, začína sa od koncových vrcholov a končí sa v koreni stromu, jeho "aktivita" priamo odpovedá hodnote funkcie.

Pre daný súbor funkcionálnych a koncových vrcholov môžeme definovať množinu $\mathcal{T}$ obsahujúcu všetky možné neekvivalentné syntaktické stromy, ktoré majú počet vrcholov z predpísaného intervalu a ktoré sú ohodnotené symbolmi z predpísaných súborov

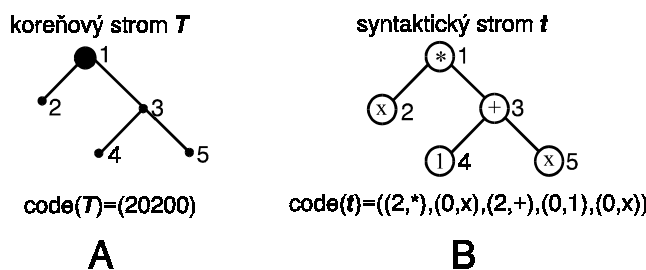
$$\mathcal{T} = \{t_1, t_2, \dots\} \quad (5.18)$$

Budeme používať konvenciu, že každý syntaktický strom $t \in \mathcal{T}$ sa identifikuje so svojou interpretáciou - funkciou $t(x, y, \dots)$. To znamená, že množina $\mathcal{T}$ sa môže tiež alternatívne chápať ako množina funkcií, ktorých tvar je ohraničený podmienkami kladenými na syntaktické stromy.

Ako kódovať syntaktické stromy? Podľa (5.15) je syntaktický strom definovaný ako koreňový strom, ktorého vrcholy sú ohodnotené zobrazením (5.16), $t = (T, \phi)$. Nech Readov lineárny kód stromu T je $\text{code}(T) = \alpha = (\alpha_1, \alpha_2, \dots, \alpha_p)$, potom sa lineárny kód syntaktického stromu t môže zostrojiť tak, že Readov kód koreňového stromu sa rozšíri o zobrazenie ϕ

$$\text{code}(t) = ((\alpha_1, \phi_1), (\alpha_2, \phi_2), \dots, (\alpha_p, \phi_p)) \quad (5.19)$$

kde ϕ_i je ohodnotenie i -teho vrcholu buď funkcionálnym alebo koncovým symbolom. Týmto spôsobom sme dostali jednoduchý prostriedok na kódovanie syntaktických stromov pomocou Readovho lineárneho kódovania koreňových stromov, pozri obr. 5.13.



Obrázok 5.13. Tvorba kódu syntaktického stromu t pomocou Readovho lineárneho kódu priradeného koreňového stromu T . Kód syntaktického stromu získame z pôvodného kódu koreňového stromu tak, že každú jeho zložku sprevádza aj ohodnotenie príslušného vrcholu.

Majme tréningovú množinu obsahujúcu n bodov (regresnú tabuľku)

$$A_{\text{train}} = \{x_i / y_i; i = 1, 2, \dots, n\} \quad (5.20)$$

Cieľom štandardnej regresnej analýzy je nájsť také optimálne parametre modelovej funkcie $G(x; \mathbf{w})$, kde $\mathbf{w}$ sú parametre funkcie G , že nasledujúca účelová funkcia sa minimalizuje

$$E(\mathbf{w}) = \sum_{i=1}^n |G(x_i; \mathbf{w}) - y_i| \quad (5.21)$$

Táto funkcia má minimum v bode

$$\mathbf{w}_{opt} = \arg \min_{\mathbf{w}} E(\mathbf{w}) \quad (5.22)$$

Hovoríme, že adaptovaná funkcia $G(\mathbf{x}, \mathbf{w}_{opt})$ modeluje tréningovú množinu A_{train} .

Symbolická regresia ide ďalej ako štandardná regresná analýza, hľadá v množine $\mathcal{T}$ takú funkciu, že nasledujúca účelová funkcia (funkcionál) sa minimalizuje

$$E(t) = \sum_{i=1}^n |t(x_i) - y_i| \quad (5.23)$$

pričom táto účelová funkcia má minimum v "bode"

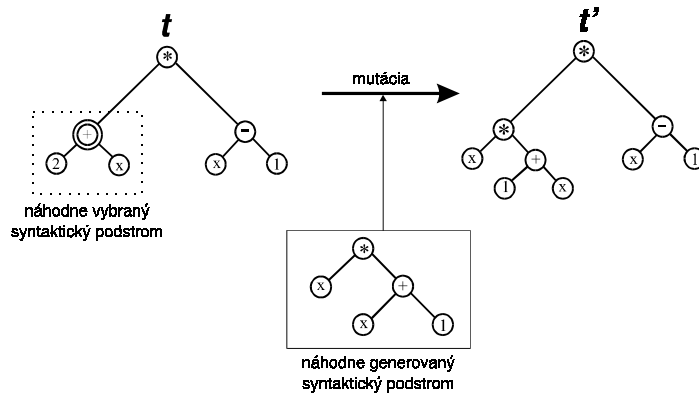
$$t_{opt} = \arg \min_{t \in \mathcal{T}} E(t) \quad (5.24)$$

Symbolická regresia je hlavným cieľom Kozovho genetického programovania s obmedzením, že povolené sú len tie funkcie, ktoré môžu byť reprezentované syntaktickým stromom obsahujúcim povolené funkcionálne a koncové vrcholy.

Riešenie minimalizačného problému (5.22) sa realizuje pomocou genetického algoritmu prezentovaného v predchádzajúcej kapitole. Stochastické operácie mutácie a kríženia sú definované podobným spôsobom ako pre koreňové stromy. Operácia mutácie zmení syntaktický strom t na iný syntaktický strom t' , formálne

$$t' = O_{mut}(t) \quad (5.25)$$

určitá časť syntaktického stromu (t.j. syntaktický podstrom) sa nahradí náhodne generovaným syntaktickým podstromom (pozri obr. 5.14).



Obrázok 5.14. Ilustračný príklad mutácie syntaktického stromu t na iný syntaktický strom t' . V prvej etape mutácie sa náhodne vyberie podstrom a tento sa v druhej etape zamení za náhodne vygenerovaný podstrom. Formálne môžeme povedať, že stochastickou operáciou mutácie sa funkcia $t(x)$ priradená pôvodnému stromu zamenila za novú funkciu $t'(x)$.

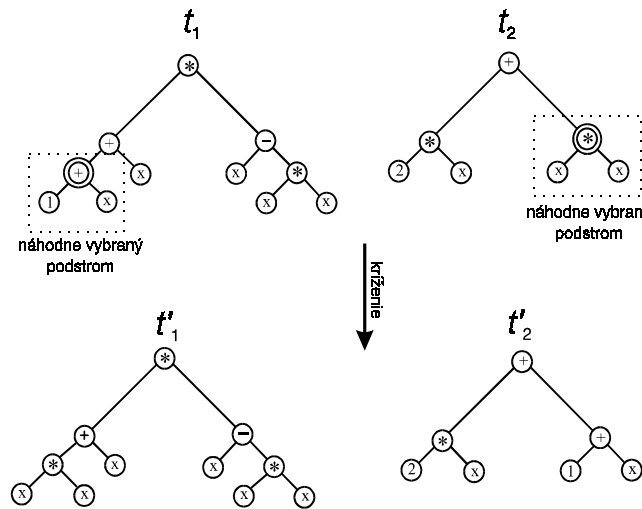
Každému syntaktickému stromu môžeme jednoznačne priradiť funkciu $t(x, y, \dots)$, potom mutáciu môžeme formálne chápať ako stochastickú transformáciu funkcie na inú funkciu, v prípade ilustračného príkladu na obr. 5.14 tieto funkcie majú tvar

$$\begin{aligned}
 t(x) &= (2+x) * (x-1) \\
 t'(x) &= (x * (1+x)) * (x-1)
 \end{aligned}
 \tag{5.26}$$

Operácia kríženia pre dva syntaktické stromy je definovaná analogickým spôsobom ako pre koreňové stromy. Majme dva syntaktické stromy t_1 a t_2 , stochastickou operáciou kríženia sa tieto dva stromy transformujú na dva nové stromy t'_1 a t'_2

$$(t'_1, t'_2) = O_{cross}(t_1, t_2) \tag{5.27}$$

Táto operácia sa realizuje tak, že v stromoch náhodne vyberieme dva podstromy a tieto navzájom vymeníme (pozri obr. 5.15).



Obrázok 5.15. Ilustračný príklad kríženia dvoch syntaktických stromov. V prvej etape v každom strome sa náhodne vyberie podstrom, v druhej etape sa tieto podstromy vzájomne zamenia. Formálne môžeme povedať, že operáciou kríženia sa dve funkcie $t_1(x)$ a $t_2(x)$ pretransformovali na nové funkcie $t'_1(x)$ a $t'_2(x)$.

Podobne ako v predchádzajúcom prípade mutácie aj pre kríženie môžeme hovoriť, že stochastická operácia kríženia nám transformovala východiskové funkcie $t_1(x)$ a $t_2(x)$ na nové funkcie $t'_1(x)$ a $t'_2(x)$. Pre prípad syntaktických stromov na obr. 5.15 majú tieto funkcie tvar

$$\begin{aligned}
 t_1(x) &= (1+2x) * (x-x^2) \\
 t_2(x) &= 2x + x^2 \\
 t'_1(x) &= (x^2 + x) * (x-x^2) \\
 t'_2(x) &= 2x + (1+x)
 \end{aligned}
 \tag{5.28}$$

Úloha 5.8. Implementujte genetický algoritmus (pozri algoritmus 4.5) tak, že chromozómy budú určené ako Readove lineárne kódy s ohodnotením vrcholov podľa (5.18-19). Toto ohodnotenie vykonajte tak, že syntaktické stromy budú odpovedať buď racionálnym polynómom (t.j. funkčné vrcholy sú $+$, $-$, $*$ a $/$, koncové vrcholy sú celočíselné konštanty alebo premenná x) alebo boolovským funkciám (t.j. funkčné vrcholy sú operácie and , or , xor , a not , a koncové vrcholy sú jednotlivé boolovské premenné).

Účelovú funkciu (5.24) rozšírite o penalizačný člen, ktorý bude uprednostňovať syntaktické stromy s menším počtom vrcholov

$$E(t) = \sum_{i=1}^n |t(x_i) - y_i| + \delta |t|$$

kde δ je malá kladná konštanta a symbol $|t|$ vyjadruje počet vrcholov v strome t . Modifikujte napísanú implementáciu tak, že bude odpovedať horolezeckému algoritmu alebo evolučnému programovaniu. Porovnajete efektívnosť týchto troch rozdielnych metód pri konštrukcii funkcií, ktoré reprodukovujú predpísanú regresnú tabuľku.

5.4 Ilustratívne príklady symbolickej regresie - parita a symetria binárnych vektorov

Nech $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)$ je n -rozmerný binárny vektor. Hovoríme, že tento vektor má párnú (nepárnu) paritu, ak obsahuje párný počet (nepárny počet) jednotiek, párna parita sa definitoricky použije aj pre binárny vektor obsahujúci samé nuly. Táto charakteristika binárnych vektorov sa môže chápať ako boolovská funkcia, ktorá zobrazuje nezávislé binárne premenné $\alpha_1, \alpha_2, \dots, \alpha_n$ na binárnu závislú premennú (1 pre párnú paritu a 0 pre nepárnu paritu)

$$y = \text{parity}(\alpha_1, \alpha_2, \dots, \alpha_n) \quad (5.29)$$

Pre $n=2,3,4$ je funkcia $\text{parity}()$ všetkých možných binárnych vektorov určená regresnými tabuľkami uvedenými v tab. 5.1. Parita binárneho vektora sa môže formálne vyjadriť takto

$$\text{parity}(\alpha) = 1 + \left(\sum_{i=1}^n \alpha_i \right)_{\text{mod } 2} \quad (5.30)$$

To znamená, že parita binárneho vektora α sa určí ako jeden plus modulo 2 počtu jednotiek v binárnom vektore α . Napríklad ak α obsahuje párný počet jednotiek, potom modulo 2 tohto počtu je nula, čiže podľa vyššie uvedenej definície parita je rovná $\text{parity}(\alpha)=1+0=1$.

Symetria binárneho vektora $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)$ je definovaná nasledujúcim spôsobom

$$\text{symmetry}(\alpha) = \begin{cases} 1 & (\text{ak } \alpha_i = \alpha_{n-i+1} \text{ pre } i = 1, 2, \dots, k) \\ 0 & (\text{ostatné prípady}) \end{cases} \quad (5.31)$$

kde $n=2k$ alebo $n=2k+1$. Binárny vektor α má jednotkovú symetriu vtedy, ak má "stred symetrie". Pre $n=2,3,4$ je funkcia symmetry všetkých možných binárnych vektorov určená regresnými tabuľkami uvedenými v tab. 5.1.

Boolovskú funkciu parity môžeme vyjadriť pomocou negácie (*not*) a vylučovacieho alebo (*exclusive or*, *xor*) (pozri tab. 5.2 pre definíciu týchto funkcií) nasledujúcim jednoduchým spôsobom

$$\begin{aligned} \text{parity}(\alpha_1, \alpha_2, \dots, \alpha_n) &= \text{not}(\alpha_1 \text{ xor } \alpha_2 \text{ xor } \dots \text{ xor } \alpha_n) \\ &= (\alpha_1 \text{ xor } \alpha_2 \text{ xor } \dots \text{ xor } \alpha_{n-1}) \text{ xor } (\text{not } \alpha_n) \end{aligned} \quad (5.32)$$

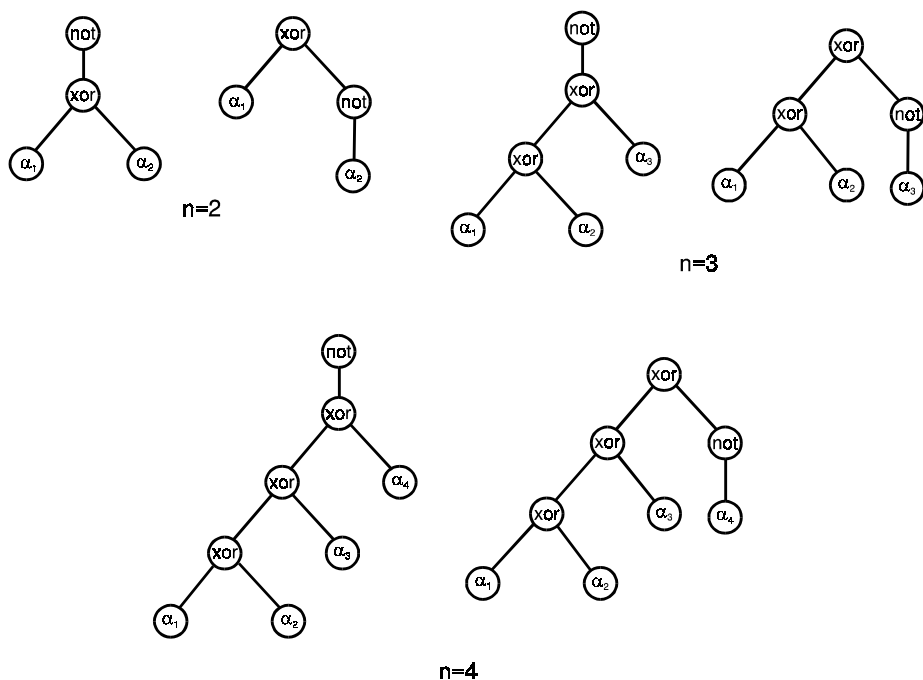
Pre $n=2,3,4$ dostaneme

$$\begin{aligned} \text{parity}(\alpha_1, \alpha_2) &= \text{not}(\alpha_1 \text{ xor } \alpha_2) \\ &= \alpha_1 \text{ xor not } \alpha_2 \end{aligned} \quad (5.33a)$$

$$\begin{aligned} \text{parity}(\alpha_1, \alpha_2, \alpha_3) &= \text{not}(\alpha_1 \text{ xor } \alpha_2 \text{ xor } \alpha_3) \\ &= (\alpha_1 \text{ xor } \alpha_2) \text{ xor } (\text{not } \alpha_3) \end{aligned} \quad (5.33b)$$

$$\begin{aligned} \text{parity}(\alpha_1, \alpha_2, \alpha_3, \alpha_4) &= \text{not}(\alpha_1 \text{ xor } \alpha_2 \text{ xor } \alpha_3 \text{ xor } \alpha_4) \\ &= (\alpha_1 \text{ xor } \alpha_2 \text{ xor } \alpha_3) \text{ xor } (\text{not } \alpha_4) \end{aligned} \quad (5.33c)$$

Diagramatická vizualizácia týchto výrazov pomocou syntaktických stromov je znázornená na obr. 5.16.



Obrázok 5.16. Syntaktické stromy pre boolovskú funkciu *parity* vyjadrenú výrazmi (5.33a-c).

Tabuľka 5.1. Regresné tabuľky pre paritu a symetriu binárnych vektorov dimenzie $n=2,3,4$

No.	$n=2$				$n=3$					$n=4$					
	α_1	α_2	y_{par}	y_{sym}	α_1	α_2	α_3	y_{par}	y_{sym}	α_1	α_2	α_3	α_4	y_{par}	y_{sym}
1	0	0	1	1	0	0	0	1	1	0	0	0	0	1	1
2	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0
3	1	0	0	0	0	1	0	0	1	0	0	1	0	0	0
4	1	1	1	1	0	1	1	1	0	0	0	1	1	1	0
5					1	0	0	0	0	0	0	1	0	0	0
6					1	0	1	1	1	0	1	0	1	1	0
7					1	1	0	1	0	0	1	1	0	1	1
8					1	1	1	0	1	0	1	1	1	0	0
9										1	0	0	0	0	0
10										1	0	0	1	1	1
11										1	0	1	0	1	0
12										1	0	1	1	0	0
13										1	1	0	0	1	0
14										1	1	0	1	0	0
15										1	1	1	0	0	0
16										1	1	1	1	1	1

Tabuľka 5.2. Hodnoty unárnej funkcie *not* a binárnych funkcií *or*, *xor* a *and*.

x_1	x_2	$not\ x_1$	$x_1\ or\ x_2$	$x_1\ xor\ x_2$	$x_1\ and\ x_2$
0	0	1	0	0	0
0	1	1	1	1	0
1	0	0	1	1	0
1	1	0	1	0	1

Boolovská funkcia *symmetry* je určená pomocou (5.31), prepis pomocou logických funkcií má jednoduchý tvar

$$\begin{aligned}
 & \text{symmetry}(\alpha_1, \alpha_2, \dots, \alpha_n) = \\
 & = \begin{cases} not((\alpha_1\ xor\ \alpha_{2k})\ or\ (\alpha_2\ xor\ \alpha_{2k-1})\ or\ \dots\ or\ (\alpha_k\ xor\ \alpha_{k+2})) & (\text{pre } n = 2k + 1) \\ not((\alpha_1\ xor\ \alpha_{2k})\ or\ (\alpha_2\ xor\ \alpha_{2k-1})\ or\ \dots\ or\ (\alpha_k\ xor\ \alpha_{k+1})) & (\text{pre } n = 2k) \end{cases} \quad (5.34)
 \end{aligned}$$

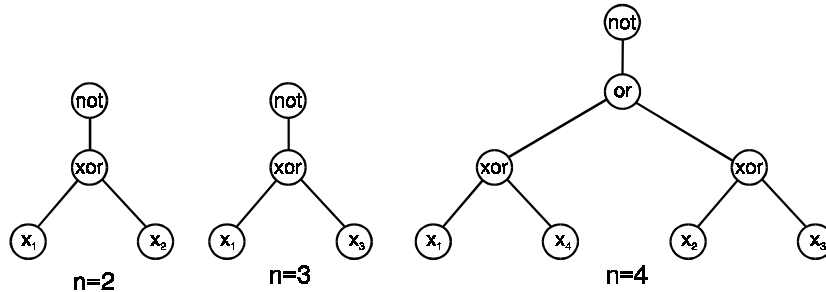
Pre $n=2,3,4$ má táto funkcia tvar

$$\text{symmetry}(\alpha_1, \alpha_2) = not(\alpha_1\ xor\ \alpha_2) \quad (5.35a)$$

$$\text{symmetry}(\alpha_1, \alpha_2, \alpha_3) = not(\alpha_1\ xor\ \alpha_3) \quad (5.35b)$$

$$\text{symmetry}(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = not((\alpha_1\ xor\ \alpha_4)\ or\ (\alpha_2\ xor\ \alpha_3)) \quad (5.35c)$$

Syntaktické stromy priradené týmto výrazom majú tvar znázornený na obr. 5.17.



Obrázok 5.17. Syntaktické stromy boolovských výrazov vyjadrujúcich symetriu binárnych reťazcov podľa výrazov (5.35).

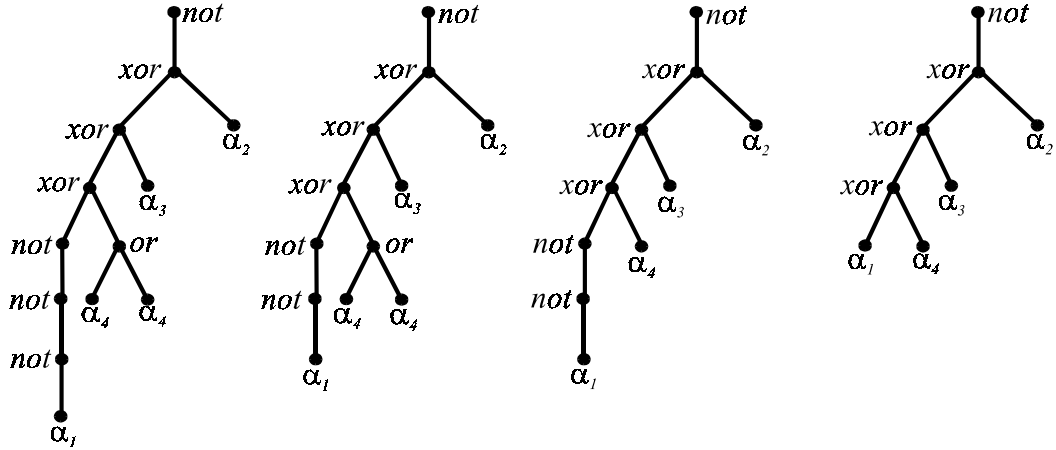
V dôsledku toho, že binárna operácia *xor* je komutatívna a asociatívna, boolovské funkcie (5.32) a (5.34) sú invariantné vzhľadom na ľubovoľnú permutáciu binárnych premenných $\alpha_1, \alpha_2, \dots, \alpha_n$. Napríklad to znamená, že syntaktický strom pre $n=4$ na obr. 5.17 definuje rovnakú boolovskú funkciu pre všetkých $4!=24$ permutácií premenných $\alpha_1, \alpha_2, \alpha_3, \alpha_4$.

Nech množiny povolených vrcholov majú toto zloženie:

- (1) koncové vrcholy $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$,
- (2) unárny vrchol $\{not\}$,
- (3) binárne vrcholy $\{and, or, xor\}$.

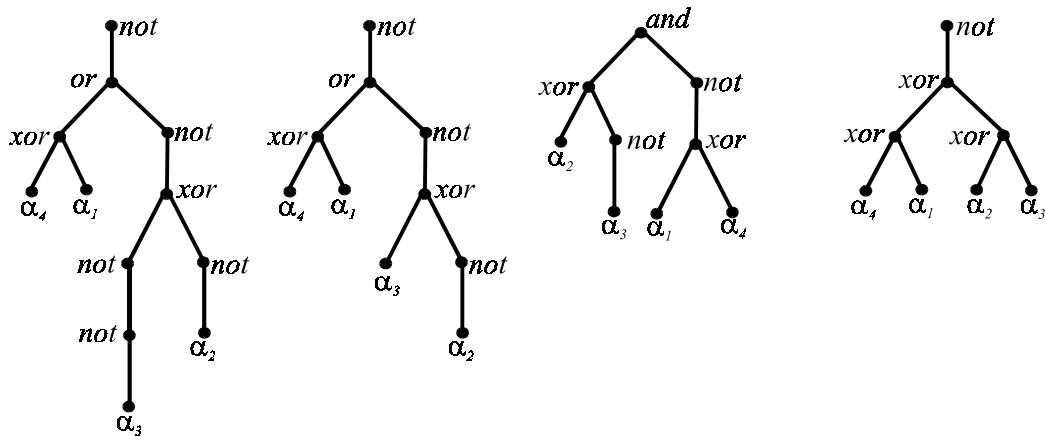
Potom množina $\mathcal{T}$ obsahuje také koreňové stromy, ktorých vrcholy patria do vyššie uvedených troch množín vrcholov. Genetický algoritmus (pozri algoritmus 4.5) je teraz

modifikovaný tak, že populácia chromozómov obsahuje syntaktické stromy z množiny $\mathcal{T}$, ktoré sa náhodne generovali pri inicializácii algoritmu. Každý syntaktický strom - chromozóm odpovedá boolovskej funkcii, pričom hodnota účelovej funkcie (funkcionálu) (5.21) priradená tomuto syntaktickému stromu sa počíta pre danú regresnú tabuľku (pozri tab. 5.1).



epocha=100, $|t|=14$ *epocha*=170, $|t|=12$ *epocha*=350, $|t|=10$ *epocha*=420, $|t|=8$

Obrázok 5.18. Znáznornenie syntaktických stromov produkovaných genetickým programovaním pre symbolickú regresiu parity binárnych vektorov dimenzie $n=4$. Všetky tieto stromy majú nulovú hodnotu účelovej funkcie (5.23), penalizačný člen z úlohy (5.8) zvýhodňuje tie stromy, ktoré majú menší počet vrcholov. Posledný strom je totožný so stromom na obr. (5.16), takže môžeme konštatovať, že v tomto špeciálnom prípade symbolickej regresie poskytuje genetické programovanie optimálny výsledok.



epocha=125, $|t|=12$ *epocha*=205, $|t|=10$ *epocha*=570, $|t|=9$ *epocha*=820, $|t|=8$

Obrázok 5.19. Syntaktické stromy pre symetriu binárnych vektorov dimenzie $n=4$. Posledný strom je totožný so stromom na obr. 5.17. V tomto špeciálnom prípade genetické programovanie opäť poskytuje správny výsledok.

Genetický algoritmus obsahoval populáciu zloženú zo 100 chromozómov (syntaktických stromov zakódovaných pomocou Readovho kódu). Pravdepodobnosť operácie kríženia $P_{cross}=0.5$, operácia mutácie sa aplikuje na každý strom vstupujúci do kríženia s pravdepodobnosťou $P_{mut}=0.1$. Účelová funkcia mala modifikovaný tvar z úlohy 5.8, pričom penalizačná konštanta $\delta=0.01$. Výsledky získané touto verziou genetického programovania pre boolovské funkcie parity a symetrie sú uvedené na obr. 5.18 a 5.19.

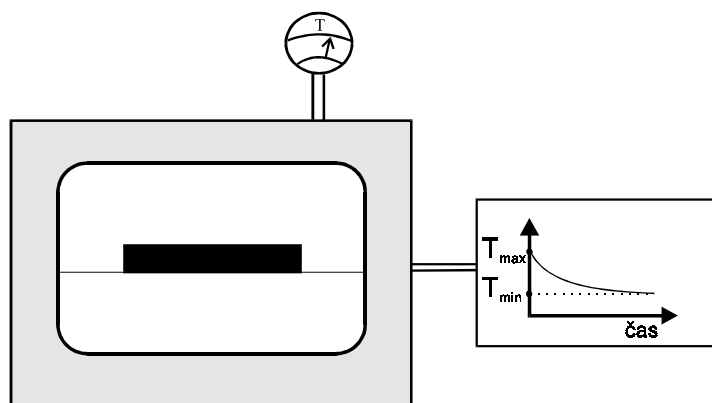
Literatúra

- [1] J. Koza: *Genetic Programming. On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA 1992.
- [2] J. Koza: *Genetic Programming (II). Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA 1994.
- [3] J. Koza, F.H. Bennett III, D. Andre, and M.A. Keane: *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann Publishers, San Francisco, CA 1999.
- [4] W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone: *Genetic Programming - An Introduction*. Morgan Kaufmann Publishers, San Francisco, CA 1998.
- [5] W.B. Langdon: *Genetic Programming and Data Structures*. Kluwer Academic Publishers, Hingham, MA 1998.
- [6] K.E. Kinneer, Jr. and K.L. Kinneer (eds.): *Advances in Genetic Programming*. The MIT Press, Cambridge, MA 1994.
- [7] P.J. Angeline and K.E. Kinneer, Jr. (eds.): *Advances in Genetic Programming 2*. The MIT Press, Cambridge, MA 1996.
- [8] L. Spector, W.B. Langdon, U-M. O'Reilly, and P. Angeline (eds.): *Advances in Genetic Programming 3*. The MIT Press, Cambridge, MA 1999.
- [9] F. Harary: *Graph Theory*. Addison Wesley, Reading, MA 1969.
- [10] R.C. Read: Coding of Unlabeled Trees. In R.C. Read (ed.): *Graph Theory and Computing*. Academic Press, New York 1972.
- [11] V. Kvasnička and J. Pospíchal: Constructive Enumeration of Acyclic Molecules. *Collect. of Czech. Chem. Comm.* **56** (1991) 1777-1802.
- [12] V. Kvasnička, J. Pospíchal, and M. Pelikán: Read's Linear Codes and Evolutionary Computation Over Populations of Rooted Trees. In P. Sinčák (ed.): *Intelligent technologies (II)*, Proceedings of the 1-st Slovak Neural Network Symposium, Herlany, November 1996, pp. 165-184, Elfa, Košice 1996.
- [13] M. Pelikán: *Genetické programovanie*. Práca ŠVOČ, CHTF STU, Máj 1996.
- [14] M. Pelikán, V. Kvasnička, and J. Pospíchal: Read's Linear Codes and Evolutionary Computation Over Populations of Rooted Trees. *Genetic Programming '97*, Stanford, July 13-15, 1997, p. 268, Morgan Kaufmann, San Francisco 1997.

6. Simulované žíhanie

6.1 Úvodné poznámky

Metóda simulovaného žíhania [1,2] (Simulated Annealing - SA) patrí medzi tie stochastické optimalizačné algoritmy, ktoré, ako naznačuje už ich názov, majú základ vo fyzike (na rozdiel od iných stochastických optimalizačných algoritmov, ktoré majú svoj základ väčšinou v biológii). Algoritmus simulovaného žíhania je založený na analógii medzi žíhaním tuhých telies a optimalizačným problémom. Počiatkom 80-tych rokov Kirkpatrick, Gelatt a Vecchi [3] (Watson Research Center of the IBM, USA) a nezávisle Černý [4] (pôsobiaci na Katedre teoretickej fyziky MFF UK v Bratislave) dostali geniálny nápad, že problém hľadania globálneho minima sa môže realizovať podobným spôsobom ako žíhanie tuhého telesa. Tento nový prístup k hľadaniu globálneho minima funkcie bol nazvaný *simulované žíhanie*.



Obrázok 6.1. Znázornenie fyzikálnej realizácie žíhania. Teleso sa vloží do pece (ľavý blok), ktorá je vyhriata na vysokú teplotu T_{max} . Teplota sa programovacím zariadením (pravý blok) pomaly znižuje na teplotu T_{min} . Týmto spôsobom sa odstránia štruktúrne defekty vyskytujúce sa v telese.

Žíhanie vo fyzike označuje taký proces, v ktorom je teleso umiestnené do pece vyhriatej na vysokú teplotu a postupným pomalým znižovaním teploty (pozri obr. 6.1) sa odstraňujú vnútorné defekty telesa. Pri vysokej teplote sú častice telesa (atómy alebo molekuly) náhodne usporiadané v priestore, takže teleso je roztopené. Potom sa teplota postupne znižuje, všetky častice telesa majú možnosť dostať sa do rovnovážnej polohy, čiže energia telesa sa znižuje. Budeme predpokladať, že proces ochladzovania je dostatočne pomalý, potom pre každú teplotu T je žíhané teleso v tepelnej rovnováhe, ktorá je opísaná boltzmannovským rozdelením pravdepodobnosti, že pri teplote T je teleso v stave i s energiou E_i

$$w_T(E_i) = \frac{1}{Q(T)} \exp\left(-\frac{E_i}{kT}\right) \quad (6.1a)$$

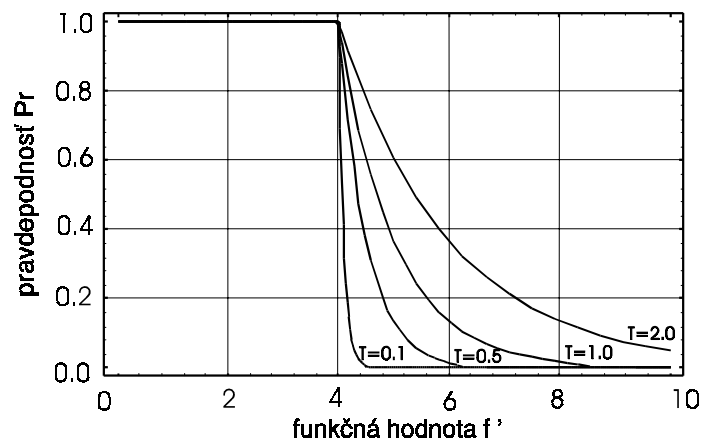
kde k je Boltzmannova konštanta a $Q(T)$ je normalizačný faktor nazývaný *partičná funkcia*

$$Q(T) = \sum_i \exp\left(-\frac{E_i}{kT}\right) \quad (6.1b)$$

kde sčítanie obsahuje všetky stavy i telesa. Ak teplota T klesá, potom boltzmannovská distribúcia uprednostňuje stavy s menšou energiou. V prípade, že sa teplota blíži k nule, stav s minimálnou energiou má nenulovú (jednotkovú) pravdepodobnosť. Žiaľ, ako je dobre známe, ak je ochladzovanie telesa príliš rýchle (telesu nie je umožnené získať tepelnú rovnováhu pre každú teplotu T), defekty v telese môžu "zamrznúť" za vzniku metastabilných štruktúr, ktoré sú vzdialené od mriežkovej štruktúry s najnižšou energiou.

Ako simulovať fyzikálnu evolúciu systému (napr. mnohočasticového kryštálu) smerom k tepelnej rovnováhe pre danú teplotu T ? Metropolis a spol. [5] navrhli metódu Monte Carlo, ktorá simuluje evolúciu systému tak, že generuje postupnosť stavov systému nasledujúcim spôsobom: Nech je daný aktuálny stav systému (určený polohou častíc telesa), potom sa malá náhodná porucha generuje tak, že častice sú "jemne" posunuté. Poznamenajme, že táto porucha musí byť "symetrická", t.j. pravdepodobnosť toho, že malou poruchou sa stav A zmení na stav B, musí byť rovnaká ako pri zmene malou poruchou stavu B na stav A. Ak je rozdiel $\Delta E = E_{\text{perturbed}} - E_{\text{current}}$ medzi porušeným stavom a aktuálnym stavom negatívny ($E_{\text{perturbed}} < E_{\text{current}}$), potom proces pokračuje s novým porušeným stavom. V opačnom prípade, ak $\Delta E \geq 0$, pravdepodobnosť $Pr(\text{perturbed} \leftarrow \text{current})$ akceptovania porušeného stavu určuje exponenciála $\exp(-\Delta E / kT)$ (pozri obr. 6.2)

$$Pr(\text{perturbed} \leftarrow \text{current}) = \min(1, \exp(-\Delta E / kT)) \quad (6.2)$$



Obrázok 6.2. Priebeh Metropolisovho kritéria $Pr = \min(1, \exp(-(f' - f)/T))$ pre rôzne teploty T , kde f je fixná funkčná hodnota ($f=4$) a f' je nezávislá premenná z intervalu $[0, 10]$. Pre klesajúce hodnoty teploty T a pre $f' > f$ pravdepodobnosť $Pr \rightarrow 0$, ak $T \rightarrow 0$.

Toto pravidlo akceptovania porušeného stavu sa nazýva *Metropolisovo kritérium*. Podľa tohto kritéria aplikovaním veľkého počtu porúch a ich akceptovaním do ďalšieho procesu s pravdepodobnosťou (6.2) dostaneme systém v tepelnej rovnováhe, pričom distribúcia pravdepodobnosti rozloženia stavov sa asymptoticky blíži k boltzmannovskej distribúcii (6.1). Tento tvar metódy Monte Carlo sa v štatistickej fyzike nazýva *Metropolisov algoritmus* [5]. K tomu, aby sme formalizovali Metropolisov algoritmus (pozri algoritmus 6.1) zavedieme nasledujúci formalizmus (ktorý je vhodný pre aplikáciu simulovaného žihania na minimalizáciu funkcií): nech je stav systému určený *stavovou premennou* x (vo všeobecnosti vektor obsahujúci mnoho nezávislých reálnych premenných) a energiou $f(x)$. Proces porušenia stavu x na iný stav x' je formálne reprezentovaný stochastickým operátorom,

$\mathbf{x}' = O_{pertur}(\mathbf{x})$. Stochastický charakter tohto operátora spočíva v náhodnej zmene stavu $\mathbf{x}$ na stav $\mathbf{x}'$.

Ako už bolo poznamenané, Metropolisov algoritmus [5] produkuje distribúciu pravdepodobnosti stavov, ktorá sa asymptoticky blíži k boltzmannovskej distribúcii (6.1)

$$w_T(\mathbf{x}) = \frac{1}{Q(T)} \exp\left(-\frac{f(\mathbf{x})}{T}\right) \quad (6.3a)$$

$$Q(T) = \sum_{\mathbf{x}} \exp\left(-\frac{f(\mathbf{x})}{T}\right) \quad (6.3b)$$

kde súčet obsahuje všetky stavy $\mathbf{x}$. Pre jednoduchosť sa Boltzmannova konštanta k teraz vynechala (pozri rovnice (6.1a-b), formálne bola táto konštanta zahrnutá do teploty T).

```

procedure Metropolis_algorithm(input:  $\mathbf{x}_{ini}$ ,  $k_{max}$ ,  $T$ ; output:  $\mathbf{x}_{out}$ );
begin  $k := 0$ ;  $\mathbf{x} := \mathbf{x}_{ini}$ ;
    while  $k < k_{max}$  do
        begin  $k := k + 1$ ;
             $\mathbf{x}' := O_{pertur}(\mathbf{x})$ ;
             $Pr := \min(1, \exp(-(f(\mathbf{x}') - f(\mathbf{x}))/T))$ ;
            if random <  $Pr$  then  $\mathbf{x} := \mathbf{x}'$ ;
        end;
         $\mathbf{x}_{out} := \mathbf{x}$ ;
    end;

```

Algoritmus 6.1. Implementácia Metropolisovho algoritmu. Procedúra sa inicializuje tak, že počiatočný stav sa položí rovný vstupnému stavu $\mathbf{x}_{ini}$, opakuje sa k_{max} -krát (toto číslo musí byť dostatočne veľké, aby sa dosiahla tepelná rovnováha). Symbol O_{pertur} modifikuje aktuálny stav $\mathbf{x}$ na $\mathbf{x}'$. Akceptovanie nového stavu sa rieši pomocou Metropolisovho kritéria realizovaného pre teplotu T . Po skončení Metropolisovho algoritmu je výstupným stavom $\mathbf{x}_{out}$ posledný stav $\mathbf{x}$.

Metropolisov algoritmus sa môže použiť na počítačovú simuláciu žihania. V tomto prístupe sa simulované žihanie chápe ako postupnosť Metropolisových algoritmov realizovaných pre postupnosť vhodne sa znižujúcich teplôt, pričom výstupný stav $\mathbf{x}_{out}$ z posledného Metropolisovho algoritmu slúži ako vstupný stav $\mathbf{x}_{in}$ pre nasledujúci Metropolisov algoritmus. Táto procedúra sa inicializuje tak, že teplota T sa položí rovná maximálnej teplote T_{max} a Metropolisov algoritmus sa aplikuje tak dlho (k_{max} -krát, kde k_{max} je parameter Metropolisovho algoritmu), až sa dosiahne tepelná rovnováha. Potom sa teplota zníži (napr. pravidlom $T := \alpha T$, kde $0 < \alpha < 1$) a znovu sa aplikuje Metropolisov algoritmus, pričom sa predpokladá, že sa dosiahne tepelná rovnováha. Algoritmus sa ukončí vtedy, keď teplota T dosiahne minimálnu hodnotu T_{min} , pre ktorú už žiadny stav s vyššou funkcionálnou hodnotou nie je akceptovaný (skoro nulová pravdepodobnosť akceptovania). "Zamrznutý" stav $\mathbf{x}_{out}$ sa potom chápe ako výsledné riešenie (pozri algoritmus 6.2).

Jeden zo základných problémov simulovaného žihania je špecifikácia teplôt T_{max} , T_{min} a pravidla pre znižovanie teploty T . V literatúre boli navrhnuté rôzne schémy, ako určiť tieto parametre simulovaného žihania [1,2,6]. Pretože náplňou tejto kapitoly nie je podrobne študovať teóriu simulovaného žihania, budeme používať najjednoduchší prístup pre ochladzovanie (multiplikatívny prístup pomocou parametra α) a počiatočná teplota je zvolená tak, aby približne 50% porušených stavov bolo akceptovaných Metropolisovým algoritmom.

```

procedure Simulated_annealing(input:  $T_{\min}, T_{\max}, k_{\max}, \alpha$ ; output:  $\mathbf{x}_{\text{opt}}$ );
begin  $\mathbf{x}_{\text{ini}}$  := náhodne vygenerovaný stavový vektor;
       $T := T_{\max}$ ;
      while  $T > T_{\min}$  do
        begin Metropolis_algorithm( $\mathbf{x}_{\text{ini}}, \mathbf{x}_{\text{out}}, k_{\max}, T$ );
           $\mathbf{x}_{\text{ini}} := \mathbf{x}_{\text{out}}$ ;
           $T := \alpha * T$ ;
        end;
       $\mathbf{x}_{\text{opt}} := \mathbf{x}_{\text{out}}$ ;
end;

```

Algoritmus 6.2. Implementácia simulovaného žihania, vstupné parametre sú $T_{\min}$, $T_{\max}$, $k_{\max}$, α , výstupný parameter je $\mathbf{x}_{\text{opt}}$. Algoritmus sa inicializuje náhodným vygenerovaním počiatočného stavu $\mathbf{x}_{\text{ini}}$ a maximálnou teplotou $T_{\max}$. Cyklus **while** sa opakuje tak dlho, pokiaľ platí $T > T_{\min}$, teplota T sa znižuje pomocou parametra α podľa vzťahu $T := \alpha * T$. Po ukončení cyklu **while** sa výsledný stav $\mathbf{x}_{\text{out}}$ považuje za výsledné riešenie označené $\mathbf{x}_{\text{opt}}$.

Úloha 6.1. Implementujte algoritmus simulovaného žihania (pozri algoritmus 6.2). Algoritmus špecifikujte pre funkciu n premenných, ktoré sú reprezentované binárne. Operátor mutácie O_{mut} je potom totožný s rovnako pomenovaným operátorom z genetického algoritmu (pozri (3.2) a algoritmus 3.2).

Hlavný cieľ simulovaného žihania je nájsť globálne riešenie optimalizačného problému

$$\mathbf{x}_{\text{opt}} = \arg \min_{\mathbf{x} \in D} f(\mathbf{x}) \quad (6.4)$$

kde $f(\mathbf{x})$ je reálna funkcia definovaná nad oblasťou D (ktorá je obvykle konečná a diskretná) a $\mathbf{x}_{\text{opt}}$ je hodnota vektora nezávislých premenných, ktorá odpovedá globálnemu minimu funkcie $f(\mathbf{x})$ nad doménou D . Premenná $\mathbf{x}$ sa teraz chápe ako stavový vektor hypotetického fyzikálneho systému a hodnota funkcie $f(\mathbf{x})$ vyjadruje jeho "energiu". Potom sa vyššie uvedený optimalizačný problém (6.4) môže úspešne študovať metódou simulovaného žihania. Parameter "teplota" bude v tomto prístupe hrať úlohu riadiaceho parametra metódy.

6.2 Vzťah k štatistickej fyzike

V predchádzajúcej časti tejto kapitoly bola naznačená blízka analógia medzi žiháním tuhého telesa (vo všeobecnosti fyzikálneho systému obsahujúceho mnoho častíc, ktoré medzi sebou interagujú) a riešením optimalizačného problému (6.4). Fyzikálny proces žihania sa môže úspešne modelovať na počítači pomocou Metropolisovho algoritmu. Tento algoritmus je založený na štatistickej fyzike, ktorá sa môže chápať ako centrálna disciplína pre fyziku tuhej fázy. Ukážeme, že existuje veľmi tesný vzťah medzi optimalizačným problémom (6.4) a štatistickou fyzikou. Táto aplikácia simulovaného žihania nie je zaujímavá len ako určitá analógia, ale aj ako vhodný rámec pre štúdium konvergenčných vlastností simulovaného žihania.

Základný predpoklad štatistickej fyziky [7] je, že fyzika mnohočasticových systémov je kompatibilná so štatistickými súbormi a že časové stredné hodnoty mechanických veličín systému v rámci mikroskopickej rovnováhy sú rovné odpovedajúcim stredným hodnotám nad súbormi. V štatistickej fyzike sa definovali vhodné veličiny popisujúce fyzikálne vlastnosti systému, a tiež sa ukázalo, ako počítať tieto veličiny pomocou rovnovážneho rozdelenia stavov systému. Ukazuje sa, že v tepelnej rovnováhe je rozdelenie pravdepodobnosti určené boltzmannovským rozdelením (6.1a-b). Vzťah medzi štatistickou fyzikou a optimalizačným

problémom (6.4) sa môže teraz študovať vo veľmi špecifickej rovine. Predpokladajme, že máme hypotetický fyzikálny systém v tepelnej rovnováhe, ktorého vnútorné stavy x (premenne z oblasti D) majú distribúciu podľa vzťahu (6.3a-b) (formálne ekvivalentný s (6.1a-b)), potom pre tento optimalizačný problém môžeme definovať rôzne fyzikálne veličiny podobným spôsobom ako v štatistickej fyzike.

Uvažujem funkciu f definovanú nad diskretnou a konečnou oblasťou D

$$f : D \rightarrow A \subset R_+ \quad (6.5)$$

kde A je oblasť (diskrétna a konečná podmnožina R_+ obsahujúca nezáporné reálne čísla) funkčných hodnôt f . Rozdelenie pravdepodobnosti stavu x (t.j. premennej x z D), ktoré je výsledkom Metropolisovho algoritmu realizovaného pre teplotu T , je určené vzťahmi (6.3a-b). Pre naše ďalšie úvahy bude vhodné poznať hustotu funkčných hodnôt $y=f(x)$, ktorá je určená takto

$$w_T(y) = \frac{|D(y)|}{Q(T)} \exp\left(-\frac{y}{T}\right) \quad (6.6a)$$

$$Q(T) = \sum_{y \in A} |D(y)| \exp\left(-\frac{y}{T}\right) \quad (6.6b)$$

kde $D(y) \subset D$ je množina obsahujúca všetky $x \in D$, pre ktoré platí $y=f(x)$

$$D(y) = \{x \in D; y = f(x)\} \quad (6.6c)$$

a $|D(y)|$ je jej kardinalita. Je jasné, že veličina $Q(T)$ určená (6.6b) je identická s jej analógom (6.3b). Nasledujúce "makroskopické" veličiny sú definované takto

(1) *Stredná hodnota* funkcie $f(x)$

$$\langle f \rangle_T = \sum_{y \in A} y w_T(y) \quad (6.7)$$

(2) *Stredná hodnota* funkcie $f^2(x)$

$$\langle f^2 \rangle_T = \sum_{y \in A} y^2 w_T(y) \quad (6.8)$$

(3) *Disperzia* funkcie $f(x)$

$$\sigma^2(T) = \sum_{y \in A} w_T(y) (\langle f \rangle_T - y)^2 = \langle f^2 \rangle_T - \langle f \rangle_T^2 \quad (6.9)$$

(3) *Entropia*

$$S(T) = - \sum_{y \in A} w_T(y) \ln \left(\frac{w_T(y)}{|D(y)|} \right) \quad (6.10)$$

(4) *Špecifické teplo*

$$C(T) = \frac{\partial}{\partial T} \langle f \rangle_T = \frac{\sigma^2(T)}{T^2} = T \frac{\partial S(T)}{\partial T} \quad (6.11)$$

Tento posledný vzťah sa môže jednoducho odvodiť derivovaním buď (6.7) alebo (6.10) podľa teploty T .

Hustota funkčných hodnôt (6.6a-b) vyhovuje týmto asymptotickým podmienkam

$$\lim_{T \rightarrow \infty} w_T(y) = w_\infty(y) = \frac{1}{|D|} |D(y)| \quad (6.12a)$$

$$\lim_{T \rightarrow 0} w_T(y) = w_0(y) = \delta(y, y_{opt}) = \begin{cases} 1 & (\text{pre } y = y_{opt}) \\ 0 & (\text{pre } y \neq y_{opt}) \end{cases} \quad (6.12b)$$

kde $y_{opt}=f(x_{opt})$ je optimálna hodnota funkcie $f(x)$ odpovedajúca bodu určenému vzťahom (6.4) ako globálne minimum. Druhý asymptotický vzťah (6.12b) je veľmi dôležitý pre metódu simulovaného žihania; ak sa podľa tohto vzťahu teplota T blíži k nule (cez rovnovážne stavy), potom *pravdepodobnosť výskytu stavu odpovedajúcemu globálnemu minimu je jednotková*.

Parciálna derivácia $\langle f \rangle_T$ podľa teploty T je určená vzťahom (6.11), potom

$$\frac{\partial \langle f \rangle_T}{\partial T} = \frac{\sigma^2(T)}{T^2} > 0 \quad (6.13)$$

To znamená, že stredná hodnota $\langle f \rangle_T$ klesá, ak aj teplota T klesá, pomocou (6.12a-b) dostaneme

$$\lim_{T \rightarrow \infty} \langle f \rangle_T = \langle f \rangle_\infty = \frac{1}{|D|} \sum_{y \in A} |D(y)| y \quad (6.14a)$$

$$\lim_{T \rightarrow 0} \langle f \rangle_T = \langle f \rangle_0 = y_{opt} \quad (6.14b)$$

Stredná hodnota $\langle f^2 \rangle_T$ vyhovuje podobným asymptotickým podmienkam

$$\lim_{T \rightarrow \infty} \langle f^2 \rangle_T = \langle f^2 \rangle_\infty = \frac{1}{|D|} \sum_{y \in A} |D(y)| y^2 \quad (6.15a)$$

$$\lim_{T \rightarrow 0} \langle f^2 \rangle_T = \langle f^2 \rangle_0 = y_{opt}^2 \quad (6.15b)$$

Pre disperziu (6.9) dostaneme

$$\lim_{T \rightarrow \infty} \sigma^2(T) = \sigma^2(\infty) = \langle f^2 \rangle_\infty - \langle f \rangle_\infty^2 \quad (6.16a)$$

$$\lim_{T \rightarrow 0} \sigma^2(T) = \sigma^2(0) = \langle f^2 \rangle_0 - \langle f \rangle_0^2 = y_{opt}^2 - (y_{opt})^2 = 0 \quad (6.16b)$$

Entropia (6.10) vyhovuje dvom dobre známym asymptotickým podmienkam

$$\lim_{T \rightarrow \infty} S(T) = S(\infty) = \ln |D| \quad (6.17a)$$

$$\begin{aligned} \lim_{T \rightarrow 0} S(T) &= S(0) = \ln |D(y_{opt})| \\ &= \ln 1 = 0 \quad (\text{pre } |D(y_{opt})| = 1) \end{aligned} \quad (6.17b)$$

Dôkaz týchto podmienok sa jednoducho uskutoční substitúciou (6.6) do (6.10) a potom asymptotickou zmenou teploty buď $T \rightarrow \infty$, alebo $T \rightarrow 0$. Druhá vlastnosť (6.17b) je vo fyzike známa ako tretí zákon termodynamický. V prípade simulovaného žihania sa entropia môže interpretovať ako *kvantitatívna miera stupňa optimality*.

V dôsledku platnosti (6.11) sa parciálna derivácia entropie podľa teploty T určí takto

$$\frac{\partial S(T)}{\partial T} = \frac{\sigma^2(T)}{T^3} > 0 \quad (6.18)$$

Potom s klesajúcou teplotou T (cez rovnovážne stavy) musí klesať aj entropia S , $S(T) \rightarrow 0$ pre $T \rightarrow 0$ (pozri rov. (6.17b)). V okolí bodu $T=0$ sa entropia $S(T)$ môže aproximovať mocninným rozvojom teploty T

$$S(T) = A_0 + A_1 T + A_2 T^2 + \dots \quad (6.19)$$

kde $A_0 = \ln(|D(y_{opt})|)$, a $A_1 > 0$. Potom sa tepelná kapacita $C(T)$ v okolí bodu $T=0$ (pozri rov. (6.11)) aproximuje takto

$$C(T) = T \frac{\partial S(T)}{\partial T} = T (A_1 + 2A_2 T + \dots) \quad (6.20)$$

V dôsledku platnosti tohto vzťahu a asymptotickej vlastnosti (6.16a) disperzie spĺňa tepelná kapacita $C(T)$ tieto dve asymptotické podmienky

$$\lim_{T \rightarrow \infty} C(T) = \lim_{T \rightarrow 0} C(T) = 0 \quad (6.21)$$

To znamená, že $C(T)$ musí mať pre $0 < T < \infty$ aspoň jedno maximum a ak sa teplota asymptoticky blíži k nule alebo k nekonečnu, tepelná kapacita $C(T)$ sa blíži k nule.

6.3 Modelové výpočty

Nech je zobrazenie $f : D = \{0,1\}^N \rightarrow A \subset \mathbb{R}_+$ s definičným oborom obsahujúcim binárne vektory dĺžky N definované takto

$$f(\mathbf{x}) = f(x_1, x_2, \dots, x_N) = \begin{cases} |\mathbf{x}| + 1 & (\text{pre } |\mathbf{x}| \leq p) \\ N - |\mathbf{x}| & (\text{pre } |\mathbf{x}| > p) \end{cases} \quad (6.22)$$

kde $|\mathbf{x}| = \sum x_i$ je norma L_1 vektora $\mathbf{x} \in \{0,1\}^N$, $0 \leq |\mathbf{x}| \leq N$. Parameter p je ohraničený podmienkou $0 \leq p \leq N$. Pre $p < N$ má táto funkcia dve minimá, prvé minimum je $f(\mathbf{x})=1$ pre $|\mathbf{x}|=0$, druhé (globálne) minimum je $f(\mathbf{x})=0$ pre $|\mathbf{x}|=N$. Pre $p=N$ má funkcia len jedno (t.j. globálne) minimum $f(\mathbf{x})=1$ pre $|\mathbf{x}|=0$. Množina A funkčných hodnôt obsahuje nezáporné celé čísla, pričom $A = \{0, 1, 2, \dots, f_{max}\}$, kde $f_{max} = \max(p+1, N-p-1)$. Priebeh tejto funkcie je pre $N=10$ a pre $p=4$ a $p=7$ znázornený na obrázku 6.3. Parameter p určuje stupeň *deceptívnosti* (falošnosti) tejto funkcie. Ak je p o trochu menšie ako N , potom funkcia obsahuje dlhé "úpätie" začínajúce v $|\mathbf{x}|=0$ a končiac v $|\mathbf{x}|=p$, za týmto úpäťom (pre $p+1 \leq |\mathbf{x}| \leq N$) funkcia rýchle klesá k nule (pozri obr. 6.3, diagram B). Táto vlastnosť funkcie $f(\mathbf{x})$ sa nazýva *deceptívnosť*, predstavuje ťažkú úlohu pre evolučné metódy, ak majú hľadať globálne minimum.

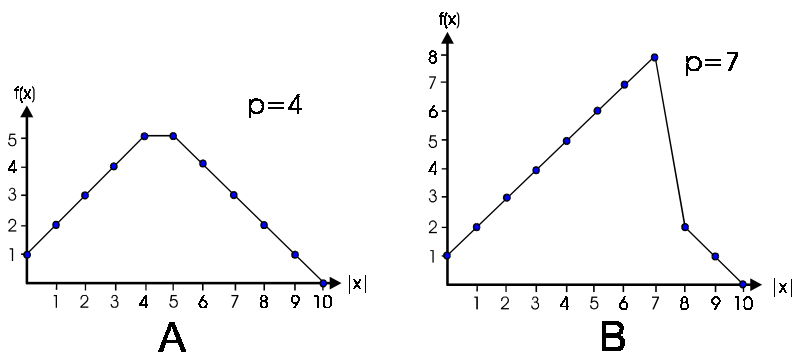
Operátor $O_{perturb}$ z Metropolisovho algoritmu transformujúci binárny vektor $\mathbf{x} \in \{0,1\}^N$ na iný binárny vektor $\mathbf{x}' \in \{0,1\}^N$ sa v tomto prípade realizuje pomocou operátora mutácie $O_{mut}^{(P_{mut})}$ (terminológia je prevzatá z genetického algoritmu)

$$\mathbf{x}' = (x'_1, x'_2, \dots, x'_N) = O_{mut}^{(P_{mut})}(\mathbf{x}) = O_{mut}^{(P_{mut})}(x_1, x_2, \dots, x_N) \quad (6.23)$$

kde sú jednotlivé zložky $\mathbf{x}'$ určené takto

$$\forall i : x'_i = \begin{cases} 1 - x_i & (\text{ak } random < P_{mut}) \\ x_i & (\text{ostatné prípady}) \end{cases} \quad (6.24)$$

kde P_{mut} je pravdepodobnosť "mutácie" premennej x_i (pre $\forall i$) na jej komplement (ak $random < P_{mut}$) alebo zostane nezmenená (ostatné prípady) a $random$ je náhodné číslo z intervalu $[0,1)$ s rovnomerným rozdelením pravdepodobnosti. Pravdepodobnosť P_{mut} je veľmi malá ($0 < P_{mut} \ll 1$), v opačnom prípade by mutačný operátor produkoval stavy, ktoré sú veľmi vzdialené od pôvodného stavu $\mathbf{x}$.



Obrázok 6.3. Priebeh funkcie (6.22) pre $N=10$ a $p=4, 7$. Pre $p=7$ funkcia obsahuje "dlhé" úpätie, za ktorým je oblasť prudkého spádu do globálneho minima.

Aby sme mohli vypočítať makroskopické veličiny definované v predchádzajúcej kapitole 6.2, musíme poznať kardinality množín $D(y)$ pre funkciu (6.6). Tak napr. pre $N=10$ a pre $p=4$ a $p=7$ sú tieto kardinality určené takto

$$\begin{aligned}
 p=4 : & |D(0)|=1, \quad |D(1)|=11, \quad |D(2)|=55, \quad |D(3)|=165, \quad |D(4)|=330, \quad |D(5)|=462 \\
 p=7 : & |D(0)|=1, \quad |D(1)|=11, \quad |D(2)|=55, \quad |D(3)|=45, \quad |D(4)|=120, \quad |D(5)|=210, \\
 & |D(6)|=252, \quad |D(7)|=210, \quad |D(8)|=120
 \end{aligned}$$

Priebehy makroskopických veličín pre funkciu $f(\mathbf{x})$ špecifikovanú konštantami $N=10$ a $p=1,4,7,9$ sú znázornené na obr. 6.4, diagramy A - E. Ako z tohto obrázka vyplýva, sú splnené všetky asymptotické vlastnosti, ktoré boli predpovedané v predchádzajúcej časti tejto kapitoly pre makroskopické veličiny. Predovšetkým stredná hodnota $\langle f \rangle_T \rightarrow 0$ pre $T \rightarrow 0$. To znamená, že pre malé teploty je s jednotkovou pravdepodobnosťou výskytu dominantný stav s nulovou funkčnou hodnotou.

Pre simulované žihanie realizované na počítači sa výpočet makroskopických veličín môže uskutočniť pomocou približných hustôt funkčných hodnôt, ktoré sú aproximované výskytom funkčných hodnôt akceptovaných Metropolisovým algoritmom pre konštantnú teplotu T . Nech $\chi_T(y)$ je počet výskytov stavu x , ktorý odpovedá funkčnej hodnote y a ktoré sa vyskytujú v Metropolisovom algoritme pre danú teplotu T . Potom môže byť hustota $w_T(y)$ približne určená takto

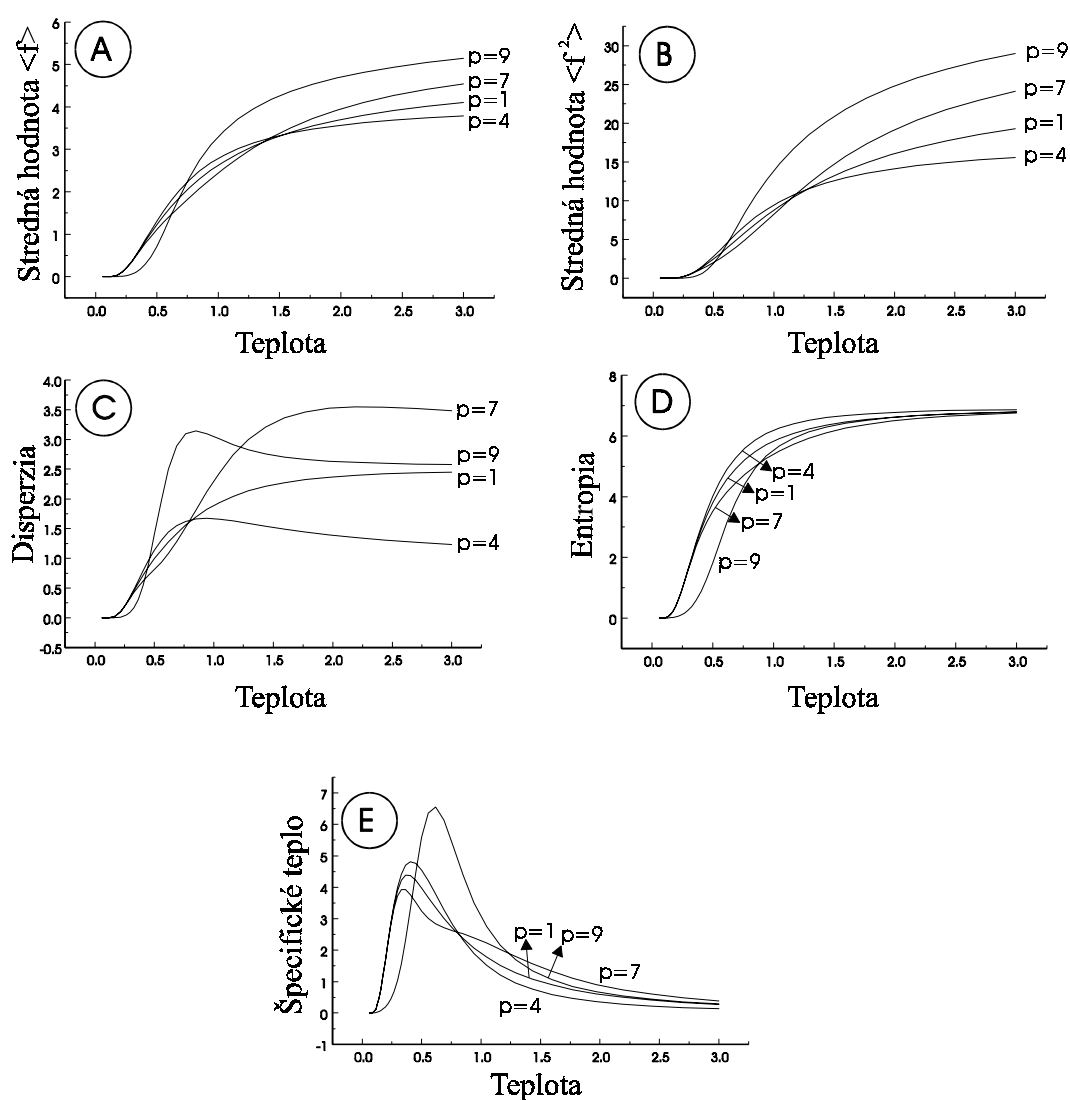
$$w_T(y) \approx \frac{\chi_T(y)}{\sum_{y' \in A} \chi_T(y')} \quad (6.25)$$

Pre dostatočne veľké k_{max} by táto aproximácia mala poskytovať výsledky, ktoré sú veľmi blízke k presným hodnotám $w_T(y)$. Priebehy makroskopických vlastností (6.7-11) s hustotami aproximovanými vzťahom (6.25) sú znázornené na obr. 6.5, diagramy A - E. Parametre simulovaného žihania použité vo výpočtoch mali tieto hodnoty

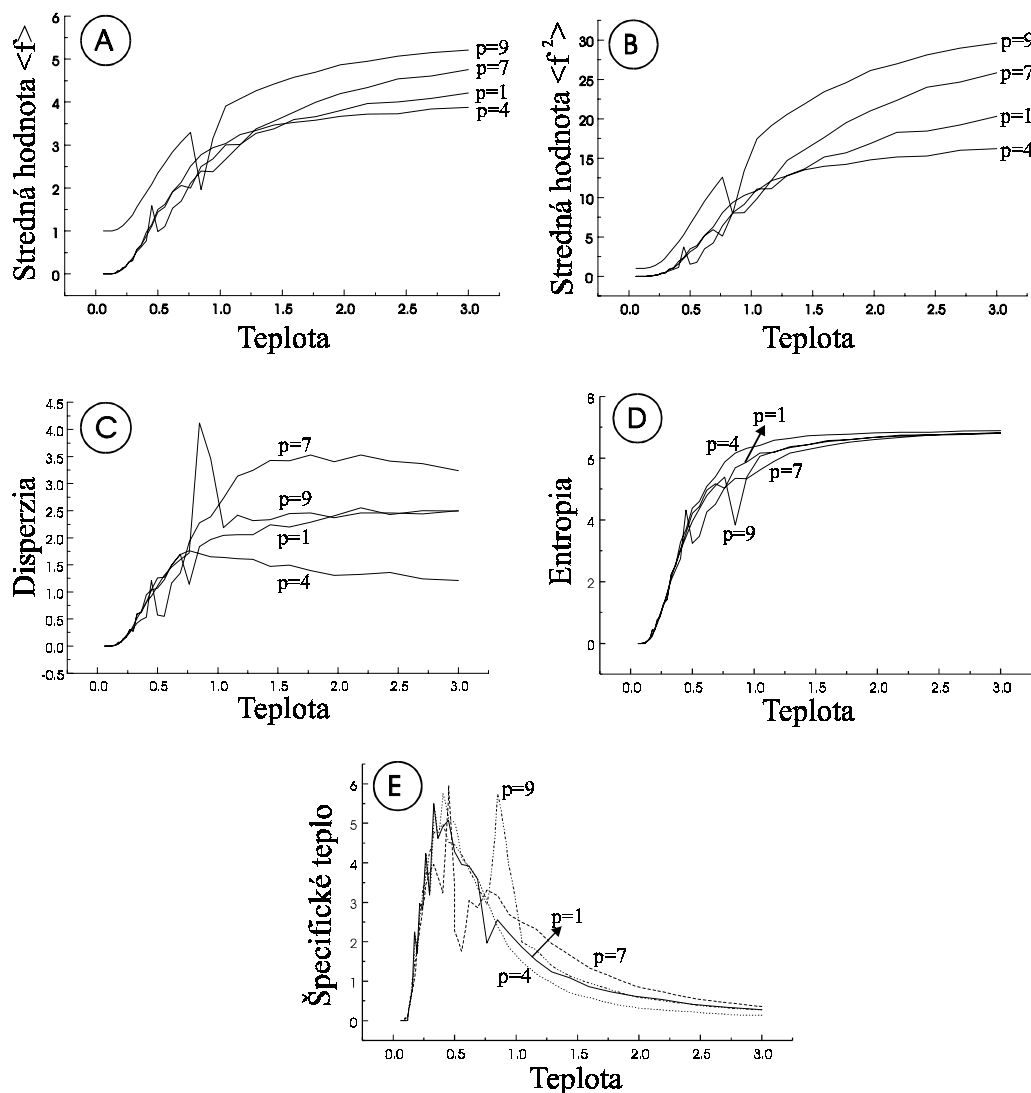
$$k_{max}=10^4, P_{mut}=0.1, T_{max}=3, T_{min}=3/50, \text{ a } \alpha=0.95 \quad (6.26)$$

Vidíme, že simulované žihanie pre $p=1,4$ a 7 poskytuje správne výsledky pre strednú hodnotu $\langle f \rangle_T$ (t.j. $\langle f \rangle_T \rightarrow 0$ pre $T \rightarrow 0$), ale pre $p=9$ (funkcia $f(\mathbf{x})$ má najväčší stupeň deceptivnosti) metóda simulovaného žihania nie je schopná poskytnúť korektný výsledok (dostali sme $\langle f \rangle_T \rightarrow 1$ pre $T \rightarrow 0$). To znamená, že pre $p=9$ je simulované žihanie schopné nájsť len suboptimálne riešenie $x=(0,0,\dots,0)$, $|x|=0$, pre ktoré platí $f(x)=1$. Existuje interval kritických teplôt $[0.7,0.9]$, nad ktorým metóda poskytuje hustoty stavu $y=0$, ktoré sú blízke

teoretickým hodnotám, avšak vo vnútri intervalu bol výskyt stavov s $y=0$ skoro nulový. To znamená, že pre nižšie teploty nie je Metropolisov algoritmus schopný vytvárať stavy s $y=0$ (v dôsledku širokej bariéry funkcionálnych hodnôt, pozri obr. 6.3, diagram B). Túto skutočnosť indikujú napr. vysoké maximá pre disperziu a špecifické teplo, ktoré sa môžu fyzikálne interpretovať ako "fázové prechody" medzi konfiguráciami, ktoré sú oddelené vysokými bariérami funkčných hodnôt. To znamená, že tieto dve makroskopické veličiny sa môžu použiť ako vhodný indikátor tých tepelných intervalov, kde znižovanie teploty T musí byť veľmi pomalé, aby sa úspešne prekonal vyššie spomínaný "fázový prechod".



Obrázok 6.4. Priebehy makroskopických veličín vypočítaných teoreticky pomocou vzorcov (6.7-11) pre parametre $p = 1, 4, 7, 9$ (diagramy A - E).



Obrázok 6.5. Priebeh makroskopických veličín vypočítaných metódou simulovaného žihania, ktorá je špecifikovaná parametrami (6.26). Pre prípad s najväčšou deceptívnosťou ($p=9$), odpovedá finálny stav $\mathbf{x}_{opt}$ jednotkovej funkčnej hodnote (lokálne minimum), t.j. metóda nebola schopná poskytnúť globálne minimum.

Uvedieme niekoľko vysvetľujúcich poznámok k simulovanému žihaniu aplikovanému na deceptívnu funkciu $f(\mathbf{x})$ s hodnotou parametra $p=9$. Ukázali sme, že simulované žihanie s parametrami (6.26) nie je schopné nájsť správne riešenie. Vo všeobecnosti sa táto neschopnosť môže jednoducho prekonať tak, že pravdepodobnosť P_{mut} sa podstatne zvýši, napr. $P_{mut}=0.5$. Avšak potom sa simulované žihanie nemôže chápať ako evolučná optimalizačná metóda, prehľadávanie priestoru riešení je viac náhodné ako systematické. Simulované žihanie je pre vysoké hodnoty pravdepodobnosti P_{mut} redukované na "slepé" hľadanie. Pre malé dimenzie môže byť tento prístup úspešný, pre veľké dimenzie je tento prístup úplne beznádejný. V nasledujúcej časti tejto kapitoly uvedieme také dve modifikácie

štandardnej verzii simulovaného žihania, ktoré budú schopné poskytnúť správne riešenie aj pre vysoko deceptívne funkcie a súčasne bude zachovaný evolučný charakter metódy.

6.3.1 Simulované žihanie s elitizmom

V originálnej verzii simulovaného žihania slúži výsledný stav z aplikácie Metropolisovho algoritmu (realizovaného pri teplote T) ako počiatočný stav nasledujúceho Metropolisovho algoritmu (realizovaného pri zníženej teplote αT). Tento základný predpoklad simulovaného žihania bude teraz modifikovaný tak, že Metropolisov algoritmus sa inicializuje najlepším riešením, ktoré sa získalo v priebehu predchádzajúcej histórie metódy (t.j. pre epochy simulovaného žihania s vyššími teplotami ako súčasná teplota); takúto jednoduchú modifikáciu simulovaného žihania budeme nazývať *simulované žihanie s elitizmom* [8]. Modifikácia procedúr Metropolis_algorithm a Simulated_annealing, ktoré boli uvedené v predchádzajúcej časti tejto kapitoly, sa realizuje algoritmami 6.3 a 6.4.

```

procedure Elitism_Metropolis_algorithm(input : $x_{in}$ ,  $k_{max}$ ,  $T$ ;
                                         output: $x_{opt}$ );
begin  $k:=0$ ;  $x:=x_{in}$ ;  $x_{opt}:=x_{in}$ ;
      while  $k < k_{max}$  do
        begin  $k:=k+1$ ;
           $x' := O_{pertur}(x)$ ;
           $Pr := \min(1, \exp(-(f(x') - f(x))/T))$ ;
          if random $<Pr$  then
            begin  $x:=x'$ ;
              if  $f(x') < f(x_{opt})$  then  $x_{opt}:=x'$ ;
            end;
          end;
        end;
      end;

```

Algoritmus 6.3. Implementácia Metropolisovho algoritmu so zahrnutým elitizmom. Vstupné parametre sú x_{in} , k_{max} , T . Algoritmus sa inicializuje vstupným vektorom x_{in} , najlepšie riešenie je uložené do x_{opt} , výsledná hodnota tohto vektora slúži ako výstupný parameter procedúry.

```

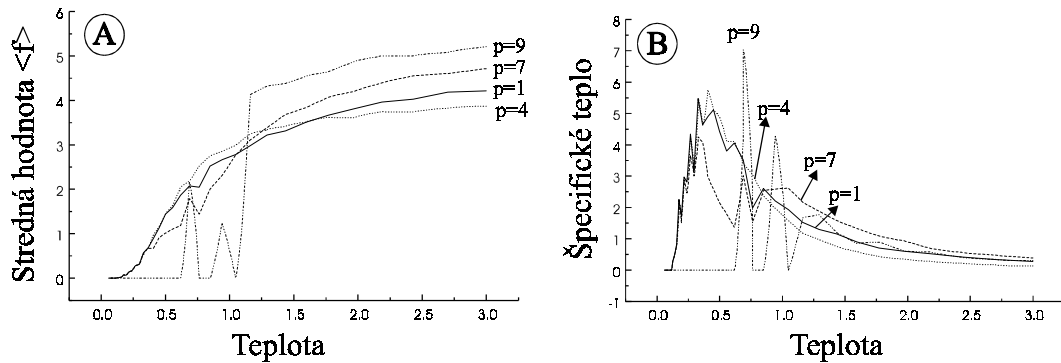
procedure Elitism_Simulated_annealing(input : $T_{min}$ ,  $T_{max}$ ,  $k_{max}$ ,  $\alpha$ ;
                                         output: $x_{opt}$ );
begin  $x_{opt} :=$  náhodne vygerovaný stavový vektor;
       $T := T_{max}$ ;
      while  $T > T_{min}$  do
        begin Elitism_Metropolis_algorithm( $x_{opt}$ ,  $k_{max}$ ;  $T$ );
           $T := \alpha * T$ ;
        end;
      end;

```

Algoritmus 6.4. Implementácia simulovaného žihania s elitizmom. Vstupné parametre do procedúry sú T_{min} , T_{max} , k_{max} , α a výstupný parameter je x_{opt} . Náhodne inicializovaný stav x_{opt} zaznamenáva najlepšie riešenie vyskytujúce sa v celej histórii algoritmu.

Vo všeobecnosti nespĺňa simulované žihanie s elitizmom dôležitú podmienku metódy, že konštanta k_{max} je dostatočne veľká a že pre diskrétny kroky teploty je Metropolisov algoritmus v tepelnej rovnováhe (t.j. stavy majú distribúciu podľa Boltzmannu (6.3)). Pretože v každom kroku zníženia teploty sa Metropolisov algoritmus inicializuje najlepším riešením,

ktoré bolo zaznamenané v celej predchádzajúcej histórii algoritmu, môže nastať taká situácia (predovšetkým pre nižšie teploty), že výsledné riešenie produkované posledným behom Metropolisovho algoritmu odpovedá lokálnemu minimu, ktoré je podstatne odlišné od globálneho minima.



Obrázok 6.6. Priebeh strednej hodnoty $\langle f \rangle$ a špecifického tepla, ktoré sa spočítali pomocou simulovaného žihania s elitizmom. Pre všetky hodnoty parametra p odpovedá výsledný stav x_{opt} globálnemu minimu s nulovou funkčnou hodnotou (pripomeňme, že štandardná verzia simulovaného žihania poskytuje pre $p=9$ len lokálne minimum s jednotkovou funkčnou hodnotou).

Numerické výsledky získané simulovaným žiháním s elitizmom pre funkciu (6.22) špecifikovanú hodnotami $N=10$ a $p=1, 4, 7, 9$ pre strednú hodnotu $\langle f \rangle$ a špecifickú teplotu, sú znázornené na obr. 6.6, diagramy A a B. Vidíme, že pre $p=9$ (najdeceptívnejší tvar funkcie) sú zmeny na priebehu veľmi dramatické. Pre stredné teploty nebol Metropolisov algoritmus schopný udržať nenulovú distribúciu stavu s nulovou funkčnou hodnotou. Pretože najlepšie riešenie s nulovou funkčnou hodnotou sa zaznamenalo už v prvých etapách simulovaného žihania, Metropolisov algoritmus produkuje pre malé teploty ($T < 0.6$) akceptované stavy s nulovou funkčnou hodnotou, ktoré odpovedajú globálnemu minimu.

Slabšia verzia simulovaného žihania s elitizmom je založená na požiadavke, aby sa Metropolisov algoritmus inicializoval najlepším riešením získaným len v predchádzajúcom kroku, a nie v celej predchádzajúcej histórii algoritmu. Žiaľ, tento prístup k najdeceptívnejšej forme optimalizovanej funkcie pre $p=9$ nebol úspešný pri prekonávaní veľkej bariéry so spádom do lokálneho minima (s jednotkovou funkčnou hodnotou).

6.3.2 Paralelné simulované žihanie

Táto verzia simulovaného žihania sa s úspechom použila na riešenie komplikovaných kombinatorických grafovo-teoretických problémov [9,10], kde štandardná verzia simulovaného žihania nie je schopná poskytnúť správne globálne riešenie. Základná idea paralelného simulovaného žihania spočíva v súčasnom aplikovaní simulovaného žihania na množinu P stavov $x, x', x'', \dots$ ktoré sú "synchronizované" tak, že Metropolisove algoritmy bežiace nad nimi majú rovnakú teplotu. Aby stavy medzi sebou interagovali, zaviedla sa operácia *kríženia* medzi stavmi podobným spôsobom ako v genetických algoritmoch. Nech $x, x' \in P$ sú dva náhodné vybrané stavy, operácia kríženia vytvorí z týchto dvoch "rodičovských" stavov dva nové stavy - potomkov

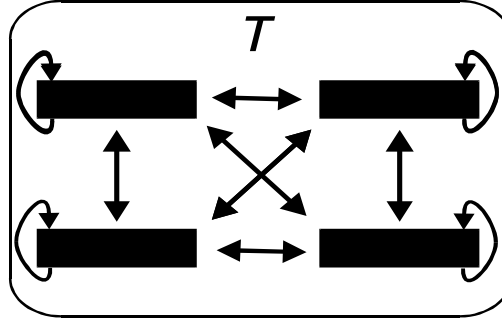
$$(z, z') = O_{cross}(x, x') \quad (6.27)$$

Formálne, nech $x = (x_1, x_2, \dots, x_N)$ and $x' = (x'_1, x'_2, \dots, x'_N)$ sú dva rodičovské stavy vyjadrené ako reťazce znakov dĺžky N , potom potomkovia sú definovaní takto

$$z = (x_1, \dots, x_r, x'_{r+1}, \dots, x'_N) \quad (6.28a)$$

$$z' = (x'_1, \dots, x'_r, x_{r+1}, \dots, x_N) \quad (6.28b)$$

kde r je náhodne vybraný bod križenia, $1 < r < N$.



Obrázok 6.7. Schematické znázornenie paralelného simulovaného žihania. Na jednotlivé stavy (znázornené čiernymi blokmi) sa aplikovali nezávislé Metropolisove algoritmy s rovnakou teplotou T . Šípky znázorňujú, že stavy medzi sebou interagujú s malou pravdepodobnosťou P_{cross} .

```

procedure Parallel_Metropolis_algorithm(input:  $P_{ini}, k_{max}, T$ ;
                                         output:  $P_{out}$ );
begin  $k := 0$ ;  $P := P_{ini}$ ;
      while  $k > k_{max}$  do
        begin  $k := k + 1$ ;
          if random  $< P_{cross}$  then
            begin vyber náhodne  $x \in P$ ;
               $x' := 0_{pertur}(x)$ ;
               $Pr := \min(1, \exp(-(f(x') - f(x))/T))$ ;
              if random  $< Pr$  then  $P := (P \setminus \{x\}) \cup \{x'\}$ ;
            end else
              begin vyber náhodne  $x, x' \in P$ ;
                 $(z, z') := 0_{cross}(x, x')$ ;
                 $Pr_1 := \min(1, \exp(-(f(z) - f(x))/T))$ ;
                 $Pr_2 := \min(1, \exp(-(f(z') - f(x'))/T))$ ;
                if random  $< Pr_1$  then  $P := (P \setminus \{x\}) \cup \{z\}$ ;
                if random  $< Pr_2$  then  $P := (P \setminus \{x'\}) \cup \{z'\}$ ;
              end;
            end;
           $P_{out} := P$ ;
        end;
      end;

```

Algoritmus 6.5. Implementácia paralelného Metropolisovho algoritmu. Vstupné parametre sú P_{ini} (počiatočná množina stavov), k_{max} , T , výstupný parameter je P_{out} (konečná množina stavov). Vonkajší cyklus **while** sa opakuje k_{max} -krát. Pravdepodobnosť P_{cross} určuje náhodný výber jednoduchej poruchy vybraného stavu x , alebo či sa vykoná križenie nad dvojicou náhodne vybraných stavov x a x' . Dvojica nových stavov, ktoré sú produkované križením, je akceptovaná do množiny stavov pomocou Metropolisovho kritéria aplikovaného nezávisle nad dvoma dvojicami stavov (x, z) a (x', z') .

Proces interakcie dvoch náhodne vybraných stavov $x, x' \in P$ je v rámci Metropolisovho algoritmu určený pravdepodobnosťou $0 < P_{\text{cross}} \ll 1$. Táto pravdepodobnosť určuje proces výmeny, t.j. namiesto mutácie stavu $x \in P$ sa uskutoční interakcia dvoch stavov $x, x' \in P$ pomocou operátora kríženia (6.27). Potom sa otázka, či sú nové stavy z a z' akceptované, rieši pomocou Metropolisovho kritéria (6.2) aplikovaného zvlášť pre páry (x, z) a (x', z') . Napríklad, ak Metropolisovo kritérium akceptuje nový stav z , potom sa množina P obnoví tak, že starý stav x sa nahradí novým stavom z , formálne $P := (P \setminus \{x\}) \cup \{z\}$. Výsledné riešenie sa určí ako najlepšie riešenie z množiny P

$$x_{\text{opt}} = \arg \min_{x \in P} f(x) \quad (6.29)$$

kde P je množina riešení rezultujúca z dokončeného simulovaného žihania. Procedúry Metropolis_algorithm a Simulated_annealing sa môžu jednoducho modifikovať pre potreby paralelného simulovaného žihania (pozri algoritmy 6.5 a 6.6).

```

procedure Parallel_Simulated_annealing(input:  $T_{\text{max}}, T_{\text{min}}, k_{\text{max}}, \alpha$ ;
                                         output:  $x_{\text{opt}}$ );
begin  $P :=$  náhodne generovaný súbor stavov  $x, x', \dots$ ;
       $T := T_{\text{max}}$ ;
      while  $T > T_{\text{min}}$  do
        begin  $P_{\text{ini}} := P$ ;
              Parallel_Metropolis_algorithm( $P_{\text{ini}}, k_{\text{max}}, T, P_{\text{out}}$ );
               $P := P_{\text{out}}$ ;
               $T := \alpha * T$ ;
        end;
       $x_{\text{opt}} := \arg \min \{f(x), x \in P\}$ ;
end;

```

Algoritmus 6.6. Implementácia paralelného simulovaného žihania. Vstupné parametre sú $T_{\text{max}}, T_{\text{min}}, k_{\text{max}}, \alpha$, výstupný parameter je x_{opt} . Algoritmus sa inicializuje náhodnou generáciou množiny stavov z P . Po skončení algoritmu je výstupom najlepší stav z P označený x_{opt} .

Úloha 6.2. Formuláciou simulovaného žihania sme zavŕšili celú plejádu evolučných optimalizačných algoritmov: (1) horolezecký algoritmus (rôzne verzie, s učením, metóda zakázaného prehľadávania, evolučné programovanie), (2) genetické algoritmy a (3) simulované žihanie. Zvoľte si určité modelové úlohy a na týchto testujte efektívnosť vyššie uvedených metód a porovnajte ich.

Literatúra

- [1] P.J.M. van Laarhoven and E.H.L. Aarts: *Simulated Annealing. Theory and Applications*. Reidel, Dordrecht 1987.
- [2] R.H.J.M. Otten and L.P.P.P. van Ginneken: *Annealing Algorithm*. Kluwer, Boston 1989.
- [3] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi: Optimization by Simulated Annealing. *Science* **220** (1983) 671-680.
- [4] J. Černý: Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. *J. Opt. Theory Appl.* **45** (1985) 41-51.

- [5] N. Metropolis, A.W. Rosenbluth, N.N. Rosenbluth, A.H. Teller, and E. Teller: Equation of State Calculations for Fast Computing Machines. *J. Chem. Phys.* **21** (1953) 1087-1092.
- [6] A L. Ingber: Very fast simulated re-annealing. *J. Mathematical Computer Modelling*, **12** (8) (1989) 967-973, spolu so zdrojovým kódom dostupné na internetovskej adrese <http://www.ingber.com/>
- [7] M. Toda, R. Kubo, and N. Saito: *Statistical Physics*. Springer Verlag, Berlin 1983.
- [8] V. Kvasnička and J. Pospíchal: Simulated Annealing. *MATCH (Communications in Mathematical and Computer Chemistry)* **34** (1996) 7-49.
- [9] S.W. Mahfoud and D.E. Goldberg: A genetic algorithm for parallel simulated annealing. In R. Manner and B. Manderick (eds.): *Parallel Problem Solving from Nature 2*, pp. 301-310, North-Holland, Amsterdam 1992.
- [10] J. Pospíchal and V. Kvasnička: Fast Evaluation of Chemical Distance by Simulated Annealing Algorithm. *J. Chem. Inf. Comp. Sci.* **33** (1993) 879-885.

7. Kombinatorické optimalizačné problémy

7.1 Formulácia základných kombinatorických optimalizačných problémov

Jeden z prvých veľkých úspechov evolučných algoritmov bolo ich použitie na riešenie ťažkých kombinatorických problémov [1], ktorých čas CPU rastie buď faktoriálovo alebo exponenciálne s rastom dimenzie problému (hovoríme, že sú NP = "Nondeterministic Polynomial" ťažké [2]). Budeme študovať dva typy kombinatorických problémov:

Typ 1. Funkcia f je definovaná nad symetrickou grupou S_N zloženou zo všetkých permutácií N objektov

$$f : S_N \rightarrow R \quad (7.1)$$

kde permutácie z S_N sú definované ako N -tice celých rôznych čísel

$$P = (p_1, p_2, \dots, p_N) \quad (7.2)$$

Optimalizačný problém, ktorý je podobný problému (2.2) alebo (2.5), má tvar

$$P_{opt} = \arg \min_{P \in S_N} f(P) \quad (7.3)$$

Riešenie tohto optimalizačného problému je obvykle veľmi ťažké v dôsledku skutočnosti, že symetrická grupa S_N obsahuje $N!$ permutácií a je potrebné preskúmať hodnoty pre všetky možné argumenty. Z týchto dôvodov môže nastať situácia, že pre veľké N čas CPU rastie zhruba ako $N!$.

Typ 2. Nech $R_{set} = \{1, 2, \dots, r\}$ je množina obsahujúca prvých r prirodzených čísel, jej priamy súčin R_{set}^N obsahuje N -tice

$$\pi = (\pi_1, \pi_2, \dots, \pi_N) \in R_{set}^N = R_{set} \times R_{set} \times \dots \times R_{set} \quad (7.4)$$

Jednotlivé zložky tohto výrazu sú celé čísla z uzavretého intervalu $[1, r]$. Funkcia

$$f : R_{set}^N \rightarrow R \quad (7.5)$$

zobrazuje N -násobný karteziánsky produkt R_{set}^N na reálne čísla, analógia optimalizačného problému (2.5) má tvar

$$\pi_{opt} = \arg \min_{\pi \in R_{set}^N} f(\pi) \quad (7.6)$$

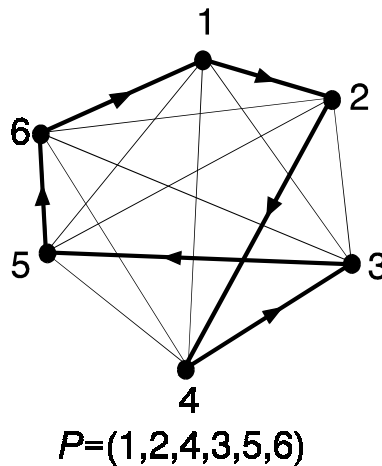
V dôsledku toho, že kardinalita množiny R_{set}^N je r^N , optimalizačný problém môže patriť medzi ťažké problémy s exponenciálnym rastom času CPU pre veľké N . Poznamenajme, že pre $r=2$ sa tento optimalizačný problém redukuje na obyčajný binárny problém (2.5).

Typ 1 kombinatorického problému budeme ilustrovať známou úlohou obchodného cestujúceho (TSP, Traveling Salesman Problem) [2]. Jeho grafovo-teoretická formulácia je nasledovná: nech G je úplný graf obsahujúci N vrcholov a $N(N-1)/2$ hrán (spojov). Každá hrana $[i, j]$ spájajúca i -ty vrchol s j -tym vrcholom (mestom) sa ohodnotí kladným číslom $d(i, j)$, ktoré sa interpretuje ako vzdialenosť medzi danými vrcholmi - mestami. V prípade, že by priamy spoj v skutočnosti neexistoval, vzdialenosť $d(i, j)$ dvoch vrcholov sa môže definovať ako dĺžka najkratšej cesty medzi nimi prechádzajúcej aj cez ostatné vrcholy. Služobná cesta

obchodného cestujúceho (hamiltonovský cyklus) navštívi každé mesto len raz, pričom sa vracia do východiskového mesta. Táto cesta je jednoducho určená ako permutácia N objektov - miest (7.2), odpovedá ceste, ktorá sa začala v meste p_1 , potom pokračuje postupne cez mestá $p_2, p_3, \dots, p_N$, a na záver sa vráti do východiskového mesta p_1 (pozri obr. 7.1). Každé ceste je priradená jej dĺžka

$$f(P) = d(p_1, p_N) + \sum_{i=2}^N d(p_{i-1}, p_i) \quad (7.7)$$

Hlavným cieľom úlohy TSP je nájsť takú cestu - permutáciu P_{opt} , ktorá poskytuje minimálnu dĺžku cesty $f_{opt} = f(P_{opt})$.



Obrázok 7.1. Cyklická cesta (hamiltonovský cyklus) na úplnom grafe, ktorý obsahuje 6 vrcholov. Cestu reprezentuje permutácia $P = (1, 2, 4, 3, 5, 6)$.

```

procedure Permutation_Mutation(input: P; output P');
begin for i:=1 to N do  $p'_i := p_i$ ;
    for i:=1 to N do
        if random <  $P_{mut}$  then
            begin j:=1+random(N);
                aux:= $p'_i$ ;  $p'_i := p'_j$ ;  $p'_j := aux$ 
            end;
    end;
end;

```

Algoritmus 7.1. Procedúra pre realizáciu stochastickej transformácie permutácie P na permutáciu P' tak, aby platili podmienky (7.8a-b). Premenná `random` je náhodné číslo z intervalu $[0,1)$ generované s rovnomernou distribúciou pravdepodobnosti. Podobne je `random(N)` náhodné celé číslo z intervalu $[0, N-1]$ s rovnomernou distribúciou.

Cesta P sa môže zmeniť na novú cestu P' pomocou stochastického operátora mutácie O_{mut} špecifikovaného pravdepodobnosťou P_{mut} . Budeme požadovať, že ak je cesta P permutácia, potom aj nová cesta P' je permutácia, a navyše, ak pravdepodobnosť $P_{mut} \rightarrow 0$, potom sú permutácie P a P' identické

$$P \in \mathcal{S}_N \Rightarrow P' = O_{mut}(P) \in \mathcal{S}_N \quad (7.8a)$$

$$\lim_{P_{mut} \rightarrow 0} O_{mut}(P) = P \quad (7.8b)$$

Ako algoritmicke zrealizovať tieto dve podmienky? Jednoduchá metóda algoritmickej stochastickej transformácie permutácie na permutáciu je vyjadrená pomocou algoritmu 7.1, ktorý "mutáciu" cesty realizuje ako jednoduchú transpozíciu permutácie, ktorá danú cestu reprezentuje.

Na to, aby sa tento typ kombinatorických úloh mohol študovať genetickým algoritmom, musíme zaviesť ešte operáciu kríženia medzi dvoma permutáciami [3]

$$(P', Q') = O_{cross}(P, Q) \quad (7.9)$$

ktoré zachováva ich charakter permutácie, t.j. P' a Q' sú taktiež permutácie. Študujme dve permutácie P a Q n objektov, vyberme bod kríženia a tak, že $1 < a < n$

$$P = (p_1, \dots, p_a, p_{a+1}, \dots, p_n) \quad (7.10a)$$

$$Q = (q_1, \dots, q_a, q_{a+1}, \dots, q_n) \quad (7.10b)$$

Keď vymeníme segmenty, ktoré nasledujú za bodom kríženia, dostaneme dva nové objekty

$$\hat{P} = (p_1, \dots, p_a, q_{a+1}, \dots, q_n) \quad (7.11a)$$

$$\hat{Q} = (q_1, \dots, q_a, p_{a+1}, \dots, p_n) \quad (7.11b)$$

Žiaľ, takto vytvorené objekty nemusia byť už permutácie. Môže nastať situácia, že nejaké celé číslo sa v objektoch $\hat{P}$ alebo $\hat{Q}$ vyskytuje dvakrát. Z tohto dôvodu je potrebné aplikovať "opravný proces" [3] (nazývaný čiastočné priradenie - partial matching), ktorý z objektov $\hat{P}$ a $\hat{Q}$ vytvorí normálne permutácie (poznamenajme, že v literatúre [3] je opísaných mnoho rôznych druhov kríženia dvoch permutácií a spôsobov ich opráv). Definujme zobrazenie f_{PQ} tak, že zložky permutácie $\hat{P}$, ktoré sa vymenili, sú zobrazené na komponenty permutácie $\hat{Q}$, ktoré sa tiež zúčastnili výmeny

$$f_{PQ} : p_i \rightarrow q_i \text{ (pre } i > a) \quad (7.12)$$

Druhé zobrazenie f_{QP} je určené ako inverzné zobrazenie k f_{PQ} , $f_{QP} = f_{PQ}^{-1}$. Pomocou týchto dvoch zobrazení budeme opravovať "pôvodné" časti v $\hat{P}$ a $\hat{Q}$, t.j. tie, ktoré ležia pred bodom kríženia. Ak zložka p_i (pre $1 \leq i \leq a$) nepatrí do definičného oboru zobrazenia f_{QP} (to tiež znamená, že p_i sa vyskytuje v $\hat{P}$ práve raz), potom p_i zostáva nezmenené. V opačnom prípade, ak sa p_i nachádza v definičnom obore zobrazenia f_{QP} , potom sa p_i zamení za prvú hodnotu iteračnej schémy $x_{k+1} = f_{QP}(x_k)$, ktorá je mimo definičnej oblasti funkcie f_{QP} (iteračné riešenie sa inicializuje $x_0 = p_i$). Podobný postup sa aplikuje aj pre opravu pôvodného segmentu v $\hat{Q}$, avšak teraz sa použije zobrazenie f_{PQ} .

Pre ilustráciu študujme kríženie medzi dvoma permutáciami

$$P = (3, 4, 5, 1, 2, 6, 10, 8, 9, 7) \text{ a } Q = (1, 2, 6, 10, 8, 3, 4, 5, 7, 9)$$

a predpokladajme, že bod kríženia je $a=4$. Ak vymeníme medzi dvoma permutáciami segmenty za bodom kríženia, dostaneme

$$\hat{P} = (3, 4, 5, 1, 8, 3, 4, 5, 7, 9) \text{ a } \hat{Q} = (1, 2, 6, 10, 2, 6, 10, 8, 9, 7)$$

Vidíme, že objekty $\hat{P}$ a $\hat{Q}$ nie sú permutácie, pretože obsahujú niektoré indexy dvakrát. Na opravu týchto objektov na permutácie zostrojíme zobrazenia f_{QP} a f_{PQ}

$$f_{PQ} = \begin{pmatrix} 2 & 6 & 10 & 8 & 9 & 7 \\ 8 & 3 & 4 & 5 & 7 & 9 \end{pmatrix} \text{ a } f_{QP} = \begin{pmatrix} 8 & 3 & 4 & 5 & 7 & 9 \\ 2 & 6 & 10 & 8 & 9 & 7 \end{pmatrix}$$

Objekt $\hat{P}$ opravíme pomocou zobrazenia f_{QP} a objekt $\hat{Q}$ pomocou zobrazenia f_{PQ} . Prvé tri indexy v P' zameníme za $3 \rightarrow 6$, $4 \rightarrow 10$ a $5 \rightarrow 8 \rightarrow 2$. Podobne zameníme prvé indexy 2, 6 a 10 v

Q' za $2 \rightarrow 8 \rightarrow 5$, $6 \rightarrow 3$ a $10 \rightarrow 4$, takto dostaneme opravené objekty, ktoré už sú permutácie a považujú sa za výsledok výmeny medzi permutáciami P a Q

$$P' = (6, 10, 2, 1, 8, 3, 4, 5, 7, 9) \text{ a } Q' = (1, 5, 3, 4, 2, 6, 10, 8, 9, 7)$$

Kombinatorické úlohy typu 2 budeme ilustrovať úlohou známou pod názvom problém rozkladu čísla (NPP, Number Partitioning Problem [4]). Nech $Q = \{q_1, q_2, \dots, q_N\}$ je množina obsahujúca N kladných reálnych čísel a nech π je zobrazenie Q na množinu $R_{set} = \{1, 2, \dots, r\}$

$$\pi : Q \rightarrow R_{set} \quad (7.13)$$

Toto zobrazenie π môže byť jednoznačne vyjadrené ako N -tica $\pi = (\pi_1, \pi_2, \dots, \pi_N) \in R_{set}^N$. Jej jednotlivé zložky sa interpretujú tak, že celé číslo $\pi_i \in R_{set}$ je priradené objektu $q_i \in Q$. Vzhľadom na toto zobrazenie sa množina Q môže rozložiť na disjunktné podmnožiny

$$Q = Q_1 \cup Q_2 \cup \dots \cup Q_r \quad (7.14a)$$

kde

$$Q_i = \{q \in Q; \pi(q) = i\} \quad (7.14b)$$

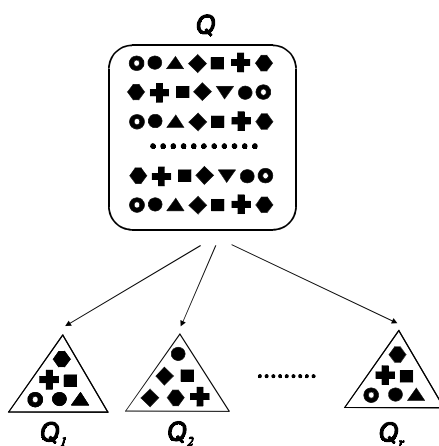
Podmnožina Q_i obsahuje všetky elementy - čísla množiny Q , ktoré sú zobrazené funkciou π na celé číslo i . Definujme účelovú funkciu

$$f(\pi) = \max_{q \in Q_i} q - \min_{q \in Q_i} q \quad (7.15)$$

Vyjadruje rozdiel medzi maximálnym a minimálnym súčtom čísel z podmnožín $Q_1, Q_2, \dots, Q_N$. Cieľom úlohy NPP je nájsť také zobrazenie π , ktoré minimalizuje účelovú funkciu f

$$\pi_{opt} = \arg \min_{\pi \in R_{set}^N} f(\pi) \quad (7.16)$$

Ako názorne interpretovať túto úlohu? Predstavme si, že máme kopu obsahujúcu N vecí, pričom každej veci je priradená cena $q_1, q_2, \dots, q_N$. Cieľom je rozdeliť túto kopu vecí na r kôp tak, aby ich ceny (súčty cien jednotlivých vecí patriacich do danej kopy) vykazovali medzi sebou minimálne rozdiely (pozri obr. 7.2).



Obrázok 7.2. Cieľom úlohy NPP je rozdeliť kopu (množinu) Q na r kôp $Q_1, Q_2, \dots, Q_r$ tak, aby rozdiely medzi ich cenami boli minimálne. Predstavme si partiu r zlodejov, ktorá nakradla kopu Q vecí, pričom každá z týchto vecí má určitú cenu. Po čase si zloději chcú nakradnuté veci rozdeliť tak, aby každý dostal veci s približne rovnakou hodnotou [4]. Táto úloha je totožná s úlohou NPP.

K zobrazeniu π môžeme pomocou operátora mutácie, $\pi' = O_{mut}(\pi)$ priradiť iné zobrazenie π' podobným spôsobom ako pri binárnej reprezentácii (pozri algoritmus 7.2), pričom pre $P_{mut} \rightarrow 0$ sa vytvorí zobrazenie π' totožné s pôvodným zobrazením π . Operácia križenia je pre tento typ chromozómov veľmi jednoduchá a môže sa priamo stotožniť s podobnou operáciou pre binárne vektory (pozri kapitolu 4.3).

```

procedure Mapping_Mutation(input: $\pi$ ; output: $\pi'$ );
begin for i:=1 to N do
    if random< $P_{mut}$  then
         $\pi'_i := 1 + \text{random}(r)$  else  $\pi'_i := \pi_i$ ;
end;

```

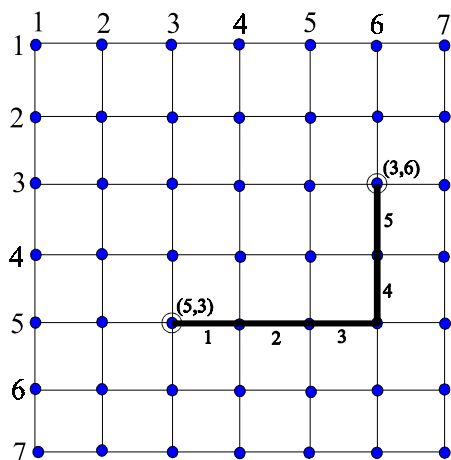
Algoritmus 7.2. Pseudopascalovská procedúra realizujúca mutáciu zobrazenia π na nové zobrazenie π' . Premenné `random` a `random(r)` sú definované podobne ako v algoritme 7.1.

Úloha 7.1. V tejto kapitole sa opisuje prístup k riešeniu kombinatorických úloh pomocou permutácií a n -tíc prirodzených čísel, ktorý sa líši od štandardnej binárnej reprezentácie riešení. Ukážeme, že aj v tomto prípade je binárna reprezentácia stále použiteľná.

Permutácia N objektov (kombinatorické úlohy typu 1) sa môže zadať ako binárny vektor dĺžky kN , kde k je také celé číslo, ktoré zabezpečuje, aby binárny vektor dĺžky k bol schopný reprezentovať číslo N . Takýto binárny vektor sa ľahko pretransformuje postupom z kapitoly 2.2 na vektor N celých čísel z intervalu $[0, 2^k - 1]$. Potom je permutácia P definovaná tak, že celočíselné zložky sú usporiadané do nerastúcej alebo neklesajúcej postupnosti. Nech $N=5$, $k=3$, binárny vektor (101|011|001|111|101), jeho celočíselná verzia má tvar (5,4,1,7,5) (pozri tab. 2.1). Nech je permutácia P určená tak, že preusporiada túto N -ticu do rastúcej postupnosti, potom $P=(3,2,1,5,4)$. Nevýhodou tohto prístupu ku kódovaniu permutácií je, že rovnaká permutácia je určená rôznymi binárnymi reťazcami, čo môže spôsobovať pomerne malú efektívnosť evolučných algoritmov aplikovaných na riešenie kombinatorických algoritmov. Napíšte program, ktorý binárny vektor pretransformuje na permutáciu N objektov. Kritickým miestom programu bude podprogram na usporiadanie N -tice čísel klesajúcej alebo rastúcej postupnosti, preto použite nejaký efektívny algoritmus na usporiadavanie (napr. quicksort).

Binárna reprezentácia N -tice celých čísel z množiny $R_{set}^N = \{1, 2, \dots, r\}^N$ (kombinatorické úlohy typu 2) sa môže priamo reprezentovať binárnym vektorom dĺžky kN , kde k je určené podmienkou $r \leq 2^k - 1$. Nech binárny podreťazec dĺžky k reprezentuje reálne číslo z intervalu $[1, r]$, k reálnemu číslu x z tohto intervalu priradíme celé číslo z intervalu $[1, r]$ operáciou $\lfloor x \rfloor$, ktorá sa nazýva "zaokrúhlenie" (napr. $\lfloor 1.1 \rfloor = 1$ a $\lfloor 1.8 \rfloor = 1$). Na ilustráciu uvažujme $N=5$, $k=3$ a $r=4$ a rovnaký binárny vektor (101|011|001|111|101) ako v prechádzajúcej úlohe. Vektor s reálnymi zložkami vytvorený pomocou transformácie (2.9) má tvar (3.143, 2.286, 1.429, 4.000, 3.143). Aplikovaním operácie zaokrúhlenia dostaneme vektor celých čísel (3,2,1,4,3). Napíšte program, ktorý binárny vektor pretransformuje na N -ticu celých čísel z intervalu $[1, r]$.

Úloha 7.2. Napíšte programy pre vybrané evolučné algoritmy opísané v predchádzajúcich kapitolách, ktoré riešia úlohu obchodného cestujúceho (Traveling Salesman Problem). Použite oba prístupy na kódovanie cesty, teda priamo pomocou permutácie aj pomocou binárneho reťazca (pozri úlohu 7.1). Porovnajte efektívnosť týchto dvoch rôznych prístupov.

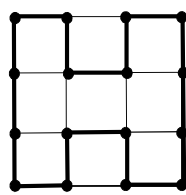


Verifikácia výsledkov pre problém obchodného cestujúceho nie je jednoduchou záležitosťou. Preto sa navrhujú také príklady, kde sme schopní presne odhadnúť dĺžku optimálnej cesty. Často sa študuje ortogonálna mriežka bodov typu $N \times N$, pričom vzdialenosť medzi jednotlivými bodmi sa počíta pomocou vzdialenosti typu Manhattan (L_1 metriky, teda súčtu absolútnych hodnôt rozdielov jednotlivých súradníc dvoch bodov). Na obrázku je znázornená mriežka typu 7×7 , ktorá obsahuje 49 bodov. Vzdialenosťi typu Manhattan (D_{ij}) medzi dvoma bodmi s riadkovými-stĺpcovými indexami (5,3) a (3,6) je $|5-3|+|3-6|=5$. Dva susedné body ležiace na rovnakom riadku alebo stĺpci majú jednotkovú vzdialenosť.

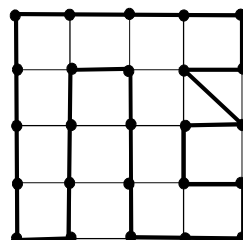
Pre tento špeciálny prípad je optimálna vzdialenosť

$$f_{opt} = \begin{cases} N^2 & (\text{pre párne } N) \\ N^2 + 1 & (\text{pre nepárne } N) \end{cases}$$

Tento vzorec sa môže jednoducho verifikovať pomocou matematickej indukcie. Dva jednoduché ilustratívne príklady sú znázornené na nasledujúcich obrázkoch.



$N=4, f_{opt}=16$



$N=5, f_{opt}=26$

Optimalizujte parametre metód pre rôzne $N=4,5,\dots,10$ tak, aby vypočítaná minimálna dĺžka cesty na ortogonálnej mriežke bodov bola rovná teoretickej minimálnej vzdialenosti sa skoro 100% pravdepodobnosťou.

Úloha 7.3. Napíšte programy pre vybrané evolučné algoritmy, ktoré riešia úlohu rozkladu čísla (Number Partitioning Problem). Použite oba prístupy na kódovanie zobrazenia π priamo pomocou N -tice celých čísel, ako aj pomocou binárneho reťazca (pozri úlohu 7.1). Porovnajete efektívnosť týchto dvoch rôznych prístupov.

Podobne ako v predchádzajúcej úlohe použijeme jednoduché zadanie umožňujúce priamu verifikáciu správnosti (optimálnosti) získaných riešení. Nech je úloha určená parametrami $N=100$ a $r=10$. To znamená, že počet všetkých možných zobrazení π je 10^{100} . Na to, aby sa nám podarilo určiť optimálne riešenie, predpokladajme, že Q obsahuje 10-krát číslo 1 až 10. Optimálny rozklad Q na 10 podmnožín $Q_1, Q_2, \dots, Q_{10}$ je taký, že každá z týchto podmnožín obsahuje čísla 1 až 10 práve 1-krát. Potom cena každej podmnožiny je

$$\sum_{q \in Q_i} q = 1 + 2 + \dots + 10 = 55$$

pre $i=1,2,\dots,10$, preto $f(\pi_{opt})=0$. Optimalizujte parametre metód tak, aby získaná hodnota účelovej funkcie bola nulová so skoro 100% pravdepodobnosťou.

Úloha 7.4. Napíšte program pre metódu zakázaného hľadania (pozri kapitolu 3.3) aplikovanú na riešenie problému obchodného cestujúceho (TSP), pričom cesta nech je vyjadrená pomocou permutácie. Transformačné operátory $t \in S$ sa realizujú ako permutácie transpozícií dvoch objektov. Operátor t_{ij} pôsobí na permutáciu P ako transpozícia objektov i a j

$$t_{ij}P = t_{ij}(p_1, \dots, p_i, \dots, p_j, \dots, p_N) = P' = (p_1, \dots, p_j, \dots, p_i, \dots, p_N)$$

pre $i < j$. Množina S potom obsahuje $N(N-1)/2$ transformácií. Inverzná transformácia t_{ij}^{-1} k transformácii t_{ij} je totožná s touto transformáciou, $t_{ij}^{-1} = t_{ij}$. Okolie $U(P)$ priradené riešeniu - permutácii P obsahuje $N(N-1)/2$ elementov

$$U(P) = \{P' = tP; \forall t \in S\}$$

Ako testovaciu úlohu použite ortogonálnu mriežku bodov - miest, kde sa vzdialenosť počíta pomocou L_1 metriky (pozri úlohu 7.2). Pokúste sa o optimalizáciu veľkosti "s" zakázaného zoznamu T (tabu listu) a maximálneho počtu iterácií $time_{max}$ pre rôzne hodnoty N dimenzie ortogonálnej mriežky bodov, $N=4,5,\dots,10$, tak, aby ste s 95% pravdepodobnosťou dostali korektný výsledok (špecifikovaný v úlohe 7.2). Podobné úvahy vykonajte aj pre suboptimálne výsledky, ktoré sa líšia len o niekoľko málo jednotiek od optimálneho výsledku. Zistíte, že metóda zakázaného hľadania poskytuje veľmi rýchlo suboptimálne výsledky, ktoré sú veľmi blízke optimálnym.

Úloha 7.5. Metóda zakázaného hľadania (a nielen táto metóda) sa v literatúre často ilustruje známou jednoduchou kombinatorickou úlohou nájsť na šachovnici s rozmermi $N \times N$ postavenie N dām tak, aby v každom riadku, stĺpci a diagonále bola umiestnená práve jedna dáma (t.j. neexistuje kolízne postavenie medzi dāmami). Zložitosť riešenia tohto problému veľmi závisí od zvolenej reprezentácie postavenia dām na šachovnici. Jednoduchá reprezentácia postavenia dām je pomocou permutácie $P=(p_1, p_2, \dots, p_N)$. Potom dámy na šachovnici sú umiestnené takto: v 1. stĺpci na p_1 -tom riadku, v 2. stĺpci na p_2 -tom riadku, atď. Táto jednoduchá reprezentácia rozmiestnenia dām na šachovnici pomocou permutácií má výhodu v tom, že v každom stĺpci a riadku šachovnice sa nachádza práve jedna dáma. Každéj permutácii N objektov priradíme celé nezáporné číslo $col(P)$, ktoré odpovedá počtu diagonálnych kolízií. Riešenie problému je potom taká permutácia P , ktorá má nulový počet kolízií, $col(P)=0$. Funkcia $col(P)$ je určená vzťahom

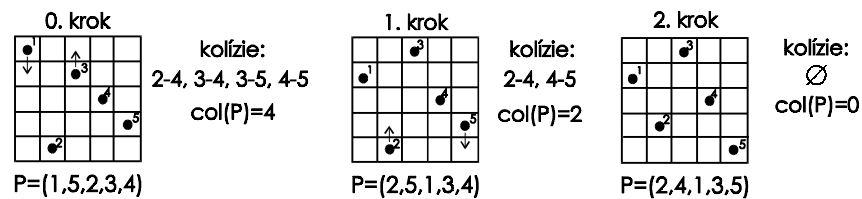
$$col(P) = \sum_{\substack{i,j=1 \\ (i < j)}}^N [\delta(p_i + j - i, p_j) + \delta(p_i - j + i, p_j)]$$

kde $\delta(i,j)$ je Kroneckerovo delta, $\delta(i,j)=1$, pre $i=j$, $\delta(i,j)=0$, pre $i \neq j$. Výpočet zmeny počtu kolízií pri prechode od permutácie P k novej permutácii $P'=t_{ij}P$ v rámci metódy zakázaného hľadania určuje vzorec

$$col(P') - col(P) = \sum_{k=1}^N (k \neq i, j) [\delta(p_k + i - k, p_j) + \delta(p_k - i + k, p_j) + \delta(p_k + j - k, p_i) + \delta(p_k - j + k, p_i) - \delta(p_k + i - k, p_i) - \delta(p_k - i + k, p_i) - \delta(p_k + j - k, p_j) - \delta(p_k - j + k, p_j)]$$

Numerická náročnosť výpočtu tohto vzorca je podstatne menšia ako predchádzajúceho vzorca. Preto je výhodnejšie počítať počet kolízií $\text{col}(t_{ij}P)$ pomocou tohto vzorca za predpokladu, že je známy počet kolízií $\text{col}(P)$.

Program z predchádzajúcej úlohy 7.4 upravte tak, aby riešil úlohu N dám. Na ilustráciu tohto algoritmu uvádzame prvé tri iteračné kroky zakázaného hľadania pri riešení úlohy pre $N=5$. V nultom kroku sa náhodne vygenerovalo postavenie dám na šachovnici, ktoré má 4 kolízie. Nová permutácia P v 1. kroku vznikla z predchádzajúcej permutácie použitím transpozície t_{13} , počet kolízií sa znížil na 2. V 2. kroku dostaneme použitím transpozície t_{25} novú permutáciu, ktorá má už nulový počet kolízií, čiže je riešením problému.



Úloha 7.6. Napíšte program pre metódu zakázaného hľadania, ktoré rieši úlohu N dám na šachovnici $N \times N$, pričom permutácia P je reprezentovaná binárne (pozri úlohu 7.1). Porovnajte efektívnosť tohto programu s programom z predchádzajúcej úlohy 7.5.

Úloha 7.7. Riešte problém obchodného cestujúceho pomocou genetického algoritmu, pričom cesta obchodného cestujúceho je zadaná pomocou permutácie. Ako cvičné úlohy riešte konštrukciu optimálnej cesty na ortogonálnej mriežke bodov s L_1 metrikou (pozri úlohu 7.2). Pokúste sa navrhnúť novú operáciu kríženia medzi dvoma permutáciami, tak, aby výsledné objekty - potomkovia boli znovu permutáciami.

Literatúra

- [1] C.R. Reeves (ed.): *Modern Heuristic Techniques for Combinatorial Problems*. McGraw-Hill, London 1995.
- [2] L. Kučera: *Kombinatorické algoritmy*. SNTL, Praha 1983.
- [3] D.E. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading, MA 1989.
- [4] D. Whitley, V.S. Gordon, and A.P.W. Böhm: *Knapsack Problems*. In T. Bäck, D.B. Fogel, and Z. Michalewicz (eds.): *Handbook of Evolutionary Computation*. Oxford University Press and Institute of Physics Publishing, Oxford 1997.

8. Evolučné stratégie

8.1 Historický úvod

Evolučná stratégia patrí historicky medzi prvé úspešné stochastické algoritmy. Bola navrhnutá už počiatkom 60-tych rokov Rechenbergom a Schwefelom, aj keď knižne publikovaná bola až neskôr [1-3]. Podrobný popis možno nájsť v poslednej Schwefelovej knihe [4], ale dobrý prehľad poskytnú aj časopisecké články, konferencie a správy [5-10]. Evolučná stratégia vychádza zo všeobecných predstáv prirodzeného výberu, avšak omnoho vážnejších ako napríklad pri genetickom algoritme. Na rozdiel od väčšiny ostatných stochastických optimalizačných metód nie je evolučná stratégia založená na binárnej reprezentácii premenných, ale manipuluje priamo s "reálnou" reprezentáciou premenných (ako všade, aj tu existujú výnimky, diskretnú optimalizáciu použil napr. Herdy [11], ktorý smerodajnú odchýlku nahradzoval rôznymi parametrami určujúcimi "veľkosť" mutácie - pri probléme obchodného cestujúceho je to dĺžka podreťazca permutácie určujúcej poradie miest, ktorý bude invertovaný).

Budeme študovať minimalizáciu funkcie $f(\mathbf{x})$, kde $\mathbf{x} \in \mathbf{R}^n$, teda nezávislé premenné funkcie predstavuje n reálnych čísel zoradených do vektora $\mathbf{x}$. Prípustné hodnoty jednotlivých premenných vektora $\mathbf{x}$ môžu byť napr. ohraničené súčasne zdola aj zhora, $x_i \in [a, b]$, potom $\mathbf{x} \in [a, b]^n \in \mathbf{R}^n$.

Ingo Rechenberg, Hans-Paul Schwefel (teraz už obidvaja profesori) a Peter Bienert sa začali ako študenti na Technickej univerzite v Berlíne v r. 1963 zaoberať experimentmi v aerodynamickom tuneli, kde optimalizovali tvar telies tak, aby tieto mali čo najmenší odpor vo vzdušnom prúde. Pretože intuitívny prístup ani gradientové metódy neboli veľmi úspešné, začali sa zaoberať myšlienkou náhodných zmien parametrov definujúcich tvar (podobne ako je to pri mutáciách) v priebehu stochastického optimalizačného procesu.

Evolučné stratégie patria v súčasnosti medzi dobre rozvinuté stochastické optimalizačné metódy, ale stále ostávajú spojené predovšetkým s technickými aplikáciami v inžinierskych oblastiach a používajú sa najviac v Nemecku. Prvé evolučné stratégie, ktoré teraz opíšeme, používali jediný rekombinačný operátor – mutáciu a iba jedno riešenie v populácii s maximálne dvoma "jedincami" uchovávanými súčasne - rodič a navrhovaný potomok. Taký prístup sa nazýva dvojčlenná evolučná stratégia alebo (1+1)-ES.

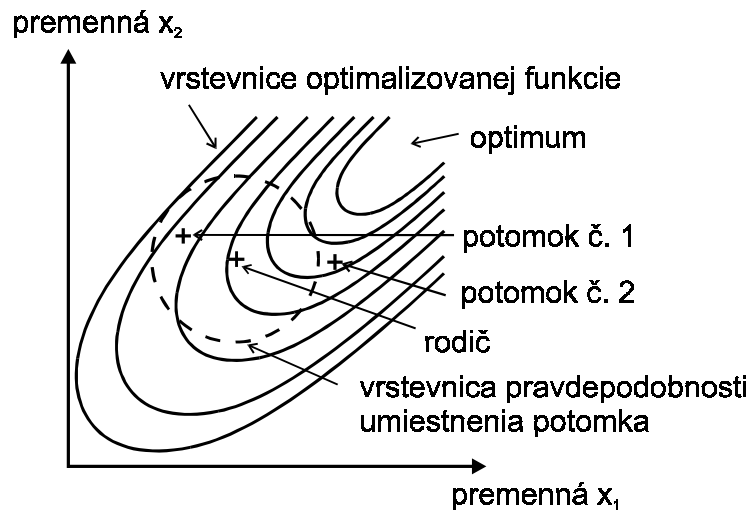
Základom evolučnej stratégie je nasledujúci predpis, ktorý "mutuje" aktuálne riešenie $\mathbf{x}$ na nové riešenie $\mathbf{x}'$ (viď obr. 8.1 a 8.2),

$$\mathbf{x}' = \mathbf{x} + \mathbf{N}(0, \sigma), \quad (8.1)$$

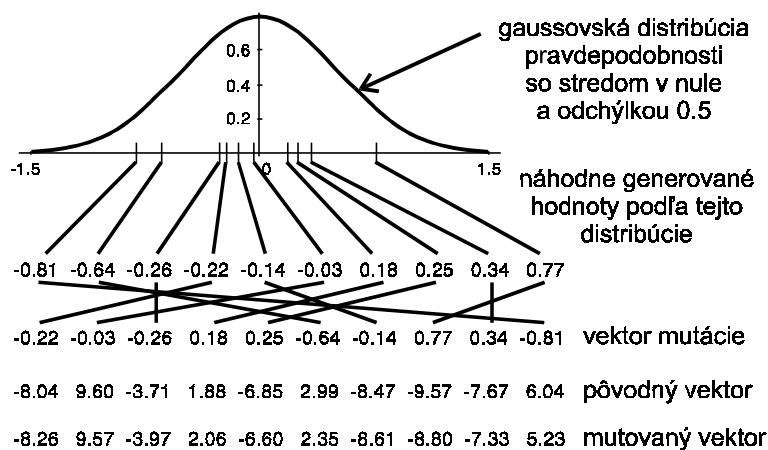
kde $\mathbf{N}(0, \sigma)$ je vektor nezávislých náhodných čísel s nulovou strednou hodnotou a štandardnou odchýlkou σ . Jednotlivé čísla sú náhodne rozložené podľa gaussovskej distribúcie, menšie čísla sa objavujú pravdepodobnejšie ako väčšie čísla. Štandardná odchýlka je mierou rozptylu generovaných hodnôt, mierou variácie náhodnej premennej. Určuje šírku dvojrozmerného "kopčeka pravdepodobnosti", ktorého základňu tvorí os možných hodnôt vygenerovaných čísel a ktorého výška v danom bode odpovedá pravdepodobnosti, s akou bude číslo vygenerované. Tento "kopček pravdepodobnosti" sa nazýva funkcia hustoty rozdelenia náhodnej premennej a pre normálnu gaussovskú distribúciu je opísaný ako

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-0.5(x-\mu)^2/\sigma^2} \quad (8.2)$$

kde μ je stredná hodnota premennej x a σ je štandardná odchýlka. Väčšia štandardná odchýlka zväčšuje (v priemere z viacerých pokusov) vzdialenosť medzi rodičom a vygenerovaným potomkom.



Obrázok 8.1. Znáznornenie optimalizácie funkcie $f(x_1, x_2)$ dvoch premenných. Rovnaké funkčné hodnoty sú na obrázku znázornené neprerušovanými vrstevnicami. Optimálna hodnota funkcie sa nachádza v pravom hornom rohu. Prerušovaná kružnica označuje vrstevnicu pravdepodobnosti výskytu potomka. Polohy bližšie k rodičovi majú podľa gaussovskej krivky normálneho rozdelenia väčšiu pravdepodobnosť umiestnenia potomka. Náhodne vygenerovaný potomok č. 1 zodpovedá horšiemu riešeniu ako rodič (čo vidno z vrstevníc funkcie). Preto namiesto neho vygenerujeme ďalšieho potomka č. 2. Ten zodpovedá lepšiemu riešeniu ako rodič, rodiča teda nahradíme potomkom č. 2 a ďalších potomkov už generujeme z potomka č. 2.



Obrázok 8.2. Príklad mutácie vektora s reálnymi premennými za použitia gaussovskej distribúcie pravdepodobnosti na určenie malej náhodnej zmeny (nie vždy je nutné mutovať všetky hodnoty vektora, stačí si náhodne vybrať, ktorý prvok sa bude mutovať).

Akceptovanie nového riešenia $\mathbf{x}'$ je striktné deterministické, riešenie $\mathbf{x}'$ je akceptované (úspešné), keď $f(\mathbf{x}') < f(\mathbf{x})$. Z rodiča generujeme jedného potomka, ktorý sa od rodiča líši len málo. Keď potomok zodpovedá horšiemu riešeniu ako rodič, vymažeme potomka a generujeme iného. To robíme tak dlho, až potomok zodpovedá lepšiemu riešeniu ako rodič a nahradí rodiča. Tento prístup sa podobá gradientovým metódam alebo horolezeckému algoritmu.

8.2 Stratégia (1+1) – jeden rodič a jeden potomok

Stratégia, v ktorej sa do ďalšej generácie vyberú lepší jedinci (tu iba jeden jedinec) tak z populácie rodičov, ako aj z populácie potomkov, sa volá stratégia PLUS. Pre jedného rodiča a jedného potomka sa tento prístup označuje stratégia (1+1). Rechenberg študoval vlastnosti tejto metódy pre dve základné funkcie. Pre prvú, "sférickú" funkciu $f(\mathbf{x}) = \sum_{i=1}^n x_i^2$ tvoria vrstevnice rovnakých funkčných hodnôt sústredné kružnice (sféry). Pre druhú funkciu $f(\mathbf{x}) = -\sum_{i=1}^n x_i$, nazývanú koridorový model, tvoria tieto vrstevnice rovnobežné čiary (pridané ohraničenia tvoria obdĺžnik, na ktorého dlhšiu stranu sú vrstevnice kolmé). Pretože ide o zjednodušenú stratégiu [7] aplikovanú na jednoduché funkcie, mohol Rechenberg [1] odvodiť predpis na zmenu veľkosti "mutácie" - teda vzdialenosti potomka od rodiča - a na "zmenu tejto zmeny". Štandardná odchýlka σ sa v priebehu evolučnej stratégie mení podľa pravidla 1/5 úspešnosti. Nech $\phi(k)$ je koeficient úspešnosti definovaný ako pomer počtu úspešných mutácií v priebehu posledných k iterácií k počtu k iterácií, z ktorých bola úspešnosť meraná, potom

$$\sigma' = \begin{cases} c_d \cdot \sigma & (\phi(k) < 1/5) \\ c_i \cdot \sigma & (\phi(k) > 1/5) \\ \sigma & (\phi(k) = 1/5) \end{cases} \quad (8.3)$$

kde $c_i > 1$ a $c_d < 1$ riadia zväčšovanie resp. zmenšovanie štandardnej odchýlky, v literatúre [1,3,4] sú tieto koeficienty špecifikované $c_d=0.82$ a $c_i=1/c_d=1.22$.

Intuitívnym zdôvodnením pravidla 1/5 úspechu je zvýšenie efektívnosti prehľadávania. Pri akceptácii viac ako jednej pätiny potomkov bude prehľadávanie pokračovať vo väčších krokoch, pri neúspešnosti by mali byť kroky kratšie. Toto pravidlo pôvodne odvodené pre uvedené dve jednoduché funkcie sa používa aj pre iné funkcie, pretože sa empiricky zistilo, že pre mnohé problémy zabezpečuje jeho používanie najrýchlejší postup k optimu. Metóda evolučnej stratégie v pseudopascalé je vyjadrená algoritmom 8.1.

Ako pre každý algoritmus, aj pre algoritmus 8.1 existujú drobné modifikácie. Napríklad zmena smerodajnej odchýlky sa môže robiť po každých n generáciách (prvá zmena sa vykoná až po 10 n generáciách). Vždy spočítame, koľko úspešných potomkov, ktorí nahradili rodiča, sa vygenerovalo za posledných 10 n generácií. Ak je to menej ako $2n$, vynásobíme veľkosť smerodajnej odchýlky tak, aby sa zmenšila (napr. číslom 0.82), keď bolo viac ako $2n$ pokusov úspešných, potom vynásobíme smerodajnú odchýlku tak, aby sa zväčšila (napr. číslom 1.22).

Je však vždy potrebné ohraničiť zmeny smerodajnej odchýlky tak, aby nedosiahla nulovú hodnotu pre danú presnosť.

Keď niektorá hodnota x_i nezapadá do svojho vymedzeného intervalu po pripočítaní vektoru mutácie k pôvodnému vektoru, t.j. $x_i \notin \langle a, b \rangle$, je potrebné použiť opravný proces, aby sa zaistilo, že opravená hodnota spadá do vymedzeného intervalu. Najjednoduchší takýto opravný proces je tzv. zrkadlový odraz, kedy umiestnime hodnotu x_i na opačnú stranu

hranice intervalu, v rovnakej vzdialenosti od hranice. Teda keď je napr. $b=3.5$ a $x_i=4$, potom nová hodnota $x_i=3$ je vzdialená o 0.5 od hodnoty b , ale na opačnú stranu. Pokiaľ sa hodnota x_i dostane týmto presunom za hranice intervalu a na opačnej strane, postupujeme rovnako. Algoritmicky tento prístup vyjadrený procedúrou *Perturbation* nahradí výraz $\mathbf{x}' := \mathbf{x} + \mathbf{N}(0, \sigma)$ v riadku 6 algoritmu 8.1. Symbol $\mathbf{N}(0, \sigma)$ tu vyjadruje jedno náhodné číslo vygenerované podľa gaussovskej distribúcie s nulovou strednou hodnotou a štandardnou odchýlkou σ .

Evolučná stratégia:

```

1   $\mathbf{x}$ :=náhodne generovaný vektor reálnych premenných;
2   $t:=0$ ;  $\sigma:=\sigma_{ini}$ ;  $\mathbf{x}^*:=\mathbf{x}$ ;
3  while  $t < t_{max}$  do
4  begin  $i:=0$ ;  $k:=0$ ;  $t:=t+1$ ;
5        while  $i < i_{max}$  do
6        begin  $i:=i+1$ ;  $\mathbf{x}' := \mathbf{x} + \mathbf{N}(0, \sigma)$ ;
7              if  $f(\mathbf{x}') < f(\mathbf{x})$  then
8              begin  $k:=k+1$ ;  $\mathbf{x}:=\mathbf{x}'$ ;
9                    if  $f(\mathbf{x}) < f(\mathbf{x}^*)$  then  $\mathbf{x}^*:=\mathbf{x}$ ;
10             end;
11        end;
12        if  $k/i_{max} < 0.2$  then  $\sigma:=c_d \sigma$ 
13        else if  $k/i_{max} > 0.2$  then  $\sigma:=c_i \sigma$ ;
14  end;
```

Algoritmus 8.1. Evolučná stratégia (1+1). Premenná t je počítadlo epoch evolučnej stratégie. Algoritmus obsahuje dva cykly *while*, vonkajší a vnútorný. Vo vnútornom cykle sa pre dané σ opakuje elementárny krok evolučnej stratégie i_{max} -krát, pričom premenná k zaznamenáva úspešnosť mutácií v tomto vnútornom cykle. Vonkajší cyklus, s počítadlom t , aplikuje pre rôzne hodnoty σ evolučnú stratégiu t_{max} -krát. Štandardná odchýlka σ sa inicializuje hodnotou σ_{ini} . Modifikácia riešenia $\mathbf{x}$ sa vykonáva pomocou generátora náhodných čísel s nulovou strednou hodnotou a so štandardnou odchýlkou σ . Voľba základných parametrov evolučnej stratégie (t_{max} , σ_{ini} a i_{max}) si vyžaduje určité experimentovanie, pomocou ktorého tieto konštanty nastavíme. Zvyčajne je σ_{ini} blízke jednotke a konštantá i_{max} sa rádovo rovná tisícom.

```

procedure Perturbation( $\mathbf{x}, \mathbf{x}', \sigma$ )
begin for  $i:=1$  to  $n$  do
  begin  $x'_i := x'_i + N(0, \sigma)$ ;
    while ( $x'_i > b$ ) or ( $x'_i < a$ ) do
      if  $x'_i > b$  then  $x'_i := 2b - x'_i$  else
      if  $x'_i < a$  then  $x'_i := 2a - x'_i$ ;
    end;
  end;
end;
```

Algoritmus 8.2. Pseudopascalovský kód procedúry pre mutáciu pôvodného vektora $\mathbf{x}$ na nový vektor $\mathbf{x}'$. V prípade, že niektorá zložka nového vektora je mimo intervalu $\langle a, b \rangle$, opravný proces "zrkadlenie" zabezpečí návrat zložky do intervalu.

Úloha 8.1. Naprogramujte evolučnú stratégiu (1+1) a aplikujte ju na optimalizáciu funkcie $y=x^2$, pričom $\sigma_{ini}=1$ a štartovací bod bude 100. Použite pravidlo 1/5 úspešnosti zmeny σ a nakreslite graf, kde na vodorovnej osi bude počet generácií a na zvislej momentálna hodnota rodiča. Porovnajte priebeh tohto grafu s obdobným grafom, kde sa v algoritme smerodajná odchýlka σ nebude meniť a ostane konštantná, teda rovná jednej.

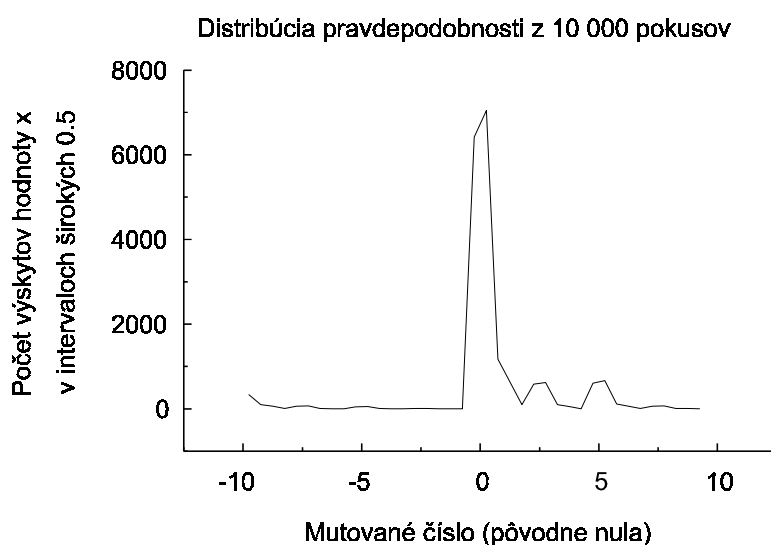
8.3 Distribúcia pravdepodobnosti a generovanie náhodných čísel

Prečo je gaussovská distribúcia lepšia ako zmena bitov v binárnom kóde? Majme reálne číslo z intervalu $[-10,10]$ rovné povedzme nule, zakódované do 12 binárnych číslic. Bude transformované na $\frac{0-(-10)}{10-(-10)}(2^{12}-1)=\lceil 2047.5 \rceil$, čo sa binárne zakóduje 100000000000.

Mutáciou každého bitu by sme dostali čísla

000000000000 $\rightarrow -10.0$	100000100000 $\rightarrow +0.5873$
110000000000 $\rightarrow +5.00366$	100000010000 $\rightarrow +0.805861$
101000000000 $\rightarrow +2.50305$	100000001000 $\rightarrow +0.41514$
100100000000 $\rightarrow +1.25275$	100000000100 $\rightarrow +0.21978$
100010000000 $\rightarrow +0.27595$	100000000010 $\rightarrow +0.1221$
100001000000 $\rightarrow +0.15018$	100000000001 $\rightarrow +0.0732601$

Ako možno vidieť, mutované čísla nemajú dobrú distribúciu okolo nuly. Nasledujúci príklad potvrdzuje, že to nie je náhoda, čiže tento problém existuje všeobecne.



Obrázok 8.3. Distribúcia pravdepodobnosti výsledku mutácie binárneho čísla 100000000000 je veľmi nerovnomerná, a dokonca nie je symetrická! Prečo? Dôvodom je tzv. *Hammingova bariéra*: binárne čísla 100000000000 a 011111111111 sú susedmi v reálnej reprezentácii, no bolo by potrebné zmeniť hodnoty všetkých 12 bitov, aby sme z prvého čísla dostali to druhé.

Binárne číslo 100000000000 bolo mutované 10000-krát, každý bit v každej mutácii s pravdepodobnosťou 1/12 (viac ako jeden bit mohol zmeniť svoju hodnotu). Po transformácii

výsledných binárnych čísel na reálne čísla sme interval $[-10,10]$ rozdelili do 40 oblastí a spočítali, koľko čísel sa nachádzalo v každej oblasti (viď obr. 8.3).

Ako dostať náhodné čísla s normálnym rozdelením podľa gaussovskej distribúcie? Algoritmy 8.3 a 8.4 sa zakladajú na použití Boxovej a Mullerovej metódy, implementáciu môžete nájsť v knihách Numerical Recipes (existujú pre Pascal, C, aj Fortran [12]). Tieto pseudopascalovské algoritmy vracajú náhodnú premennú s nulovou strednou hodnotou a jednotkovou smerodajnou odchýlkou. Používajú pritom generátor náhodných čísel z intervalu $(0,1]$ s uniformným rozdelením - teda každé číslo z tohto intervalu dostanete s rovnakou pravdepodobnosťou. Pokiaľ je k dispozícii iba generátor náhodných prirodzených čísel, vygenerujte také číslo, preveďte na reálne číslo a vydeľte ho maximálnou možnou hodnotou dosiahnuteľnou generátorom - to by malo zodpovedať výsledku funkcie random. Algoritmy 8.3 a 8.4 vlastne vyrábajú vždy dve náhodné premenné s normálnym rozdelením, *Gauss1* (*Gauss2*) a *save*. Prvú dávajú ako výsledok hneď, druhú pri ďalšom volaní funkcie.

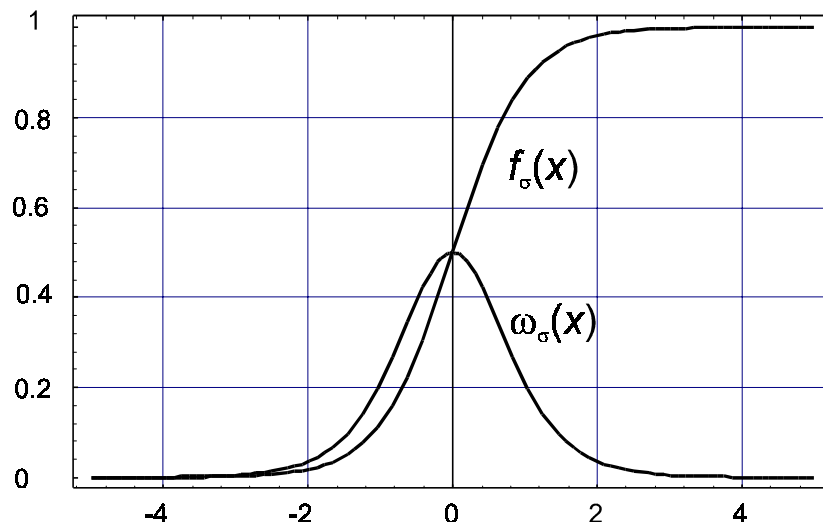
Zložitejší, no v mnohých prípadoch rýchlejší je tzv. trapézový algoritmus, opísaný v [13], ktorého kód vo Fortrane je uvedený v [4].

Pretože sa však generátor náhodných čísel s normálnou distribúciou používa často a podstatne ovplyvňuje časovú náročnosť výpočtu, obrátime pozornosť na logistickú distribúciu (viď obr. 8.4). Táto distribúcia je veľmi podobná normálnej distribúcii, ale je jednoduchšia na výpočet a pomerne často nám môže vyhovujúco slúžiť ako jednoduchšia alternatíva. Formálne sa tvar logistickej distribučnej funkcie $f_{\sigma}(x)$ dá vyjadriť nasledovne,

$$f_{\sigma}(x) = \frac{1}{1 + e^{-x/\sigma}} \quad (8.4)$$

kde funkcia $\omega_{\sigma}(x)$ predstavuje odpovedajúcu funkciu hustoty rozdelenia náhodnej premennej

$$\omega_{\sigma}(x) = f'_{\sigma}(x) = \frac{1}{\sigma} \frac{e^{x/\sigma}}{(1 + e^{x/\sigma})^2} \quad (8.5)$$



Obrázok 8.4. Zobrazenie logistickej distribučnej funkcie $f_{\sigma}(x)$ a jej hustoty $\omega_{\sigma}(x)$ pre $\sigma=0$.

kde parameter σ s kladnou hodnotou (približne zodpovedajúci štandardnej odchýlke u normálnej distribúcie) určuje "šírku" funkcie hustoty $\omega_\sigma(x)$. Distribúcia $f_\sigma(x)$ mapuje celú reálnu os na otvorený interval $(0,1)$, $f_\sigma: \mathbb{R} \rightarrow (0,1)$. Môže sa použiť na jednoduchú konštrukciu generátora náhodných reálnych čísel zachovávajúcich logistickú distribúciu. Keď vyriešime rovnicu

$$r = \frac{1}{1 + e^{-x/\sigma}} \quad (8.6)$$

pre $r \in (0,1)$ dostávame

$$x = \sigma \ln \frac{r}{1-r} \quad (8.7)$$

Teda keď je r náhodné číslo s uniformnou distribúciou, potom x určené pomocou (8.7) je náhodné číslo s nulovou strednou hodnotou, splňujúce logistickú distribúciu. Ako interpretovať a určiť parameter σ ? Požadujeme, aby bolo 100 p % náhodných čísel (pre $0 < p < 1$) generovaných z intervalu $(-p,p)$; potom

$$\int_{-p}^p \omega_\sigma(x) dx = p \quad (8.8)$$

```
function Gauss1 : real;
begin
  if help = 0 then
    begin
      repeat
        v1:=2.0*random - 1.0;
        v2:=2.0*random - 1.0;
        r:= v12 + v22;
      until (r<1.0);
      fac:= $\sqrt{-2.0*\ln(r)/r}$ ;
      save:=v1*fac;
      Gauss1:=v2*fac;
      help:=1;
    end else
    begin
      help:=0;
      Gauss1:=save;
    end
  end
end
```

Algoritmus 8.3. Generátor náhodného čísla s normálnym (gaussovským) rozdelením s nulovou strednou hodnotou a jednotkovou smerodajnou odchýlkou. Verzia podľa Numerical Recipes [12]. Pred aktiváciou položíme externú premennú `help:=0`. Premenná `save` je tiež externá.

```
function Gauss2 : real;
begin
  if help = 0 then
    begin
      a:= $\sqrt{-2.0*\ln(\text{random})}$ ;
      b:=2* $\pi$ *random;
      save:=a*cos(b);
      Gauss2:= a*sin(b);
      help:=1;
    end else
    begin
      help:=0;
      Gauss2:=save;
    end
  end
end
```

Algoritmus 8.4. Generátor náhodného čísla s normálnym (gaussovským) rozdelením s nulovou strednou hodnotou a jednotkovou smerodajnou odchýlkou. Jednoduchšia verzia (no vzhľadom k použitiu trigonometrických funkcií výpočtovo o niečo náročnejšia) používaná Schwefelom [4]. Pred aktiváciou položíme externú premennú `help:=0`. Premenná `save` je tiež externá.

Pri využití vzorcov (8.4) a (8.5) pre $f_{\sigma}(x)$ a $\omega_{\sigma}(x)$ po jednoduchých úpravách dostaneme

$$\sigma = p \left(\ln \frac{1+p}{1-p} \right)^{-1} \quad (8.9)$$

Napríklad ak $p=0.99$ (teda 99% generovaných náhodných čísel patrí do intervalu $(-p,p)$), potom

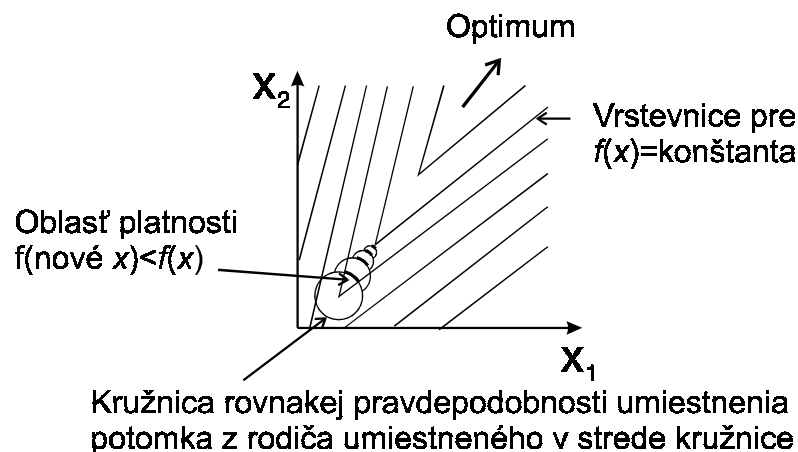
$$\ln \frac{1+p}{1-p} = \ln 199 \approx 5.3 \quad (8.10)$$

To znamená, že za predpokladu, že 99% náhodných čísel patrí do intervalu $(-p,p)$, by σ malo byť približne päťkrát menšie ako p .

Podobne ako pre simulované žihanie, aj pre evolučnú stratégiu sa dokázalo [1,14], že potenciálne poskytuje globálny extrém optimalizovanej funkcie $f(x)$. Za predpokladu regulárnosti optimalizovaného problému (optimalizovaná funkcia je spojitá, definičný obor funkcie tvorí uzatvorenú množinu a platí niekoľko ďalších podmienok väčšinou splnených u inžinierskych problémov) je možné dokázať konvergenčný teorém (prehľad v [7], všeobecne [15, 16]). Konvergenčný teorém tvrdí, že globálne optimum sa dosiahne s jednotkovou pravdepodobnosťou; nehovorí však, po akom počte generácií.

8.4 Mnohočlenné stratégie a kovariancie

Schwefel a jeho spolupracovníci [3-4,7] navrhli ďalšie, sofistikovanejšie verzie evolučnej stratégie, takže v súčasnosti už možno hovoriť o celej triede evolučných stratégií. Odpadá tu nutnosť zavádzania pravidiel pre zmeny štandardnej odchýlky, čo mohlo viesť k predčasnému ukončeniu optimalizácie, ako je to na obrázku 8.5.



Obrázok 8.5. Ukážka zlyhania prístupu zmenšovania štandardnej odchýlky riadenej pravidlom 1/5 úspešných pokusov. Toto pravidlo sa pôvodne vytvorilo na to, aby algoritmus rýchlo našiel minimum, keď sa dostane do jeho blízkosti. Vtedy nájde menej ako 1/5 lepších riešení, a teda so zmenšením štandardnej odchýlky začne prehľadávať bližšie okolie. Keď však má funkcia "úzke údolie", alebo sú aktívne iné ohraničenia, dĺžka kroku bude stále klesať bez toho, že by sa zvyšovala úspešnosť algoritmu, a ten potom "zamrzne" ďaleko od minima.

Schwefel novšie zaviedol mnohočlenné evolučné stratégie, označované $(m+1)$ alebo (m, l) , ktoré imitujú ďalšie základné princípy organickej evolúcie. Pracuje sa tu potom s celým súborom rodičovských vektorov $\mathbf{x}$, ktorých počet je m . V prípade stratégie PLUS $(m+1)$ vygenerujeme z týchto rodičov l potomkov. Potom takto navrhnuté vektory - potomkov a pôvodné "rodičovské" vektory - zoradíme spolu podľa optimálnosti riešení a prvých m najlepších jedincov - riešení zoberieme ako rodičov do ďalšej populácie. V prípade stratégie ČIARKA (m, l) vyberáme budúcich m rodičov iba z práve vygenerovaných l potomkov a rodičia týchto potomkov vymierajú. Život každého jedinca je obmedzený iba na jednu generáciu. Okrem mutácie sa pri oboch stratégiách používa kríženie, teda čiastočná výmena informácií medzi vektormi reálnych čísel (viď obr. 8.6), na rozdiel od kríženia binárnych reťazcov používaného pri genetických algoritmoch. Kríženie dvoch alebo viacerých rodičov vždy produkuje jedného potomka na rozdiel od genetických algoritmov, kedy dvaja rodičia generujú vždy dvoch potomkov. Rovnako ako pri genetických algoritmoch sa chromozóm môže vybrať ako rodič aj niekoľkokrát.

kríženie "priemerom"								
-3.91	8.84	-3.98	-2.51	-5.76	-1.40	-3.33	8.26	rodič_1
6.91	8.46	-1.33	-5.66	6.66	-0.75	9.13	-0.40	rodič_2
1.50	8.65	-2.67	-4.09	0.45	-1.08	2.90	3.93	potomok = $\frac{\text{rodič}_1 + \text{rodič}_2}{2}$
diskrétné kríženie								
-3.91	8.84	-3.98	-2.51	-5.76	-1.40	-3.33	8.26	rodič_1
↓			↓		↓		↓	
-3.91	8.46	-1.33	-2.51	6.66	-1.40	-3.33	-0.40	potomok
	↑	↑		↑			↑	
6.91	8.46	-1.33	-5.66	6.66	-0.75	9.13	-0.40	rodič_2

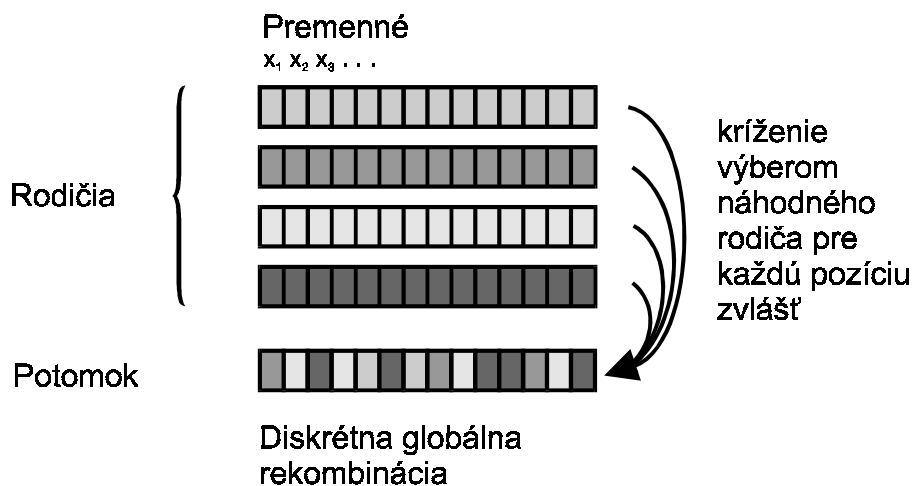
Obrázok 8.6. Dva z mnohých druhov kríženia používaných pri evolučnej stratégii. Kríženie "priemerom" zoberie dva vektory, sčíta hodnoty ich prvkov na zodpovedajúcich miestach a vydolí celý vektor dvoma. Kríženie "diskrétné" preberá hodnoty na zodpovedajúcich miestach náhodným výberom z jedného alebo druhého vektora "rodičovských jedincov".

Najväčším "objavom" evolučnej stratégie je však možnosť nebrať jedinou hodnotu štandardnej smerodajnej odchýlky pre všetky hodnoty optimalizovaného vektora reálnych čísel $\mathbf{x}$, ale uchovávať pre každú premennú z vektora $\mathbf{x}$ vlastnú hodnotu smerodajnej odchýlky (rozptyl). Každý jedinec je teda tvorený dvoma vektormi: vektorom $\mathbf{x}$ a vektorom smerodajných odchýlok σ . Nesie si tak so sebou aj informáciu o distribúcii pre novú mutáciu. Mutácie sa všeobecne robia rovnako ako pri stratégii $(1+1)$ s pomocou štatistického rozdelenia tak, že drobné zmeny sú pravdepodobnejšie a závažné zmeny nepravdepodobné. Táto tendencia sa s časom zvyšuje spolu s približovaním populácie k optimu, teda blízko optima sú mutácie už len malé. Ako ale poznať, že sme už blízko optima? Mnohočlenné evolučné stratégie túto otázku obchádzajú meta-evolučnou samoadaptačnou technikou, kedy sa aj veľkosť mutácií v priebehu algoritmu mení. Evolúcia teda nebude prebiehať iba pri premenných určujúcich riešenie, ale súčasne sa bude vyvíjať informácia o tom, ako každé z riešení bude generovať nového potomka. Táto informácia je uložená v tzv. strategických premenných. Keď zmena aktuálnej hodnoty jednej premennej veľmi nemení zodpovedajúcu hodnotu optimalizovanej funkcie, potom vývoj hodnoty smerodajnej odchýlky priradenej tejto premennej bude smerovať k väčším číslam. Teda zmeny majú inú mierku pri premenných, ktoré majú väčší vplyv na hodnotu funkcie ako iné premenné. Každý nový jedinec je

ohodnotený účelovou funkciou. Učenie prebieha na dvoch úrovniach, jedna úroveň hodnotí dosiahnuté výsledky funkcie, druhá úroveň predstavuje schopnosť jedinca učiť sa zakódovanú v smerodajnej odchýlke určujúcej veľkosť mutácie (aj keď táto schopnosť sa zhodnocuje len nepriamo pomocou ohodnotení výsledkov funkcie).

Pri použití smerodajnej odchýlky pre každú premennú poskytuje väčšinou lepšie výsledky "čiarková" stratégia, kde populácia rodičov celkom vymiera. Aj keď momentálne môže dôjsť k zabudnutiu najlepšieho doteraz nájdeného riešenia, v dlhodobom meradle táto stratégia funguje lepšie, lebo sa dostane cez lokálne minimá, kedy nejaké riešenie má síce momentálne vysokú hodnotu optimalizovanej funkcie, ale nastavenie smerodajných odchýlok je zlé. Zabúdanie rodičovských chromozómov preferuje vybudovanie vnútornej štruktúry - teda smerodajných odchýlok - optimálne nastavených na ďalší vývoj. Takýto prístup potom funguje lepšie pre optimalizáciu zašumených alebo časovo premenných funkcií. Hodnoty vektora x si môžeme predstaviť ako analógiu fenotypu a hodnoty smerodajných odchýlok ako genotyp. Výberom určitého pomeru počtu rodičov a potomkov $m//l$ možno nastavovať rýchlosť konvergenzie evolučnej stratégie. Keď chceme rýchlu, no lokálnu konvergenciu, zvolíme si silne reštriktívnu selekciu, napr. (5,100). Keď hľadáme globálne optimum, musíme selekciu zmierniť, napr. (15,100). Pre už zmienenú sférickú funkciu je ideálny pomer rodičia/potomkovia asi 1/7, no rodičov musí byť pritom rádovo aspoň desať.

Rodičia pre kríženie sa vyberajú z celej populácie s rovnakou pravdepodobnosťou, nezáleží teda na optimálnosti riešenia (konvergencia je dostatočne zabezpečená selekciou potomkov pre ďalšiu generáciu rodičov). Pre kríženie sa väčšinou používa "diskrétné kríženie", známe z analógie s genetickými algoritmi, kde potomok postupne preberá hodnoty z jedného alebo druhého rodiča s rovnakou pravdepodobnosťou (viď. obr. 8.6). Kríženie teraz už zahŕňa nielen výmenu hodnôt vektora x , ale aj hodnôt vektora zodpovedajúcich smerodajných odchýlok σ .



Obrázok 8.7. Globálne uniformné kríženie (alebo inak globálna rekombinácia), kedy potomok má viac ako dvoch rodičov (neexistovalo by rozlíšenie na "mužské" a "ženské" pohlavie, uvedený prístup by zodpovedal viacerým rovnocenným pohlaviom). Každý prvok vektora riešenia (premenná) sa berie z náhodne zvoleného rodičovského vektora.

Ďalším z možných spôsobov kríženia je kríženie priemerom, ktoré však často vedie k príliš rýchlej lokálnej konvergencii.

Najnovšie sa objavili ďalšie druhy kríženia, ktoré sa použili tak v genetických algoritmoch, ako aj v evolučnej stratégii [17]. Dva vektory x_1 a x_2 produkujú dvoch potomkov y_1 a y_2 , ktoré

sú lineárnou kombináciou svojich rodičov

$$\begin{aligned} y_1 &= a x_1 + (1-a) x_2 \\ y_2 &= (1-a) x_1 + a x_2 \end{aligned} \quad (8.11)$$

Okrem tohto kríženia sa ešte navrhlo diskkrétne globálne uniformné kríženie, ktoré si hodnoty vektorov x a σ nevyberá postupne náhodne z dvoch vektorov, no môže si vyberať zo všetkých rodičovských vektorov (viď. obr. 8.7). Obdobou “kríženia priemerom” pre globálne uniformné kríženie je použitie hodnôt jedného rodiča pre všetky premenné, zatiaľ čo druhý rodič sa vyberá pre každú premennú zvlášť. Podľa Schwefelových skúseností [2,3] sa osvedčilo použitie rôznych druhov kríženia pre rozdielne časti chromozómov, napr. diskkrétne kríženie s dvoma rodičmi pre premenné x_i spolu s krížením priemerom (prípadne aj jeho globálnym variantom) pre smerodajné odchýlky a uhly α opísané ďalej.

Pre zmenu smerodajných odchýlok sa pri viacčlenných evolučných stratégiách už nepoužíva deterministický algoritmus, ako bolo pravidlo 1/5 úspešnosti. Namiesto toho sa smerodajné odchýlky podrobia evolučnému procesu. Každá smerodajná odchýlka je teraz mutovaná

$$\sigma_i' = \sigma_i \exp(N(0, \Delta\sigma_0)) \quad (8.12)$$

kde $N(0, \Delta\sigma_0)$ je náhodná premenná vygenerovaná s normálnym rozdelením so stredom v 0 a smerodajnou odchýlkou $\Delta\sigma_0$. Táto smerodajná odchýlka $\Delta\sigma_0$ je parametrom metódy, konštantou, ktorá je rovnaká pre všetky smerodajné odchýlky všetkých premenných v celom časovom priebehu algoritmu. Novovytvorená smerodajná odchýlka σ_i' potomka príslušná nejakej i -tej hodnote x_i vo vektore premenných x sa hneď použije na vygenerovanie novej hodnoty

$$x_i' = x_i + N(0, \sigma_i') \quad (8.13)$$

Okrem vlastnej hodnoty premennej vo vektore a jej smerodajnej odchýlky môže byť súbor premenných charakterizovaný aj ďalším vektorom “strategických” premenných, ktorý riadi koreláciu mutácií. Pre dve premenné s rôznymi hodnotami smerodajnej odchýlky vrstevnice pravdepodobnosti umiestnenia mutovaného vektora nepredstavujú kružnicu, ale elipsu. Táto elipsa je však orientovaná v smere súradnicových osí. Môžeme si však predstaviť, že optimum nie je umiestnené v smere dlhšej osi elipsy, ale niekde naprieč. Ideálne by potom bolo, keby sme mohli túto elipsu vrstevníc pravdepodobností umiestnenia mutovaného vektora natočiť tak, aby hlavná os elipsy smerovala k optimu. Tak by mutovaný vektor mal najväčšiu možnosť čo najviac sa priblížiť k optimu (viď obr. 8.8). Toto natočenie možno zaistiť kovarianciami. Vektor “strategických” premenných teda môže pre n -rozmerný vektor x zahŕňať n rozptylov $c_{ii} = \sigma_i^2$ rovnako ako $n(n-1)/2$ kovariancií c_{ij} zovšeobecnenej n -rozmernej normálnej distribúcie s hustotou pravdepodobnosti vektora mutácií z

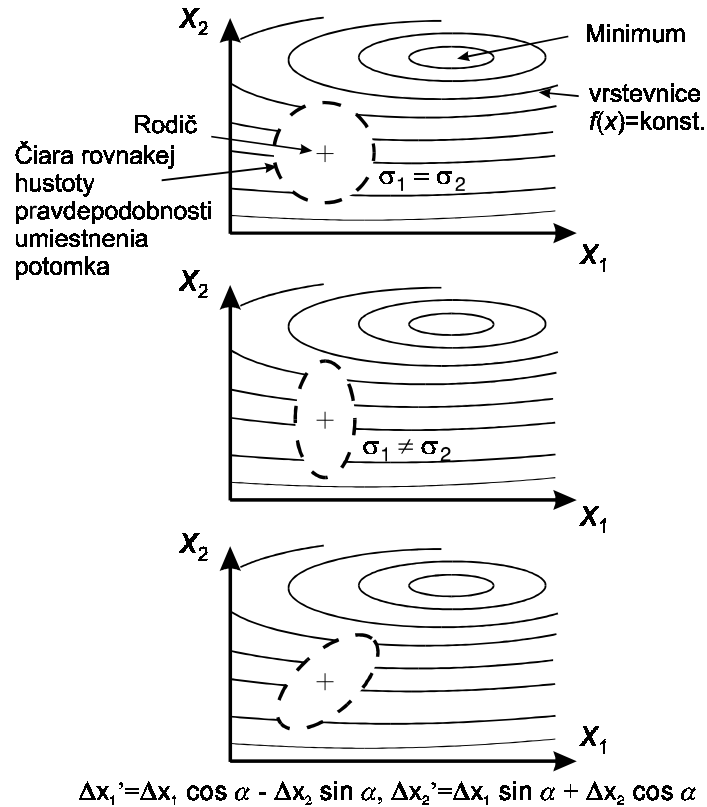
$$p(z) = \frac{\exp\left(-\frac{1}{2} z^T C z\right)}{\sqrt{(2\pi)^n \cdot \det C}} \quad (8.14)$$

Na zaistenie kladnosti a konečnosti kovariančnej matice $C^{-1} = c_{ij}$ používa algoritmus zodpovedajúce rotačné uhly α_j ($-\pi \leq \alpha_j \leq \pi$) namiesto koeficientov c_{ij} . Tieto uhly sú v nasledovnom vzťahu k variáciám a kovarianciám:

$$\tan(2\alpha_{ij}) = \frac{2c_{ij}}{\sigma_i^2 - \sigma_j^2} \quad (8.15)$$

Osi mutačných elipsoidov (povrchu rovnakej hustoty pravdepodobnosti umiestnenia potomka vytvoreného mutáciou z rodiča umiestneného v strede elipsoidu) sú potom rovnobežné s koordinačnými osami iba vtedy, ak je kovariančná matica diagonálna. Vo

všeobecnejšom prípade nenulových kovariancií môže byť elipsoid ľubovoľne natočený v priestore riešení a mutácie premenných sú lineárne korelované. Takto vytvorení potomkovia sa môžu adaptovať na ľubovoľný výhodný smer prehľadávania, čo je výhodné v prípade existencie úzkych údolí v optimalizovanej funkcii.



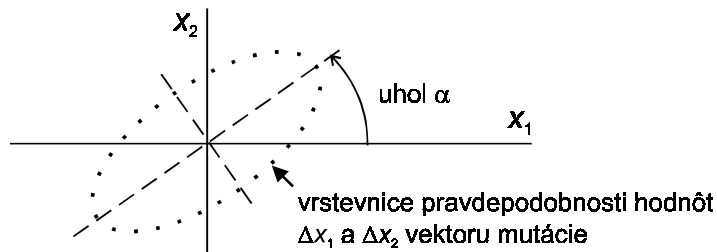
Obrázok 8.8. Prvý graf ukazuje čiary rovnakej hustoty umiestnenia potomka z rodičovského chromozómu umiestneného v strede čiarkovaného oválu. Keď sa štandardné odchýlky pre obidve premenné rovnajú, čiara je tvorená kružnicou. Želali by sme si však, aby potomok mal väčšiu možnosť umiestniť sa v smere minima označeného sústrednými vrstevnicami. Keď zmeníme štandardné odchýlky, dostaneme z kružnice určujúcej pravdepodobné umiestnenie potomka elipsu. No osi tejto elipsy sú stále rovnobežné so súradnicovými osami. Natočíť elipsu tak, aby svojou hlavnou osou smerovala k minima, dokážeme iba pootočením o uhol α , ktoré je realizáciou kovariancie.

Mutácie, ktoré berú do úvahy aj kovarianciu, sa robia nasledovne

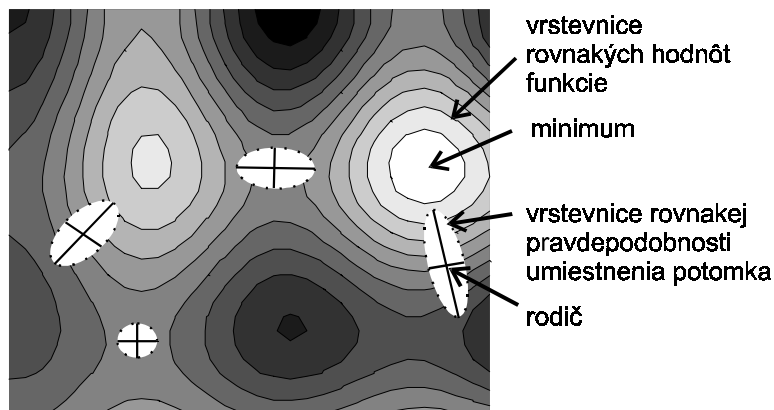
$$\begin{aligned}\sigma_i' &= \sigma_i \cdot \exp(\tau_0 \cdot N(0,1) + \tau \cdot N_i(0,1)) \\ \alpha_j' &= \alpha_j + \beta \cdot N_j(0,1) \\ x_i' &= x_i + z_i(\sigma', \alpha')\end{aligned}\tag{8.16}$$

Výraz $N(0,1)$ označuje vygenerovanú normálne distribuovanú jednorozmernú náhodnú premennú pri nulovej strednej hodnote a jednotkovej smerodajnej odchýlke, zatiaľ čo $N_i(0,1)$ označuje, že náhodná premenná je vždy znova vygenerovaná pre každú hodnotu počítadla i . Tento súbor vzorcov sa použil pre $\forall i \in \{1, \dots, n\}$, $\forall j \in \{1, \dots, n \cdot (n-1)/2\}$. Globálny faktor $\exp(\tau_0 \cdot N(0,1))$ dovoľuje celkovú zmenu veľkosti všetkých mutácií, zatiaľ čo $\exp(\tau \cdot N_i(0,1))$

povoľuje individuálnu zmenu strednej veľkosti kroku σ_i . Výraz $\exp(\tau \cdot N_i(0,1))$ je teda individuálnym parametrom generovaným zvlášť pre každú premennú z vektora $\mathbf{x}$, dovoľujúc tak individuálne zmeny "priemerných" zmien σ_i každej premennej x_i vektora $\mathbf{x}$.



Obrázok 8.9. Pootočenie elipsy rovnakých hodnôt pravdepodobnosti vygenerovania zmien Δx_1 a Δx_2 .



Obrázok 8.10. Zobrazenie vrstevníc hodnôt funkcie dvoch premenných s optimami označenými svetlou farbou. Biele bodkované elipsy označujú miesta rovnakej pravdepodobnosti umiestnenia potomka vzniknutého mutáciou z rodiča umiestneného v strede elipsy označenom krížením osí. Dlhšia os elipsy je nasmerovaná k optimu tak, ako sa to "jedinec" naučil v priebehu optimalizácie, kedy si každá z premenných uchováva vlastnú hodnotu smerodajnej odchýlky a okrem toho si každý "jedinec" ešte uchováva výhodný smer korelovaných odchýlok.

Týmto spôsobom sú mutácie premenných korelované pomocou hodnôt vektora α (viď obr. 8.9). Konštanty τ_0 , τ a β sú parametre, ktoré Schwefel [2,3] navrhuje nastaviť na hodnoty $\tau_0 \propto 1/\sqrt{2\sqrt{n}}$ a $\tau \propto 1/\sqrt{2n}$ a $\beta \approx 0.0873$ ($\approx 5^\circ$ v radiánoch). Pri použití kovariancií, čiže uhlov, nemusíme brať do úvahy vzťahy medzi všetkými premennými, teda počet uhlov môže byť aj menší ako n ($n-1$)/2. To by však vyžadovalo predbežnú analýzu alebo optimalizáciu výberu uhlov, ktoré je možné zanedbať. Konštanty τ_0 , τ sa môžu interpretovať ako "rýchlosti učenia", podobne ako je to pri neurónových sieťach. Najlepšie nastavenie týchto parametrov môže závisieť od druhu optimalizovanej funkcie. Rovnako ako pri stratégii (1+1), musia smerodajné odchýlky zostať väčšie ako nejaká minimálna nenulová hodnota. Schwefel požaduje splnenie podmienky $\sigma_i \geq \varepsilon |x_i|$, kde $\varepsilon > 0$ je malé kladné číslo. Pokiaľ uhly α_j prekročia hranice $(-\pi, \pi)$, používame zrkadlový odraz podobne ako v procedúre Perturbation opísanej pri stratégii (1+1).

Žiaľ, generovať náhodný vektor mutácií s členmi $z_i(\sigma', \alpha')$ uvedenými v (8.16) s distribúciou pravdepodobnosti $p(\mathbf{z})$ uvedenou v (8.14) nemusí byť triviálnym problémom. Namiesto priameho generovania súradníc sa preto používa postupná rotácia. Povedzme, že máme dve premenné x_1 a x_2 so smerodajnými odchýlkami σ_1 a σ_2 , takže vrstevnice rovnakých hodnôt pravdepodobnosti by vytvárali elipsu s osami rovnobežnými s hlavnými osami. Korelačný koeficient c_{12} zodpovedá uhlu α , o ktorý sa táto elipsa pravdepodobnosti pootočí (viď obr. 8.9 a 8.10). Takže ak sú pôvodné odchýlky premenných x_1 a x_2 so smerodajnými odchýlkami σ_1 a σ_2 rovné Δx_1 a Δx_2 , potom odchýlky upravené pomocou korelačného koeficientu majú tvar $\Delta x_1' = \Delta x_1 \cos \alpha - \Delta x_2 \sin \alpha$, $\Delta x_2' = \Delta x_1 \sin \alpha + \Delta x_2 \cos \alpha$.

Pre tri premenné by sa museli urobiť tri nasledujúce rotácie: v rovine $(\Delta x_1, \Delta x_2)$ o uhol α_1 s výsledkom $\Delta x_1'$ a $\Delta x_2'$, v rovine $(\Delta x_1', \Delta x_3)$ o uhol α_2 s výsledkom $\Delta x_1''$ a $\Delta x_3'$ a v rovine $(\Delta x_2', \Delta x_3')$ o uhol α_3 s výsledkom $\Delta x_2''$ a $\Delta x_3''$. Výsledné zmeny premenných by teda boli $\Delta x_1''$, $\Delta x_2''$ a $\Delta x_3''$.

Pre viac premenných sa daný prístup jednoduchšie vyjadří pomocou rotačných matic. Každá matica zodpovedá jednému uhlu, teda jednej kovariancii dvojice premenných vektora $\mathbf{x}$. Budeme mať $n(n-1)/2$ rotačných matic $R(\alpha_{ij}) = (r_{kl})$. Tieto matice sú jednotkové, teda na hlavnej diagonále majú samé jednotky a inde samé nuly, s výnimkou štyroch prvkov: $r_{ii} = r_{jj} = \cos \alpha_{ij}$, a $r_{ij} = -r_{ji} = -\sin \alpha_{ij}$. Trigonometrické výrazy sú teda umiestnené vždy v priesečníkoch riadkov a stĺpcov i a j . Násobenie týmito maticami zodpovedá vlastne transformácii súradníc vzhľadom na osi i a j pri otočení o uhol α_{ij} . Môžeme násobiť vždy prvú maticu $R(\alpha_{1,2})$ vektorom odchýlok premenných $(\Delta x_1, \Delta x_2, \dots, \Delta x_n)$ zodpovedajúcich premenným $x_1, x_2, \dots, x_n$ so smerodajnými odchýlkami $\sigma_1, \sigma_2, \dots, \sigma_n$, a potom násobíme ďalšiu maticu $R(\alpha_{2,3})$ výsledným vektorom predchádzajúceho násobenia $(\Delta x_1', \Delta x_2', \Delta x_3', \dots, \Delta x_n)$, atď. Jednoduchšie je však najskôr vynásobiť všetky matice medzi sebou a až výsledok násobiť vektorom $(\Delta x_1, \Delta x_2, \dots, \Delta x_n)$. Formálne sa to dá vyjadriť vzorcom

$$\text{rotácia}(\Delta x_1, \Delta x_2, \dots, \Delta x_n) = \left(\prod_{i=1}^{n-1} \prod_{j=i+1}^n R(\alpha_{ij}) \right) \cdot (\Delta x_1, \Delta x_2, \dots, \Delta x_n) \quad (8.17)$$

Práca Rudolpha [18] potvrdzuje správnosť tohto prístupu. Rudolph [18] ukázal, že tento prístup garantuje ortogonálnosť transformácií (teda okrem iného zachovanie dĺžky vektorov a uhlov medzi nimi) a súčasne umožňuje vytvorenie akýchkoľvek ortogonálnych transformácií. Použitie rotačných uhlov teda nijako neobmedzuje možnosť vygenerovať akúkoľvek mutáciu.

Už zmienená viacčlenná evolučná stratégia dovoľuje každému členovi vytvoriť niekoľko potomkov. Z týchto (stratégia ČIARKA) a prípadne ešte z pôvodných členov - rodičov (stratégia PLUS) sa vyberie ďalšia skupina rodičov pre novú generáciu na základe ich zoradenia pomocou im zodpovedajúcich hodnôt minimalizovanej funkcie. Pseudokód je uvedený v algoritme 8.5.

Pri krížení nekombinujeme iba hodnoty premenných $x_1, x_2, \dots, x_n$, ale aj strategické premenné, teda smerodajné odchýlky $\sigma_1, \sigma_2, \dots, \sigma_n$ a uhly $\alpha_1, \alpha_2, \dots, \alpha_{n(n-1)/2}$. Môžu sa pritom použiť rôzne typy rekombinačných operátorov pre hodnoty premenných, smerodajné odchýlky a uhly. Štandardné nastavenie parametrov evolučnej stratégie je $\sigma_{ini} = 3$ pre funkcie, o ktorých nič nevieme, počet rodičov $|P_t| = 15$, počet potomkov $|Q| = 100$, stratégia výberu je ČIARKA, teda $P_t := \{\text{najlepšie chromozómy z } Q\}$. Čo sa týka smerodajných odchýlok a uhlov, najjednoduchšie je použiť iba jednu smerodajnú odchýlku pre celý chromozóm s diskretným krížením. O niečo zložitejšie funkcie vyžadujú použitie celého vektora smerodajných odchýlok pre každý chromozóm znova s diskretným krížením, ale ešte bez použitia uhlov α . Konečne najkomplexnejší algoritmus používa aj $n(n-1)/2$ uhlov α a kríženie priemerom medzi týmito uhlami.

Evolučná stratégia: pokročilá mnohočlenná verzia

```

1  t=0;  $\sigma := \sigma_{ini}$ ;
2   $P_0 := \{\text{náhodne generovaná populácia chromozómov}\}$ ;
3  Ohodnot' každý chromozóm z  $P_0$  funkčnou hodnotou;
4  while  $t < t_{max}$  or dostatočne dobré riešenie nie je nájdené do
5    begin  $t := t + 1$ ;
6       $Q = \{\text{nové chromozómy vytvorené krížením a mutáciou z}$ 
        náhodne vybraných chromozómov z  $P_{t-1}\}$ ;
7      Ohodnot' každý chromozóm z  $Q$  funkčnou hodnotou;
8       $P_t := \{\text{najlepšie chromozómy z } Q \text{ alebo z } P_{t-1} \cup Q\}$ ;
9    end;

```

Algoritmus 8.5. Pokročilá mnohočlenná verzia evolučnej stratégie typu (m,n) alebo $(m+n)$ v závislosti od tvorby novej populácie P_t . Počet chromozómov podpopulácie Q je ohraničený podmienkou $|Q| \geq |P_t|$. Chromozómy sú tvorené vektormi reálnych premenných spolu so štandardnými odchýlkami a prípadne variáciami.

8.5 Ukážka aplikácie algoritmu na Rosenbrockovu funkciu

Ako príklad riešenia optimalizačného problému pomocou evolučnej stratégie si tu uvedieme riešenie Rosenbrockovej funkcie [19]. Táto funkcia dvoch premenných má hlboké "údolie", a iba jedno minimum (v bode $x_1=1, x_2=1$, viď obr. 8.11). Takáto funkcia by sa ľahko riešila gradientovou metódou, no napr. pre genetický algoritmus predstavuje táto funkcia "nočnú moru", pretože genetický algoritmus nemá nastavený smer mutácií = smer pohybu v priestore riešení a nie je schopný "zahýbať". Genetický algoritmus by pomerne rýchlo našiel údolie vyznačené na obrázku 8.11 prerušovanou čiarou, no nebol by už schopný dostať sa v rozumnom čase do globálneho optima. Funkcia je definovaná nasledovne:

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2 \quad (8.18)$$

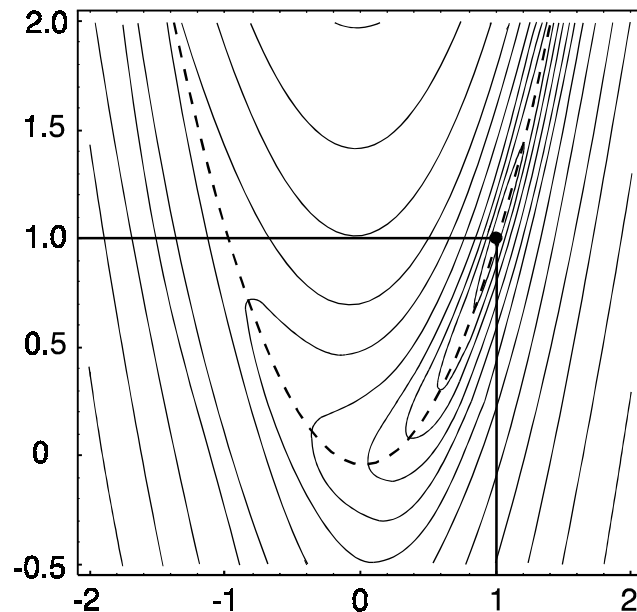
Minimum $f(1,1)=0$ je na obrázku 8.11 vyznačené veľkým bodom.

Táto funkcia sa riešila mnohočlennou verziou evolučnej stratégie s použitím smerodajných odchýlok pre každú premennú zvlášť a s použitím kovariancie medzi premennými, teda uhla α . Každý chromozóm sa teda skladal z piatich členov:

$$\text{chromozóm} = \{x_1, x_2, \sigma_1, \sigma_2, \alpha\} \quad (8.19)$$

Počiatkové hodnoty premenných sa pre $\mathbf{x}$ zvolili náhodne z intervalu $(-100,100)$, smerodajné odchýlky boli rovné 3.0 a uhol α bol nulový. Použili sme stratégiu $(15,100)$, to znamená, že z 15 chromozómov vždy vyrobíme krížením a následnou mutáciou 100 chromozómov. Pri problémoch väčšej dimenzie by sa samozrejme hodilo viac chromozómov. Rodičovské chromozómy po vygenerovaní potomkov vymažeme. Vzhľadom na to, že z potomkov berieme vždy striktnie iba najlepšie chromozómy, nie je treba prepočítavať funkčné hodnoty na fitness ako u genetických algoritmov, stačí iba zoradiť 100 novovytvorených chromozómov podľa veľkosti funkčných hodnôt od najmenej po najväčšiu (alebo naopak, pokiaľ hľadáme maximum) a zobrať iba prvých 15 chromozómov do ďalšej generácie.

Použilo sa uniformné kríženie s náhodným výberom vždy dvoch rodičov pre premenné x_1, x_2 a kríženie priemerom pre strategické premenné $\sigma_1, \sigma_2, \alpha$. Pre počet premenných $n=2$ sa použili konštanty s nasledujúcimi hodnotami: $\tau_0 = 1/\sqrt{2\sqrt{n}} = 0.5946$; $\tau = 1/\sqrt{2n} = 0.5$; $\beta = 0.0873$.



Obrázok 8.11. Vrstevnicový graf Rosenbrockovej funkcie, kde horizontálna os je označená x_1 a vertikálna os x_2 . Veľký bod predstavuje umiestnenie minima v “údolí” vyznačenom čiarkovanou parabolou.

Jednotlivé premenné sú teda po križení chromozómov mutované pomocou vzorcov

$$\begin{aligned} \sigma'_i &:= \sigma_i \exp(\tau_0 N(0,1)) \exp(\tau N_i(0,1)), \text{ teda} \\ \sigma'_1 &:= \sigma_1 \exp(0.5946 \cdot \text{konštanta}) \exp(0.5 N(0,1)) \\ \text{a } \sigma'_2 &:= \sigma_2 \exp(0.5946 \cdot \text{konštanta}) \exp(0.5 N(0,1)) \end{aligned} \quad (8.20)$$

$$\alpha' := \alpha + \beta \Delta\alpha, \text{ teda } \alpha' := \alpha + 0.0873 N(0,1) \quad (8.21)$$

kde $N(0,1)$ je výsledok generátora náhodných čísel s normálnym rozdelením, s nulovou strednou hodnotou a jednotkovou smerodajnou odchýlkou a konštanta := $N(0,1)$. Pretože pre dve premenné existuje iba jedna kovariancia, používame iba jeden uhol α bez indexov. Pri výpočte zmien premenných x_1 a x_2 používame rovnaký generátor, iba s novozmutovanou smerodajnou odchýlkou:

$$\Delta x_1 = N(0, \sigma'_1) \text{ a } \Delta x_2 = N(0, \sigma'_2) \quad (8.22)$$

Tieto hodnoty ešte “pootočíme” pomocou uhla α :

$$\Delta x'_1 = \Delta x_1 \cos \alpha' - \Delta x_2 \sin \alpha', \Delta x'_2 = \Delta x_1 \sin \alpha' + \Delta x_2 \cos \alpha' \quad (8.23)$$

Teraz konečne môžeme dostať nové hodnoty vektora $\mathbf{x}$: $x'_1 = x_1 + \Delta x'_1$ a $x'_2 = x_2 + \Delta x'_2$ a z týchto hodnôt vypočítať funkčnú hodnotu $f(x'_1, x'_2)$, ktorou bude chromozóm ohodnotený. Nasleduje príklad tvorby nového chromozómu (ktorý bude zaradený medzi 100 nových) z dvoch náhodne zvolených rodičovských chromozómov (z 15 možných).

V prvej fáze sa za premennú x_1 dosadí premenná z druhého rodiča a za premennú x_2 hodnota tejto premennej z prvého rodiča (u každej z premenných sme si “hodili korunou”, z ktorého rodiča ju zobrať), zatiaľ čo namiesto hodnôt σ_1 , σ_2 a α sa dosadili priemery obidvoch rodičov.

premenná	x_1	x_2	σ_1	σ_2	α	$f(x_1, x_2)$
rodič_1 = {	0.415204	0.129087	0.039035	0.023917	0.202127	0.529536 }
		↓	↓	↓	↓	
			0.0286605			
potomok = {	0.316526	0.129087	$\frac{0.039035 + 0.018286}{2}$		0.399014	XXXXXXX }
	↑		↑	↑	↑	
rodič_2 = {	0.316526	0.129394	0.018286	0.031714	0.595900	0.552434 }

Samozrejme, hodnota $f(x_1, x_2)$ potomka sa spočíta až po skončení mutácií. Ako prvé sa mutujú hodnoty σ_1 , σ_2 a α :

$$\begin{aligned}
&\text{konštanta} = N(0, 1) = 0.528959, \\
&\sigma_1' = \sigma_1 \exp(0.5946 \cdot \text{konštanta}) \exp(0.5 \cdot N(0, 1)) \\
&\quad = 0.0286605 \exp(0.5946 \cdot 0.528959) \exp(0.5 \cdot 0.380766) = 0.047485 \\
&\sigma_2' = \sigma_2 \exp(0.5946 \cdot \text{konštanta}) \exp(0.5 \cdot N(0, 1)) \\
&\quad = 0.027815 \exp(0.5946 \cdot 0.528959) \exp(0.5 \cdot (-0.419033)) = 0.030895 \\
&\alpha' = \alpha + 0.0873 N(0, 1) \\
&\quad = 0.399014 + 0.0873 \cdot 0.548100 = 0.446863 \\
&\Delta x_1 = N(0, \sigma_1') = N(0, 0.047485) = 0.023767 \\
&\Delta x_2 = N(0, \sigma_2') = N(0, 0.030895) = -0.042886 \\
&\Delta x_1' = \Delta x_1 \cos \alpha - \Delta x_2 \sin \alpha \\
&\quad = 0.023767 \cos(0.446863) - (-0.042886) \sin(0.446863) = 0.038562 \\
&\Delta x_2' = \Delta x_1 \sin \alpha + \Delta x_2 \cos \alpha \\
&\quad = 0.023767 \sin(0.446863) + (-0.042886) \cos(0.446863) = -0.024535 \\
&x_1' = x_1 + \Delta x_1' = 0.316526 + 0.038562 = 0.355087 \\
&x_2' = x_2 + \Delta x_2' = 0.129087 + (-0.024535) = 0.104552 \\
&f(x_1, x_2) = 0.462288
\end{aligned}$$

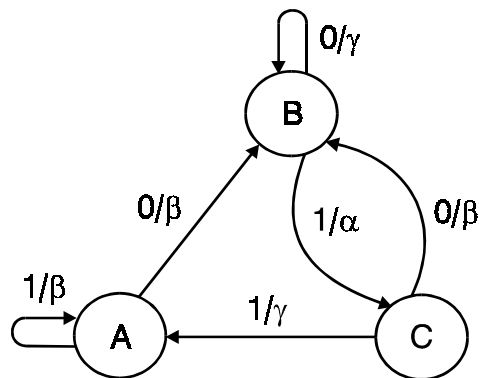
Výsledný potomok upravený mutáciou smerodajných odchýlok, uhla α a mutáciou premenných x_1 , x_2 s použitím nových mutovaných odchýlok, s nasledovným pootočením týchto súradníc pomocou zmutovaného uhla α vyzerá nasledovne:

premenná	x_1	x_2	σ_1	σ_2	α	$f(x_1, x_2)$
potomok = {	0.355087	0.104552	0.047485	0.030895	0.446863	0.462288 }

Úloha 8.2. Použite mnohočlenné stratégie na nájdenie optima Rosenbrockovej funkcie. Výpočet zopakujte 100-krát a vypočítajte priemerný počet generácií, ktorý bol potrebný, aby najlepšia nájdená funkčná hodnota bola nižšia ako 10^{-6} . Porovnajme tento priemerný počet generácií s priemerným počtom generácií potrebných na nájdenie rovnakej funkčnej hodnoty, ale bez použitia kovariancie. Podobne skúste odstrániť kríženie pri zachovaní použitia kovariancií a diskutujte vplyv tvaru alebo typu funkcie na efektívnosť použitia kríženia alebo kovariancie pri nájdení minima. (Symetria a multimodálnosť indikujú pravdepodobnú výhodnosť aplikácie kríženia, minimá uložené v "údoliach" umiestnených naprieč osí súradnicového systému sa nájdu rýchlejšie za použitia kovariancií, čo platí aj pre "viacrozmerné údolia".)

8.6 Evolučné programovanie

Evolučné programovanie predstavuje prístup využívajúci simuláciu evolúcie na populácii súťažiacich algoritmov. Metóda evolučného programovania bola už spomenutá v podkapitole 3.4. V tejto kapitole upriamime našu pozornosť na súvislosť tejto metódy s evolučnou stratégiou. V počiatkoch tohto prístupu navrhnutého okolo r. 1962 Lawrencom Fogelom [20-22] bolo cieľom predpovedať nové stavy prostredia a udalosti a na základe predpovedí prispôbiť vlastnú reakciu v závislosti od daného cieľa, čo je základnou podmienkou inteligencie organizmu. Evolučné programovanie rovnako ako evolučná stratégia kladú dôraz skôr na podobnosť správania sa rodičov a ich potomkov, ako na napodobňovanie genetických operátorov tak, ako to môžeme pozorovať v biologických procesoch na molekulárnej úrovni. Fogel začal u evolučného programovania s predpoveďami symbolických reťazcov generovaných konečnými automatmi z tzv. markovovských procesov a nestacionárnych časových radov.



Obrázok 8.12. Trojstavový konečný automat. Vstupné symboly sú uvedené naľavo, výstupné napravo od lomítka. Štartovacia pozícia je v A (podľa Fogela [21]).

Spracovanie vstupu automatom sa v prípade tabuľky 8.1 začína v stave A, na ktorý prichádza vstupná hodnota 1. Pretože z "krúžku" A vedie šípka 1/β, presunieme sa pozdĺž tejto šípky. Hodnota za lomítkom, teda β, tvorí odpoveď automatu. Šípka je však slučkou a dostávame sa zasa do stavu A. Nový vstup je 0 a pozdĺž šípky 0/β sa dostávame do stavu B, pričom β je odpoveď automatu. Takto pokračujeme, až pokiaľ na súbor signálov 100110 nedostaneme odpoveď β β γ α γ β. Túto odpoveď ohodnotíme hodnotiacou funkciou, nech už táto funkcia vyzerá akokoľvek.

Tabuľka 8.1. Odpovede konečného automatu z obrázka 8.12 v závislosti od vstupných symbolov. Ohodnotenie automatu je dané funkciou ohodnocujúcou výstupné symboly v závislosti od vstupných (stavu prostredia).

Momentálny stav	A	A	B	B	C	A
Vstupný symbol	1	0	0	1	1	0
Budúci stav	A	B	B	C	A	B
Výstupný symbol	β	β	γ	α	γ	β

U takýchto konečných automatov existuje v zásade päť prirodzených typov mutácií: zmena výstupného symbolu, zmena budúceho stavu pre nastavený konečný stav a výstupný symbol (premiestnenie šípky), pridanie stavu, vymazanie stavu, zmena počiatočného stavu.

Na rozdiel od Johna Kozu [23], ktorý vo svojom genetickom programovaní pracuje s programami vo forme ohodnotených stromov a používa okrem mutácie aj kríženie, Fogel sa na počiatku svojich úvah obmedzil na konečné automaty bez použitia kríženia. Evolučné programovanie napodobňuje totiž vývoj druhov, ktoré medzi sebou súťažia a medzidruhovú kríženie väčšinou nenastáva. Princíp súťaženia je v tomto prístupe podobný ako pri evolučnej stratégii; zo skupiny rodičov a potomkov sa vyberú do ďalšej populácie iba striktné najlepší jedinci. Z každého rodiča sa pritom vyrába mutáciou iba jeden potomok, teda zo skupiny rodičov a potomkov sa vyberie lepšia polovica ako rodičia do ďalšej populácie.

V nových variantoch evolučného programovania majú aj horší potomkovia ešte možnosť, pokiaľ sú vybraní do slabšej skupiny. Skupinky sú náhodne vybrané z nových (t.j. mutovaných) jedincov aj z rodičovských jedincov. V skupinkách potom dochádza k "turnaju", potomkovia sú zoradení podľa svojich výsledkov v turnaji a tí s najlepšími výsledkami sú vybraní do ďalšej populácie. Počet jedincov v populácii nemusí byť konštantný a jedinec môže produkovať aj viac potomkov.

Evolučné programovanie sa postupne rozvíjalo rovnakým smerom ako evolučné stratégie. Výsledné algoritmy sú v súčasnej dobe také podobné, že ich rozlišovanie nemá význam. Pretože však tieto algoritmy majú rozdielny historický pôvod, v USA sa evolučné programovanie stále udržiava ako samostatný algoritmus [24-35]. Teória evolučných algoritmov s dôrazom na evolučné stratégie a evolučné programovanie je prehľadne uvedená napr. v [36].

Spoločnou črtou evolučného programovania a evolučnej stratégie, ktorá sa líši od genetických algoritmov, je reprezentácia jedinca (riešenia). Tá sa podriadiť problému. Povedzme, že máme problém, ktorého riešenie zahŕňa nielen reálne premenné, ale aj ďalšie prvky. Tak je to napr. pri zadaní neurónovej siete [24], kde okrem váhových parametrov (čísel charakterizujúcich spojenie medzi uzlami siete) určujeme aj vlastnú topológiu, teda počet uzlov a existenciu spojení medzi nimi. Nemusíme sa predsa zaoberať tým, ako celú sieť zakódovať do binárneho reťazca, ako to robia genetické algoritmy. Stačí si predstaviť jedinca ako sieť a mutácie budú rôzneho druhu, jedny pre váhové faktory, teda reálne čísla, a iné pre topológiu siete, teda pre pridávanie alebo uberanie uzlov a spojení medzi nimi.

8.7 Záver

Zoznam viac ako 300 aplikácií evolučných stratégií sa dá nájsť napr. v správe SyS-2/92 [10]. Evolučné stratégie sú schopné riešiť multidimenzionálne nelineárne problémy s mnohými lokálnymi optimami a s lineárnymi alebo nelineárnymi obmedzeniami. Optimalizovaná funkcia môže tiež byť výsledkom simulácie, nemusí byť v presne definovanej forme. To sa týka tiež obmedzení, ktoré môžu predstavovať výsledok napr. metódy konečných prvkov (FEM). Evolučné stratégie boli vytvorené na optimalizáciu vektorov reálnych čísel, no môžu slúžiť tiež ako heuristika pre NP-úplné kombinatorické problémy, ako je problém obchodného cestujúceho, alebo pre problémy s časovo premennými alebo zašumenými hodnotami optimalizovanej funkcie.

Distribučná funkcia mutácie neurčuje smer, ale iba veľkosť mutácie, takže k veľkej mutácii môže dôjsť s rovnakou pravdepodobnosťou aj v opačnom smere, s novým potomkom ďalej od minima. Zaujímavé však je, že pri návrhu pravdepodobnosti výskytu nového potomka sa neoplatí posunúť stred kružnice alebo elipsy tak, aby sa nezhodoval s pozíciou rodiča, ale aby bol podstatne posunutý smerom, v ktorom nastal predchádzajúci posun od rodiča k potomkovi - terajšiemu rodičovi. Síce by to zvýšilo momentálnu úspešnosť generovaných potomkov, no v dlhodobom meradle pre zložité multimodálne funkcie nie je tento prístup všeobecne výhodný. Pre jednoduchšie funkcie však výhodný byť môže.

Literatúra

- [1] I. Rechenberg: *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. (Evolution Strategy: Optimization of technical systems by means of biological evolution.) Fromman-Holzboog, Stuttgart 1973, 1993 2nd edn.
- [2] H.-P. Schwefel: *Numerische Optimierung von Computermodellen mittels der Evolutionsstrategie*, Birkhäuser, Basel 1977.
- [3] H.-P. Schwefel. *Numerical Optimization for Computer Models*. J. Wiley, Chichester, UK 1981.
- [4] H.-P. Schwefel: *Evolution and Optimum Seeking*. J. Wiley, New York 1995.
- [5] H.-P. Schwefel and R. Manner (eds.): *Proceedings of the First International Conference on Parallel Problem Solving from Nature (PPSN I)*. Springer Verlag, Berlin 1991.
- [6] F. Hoffmeister and T. Bäck: *Genetic Algorithms and Evolution Strategies: Similarities and Differences*. University of Dortmund, Dept. of CS, SyS-1/92. (1990, 1992).
Dostupné na anonymnom ftp serveri
<ftp://lumpi.informatik.uni-dortmund.de/pub/EA/papers/sys-91-2.ps.gz>
- [7] T. Bäck, F. Hoffmeister, and H.-P. Schwefel: A Survey of Evolution Strategies. In R.K. Belew and L.B. Booker (eds.): *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA 1991, pp. 2-9.
Dostupné na anonymnom ftp serveri
<ftp://lumpi.informatik.uni-dortmund.de/pub/GA/papers/icga91.ps.gz>
- [8] T. Bäck and H.-P. Schwefel: An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation* **1**(1) (1993) 1-23. Dostupné na anonymnom ftp serveri <ftp://lumpi.informatik.uni-dortmund.de/pub/EA/papers/ec1:1.ps.Z>
- [9] T. Bäck, G. Rudolph, and H.-P. Schwefel: Evolutionary Programming and Evolution Strategies: Similarities and Differences. In D.B. Fogel and W. Atmar (eds.): *Proc. of the 2nd Annual Conf. on Evolutionary Programming*. Evolutionary Programming Society, La Jolla, CA 1993, pp. 11-22. Dostupné na anonymnom ftp serveri
<ftp://lumpi.informatik.uni-dortmund.de/pub/ES/papers/ep93.ps.gz>
- [10] T. Bäck, F. Hoffmeister, and H.-P. Schwefel: *Applications of evolutionary algorithms*. Technical report SYS-2/92, Systems Analysis Research Group, University of Dortmund, Department of Computer Science, July 1993. Dostupné na anonymnom ftp serveri <ftp://lumpi.informatik.uni-dortmund.de/pub/EA/papers/ea-app.ps.gz>
- [11] M. Herdy: Application of the Evolution Strategy to Discrete Optimization Problems. In H.-P. Schwefel and R. Männer (eds.): *Proceedings of the First International Conference on Parallel Problem Solving from Nature (PPSN I)*. Springer Verlag, Berlin 1991, pp. 188-192.
- [12] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery: *Numerical Recipes in C*. Cambridge University Press, Cambridge 1997. Dostupné na internetovskej adrese http://www.ulib.org/webRoot/Books/Numerical_Recipes/bookc.html
- [13] J.H. Ahrens and U. Dieter: Computer Methods for Sampling from the Exponential and Normal Distributions. *Communications of ACM* **15** (1972) 873-882, 1047.
- [14] J. Born: *Evolutionstrategien zur numerischen Lösung von Adaptationsaufgaben*. Dissertation A, Humboldt-Universität, Berlin 1978.
- [15] F.J. Solis and R.J.-B. Wets: Minimization by Random Search Techniques. *Math. Operations Research* **6** (1981) 19-30.

- [16] G. Rudolph: Parallel Approaches to Stochastic Global Optimization. In W. Joosen and E. Milgrom (eds.): *Parallel Computing: From theory to Sound Practice, Proceedings of the European Workshop on Parallel Computing*. IOS Press, Amsterdam 1992, pp. 256-267.
- [17] Z. Michalewicz and C. Janikow: Handling Constraints in Genetic Algorithms. In R.K. Belew and L.B. Booker (eds.): *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA 1991, pp. 151-157.
- [18] G. Rudolph: On correlated mutations in evolution strategies. In R. Männer and B. Manderick (eds.): *Parallel Problem Solving from Nature 2 (PPSN II)*. North-Holland, Amsterdam 1992, pp. 105-114.
- [19] H.H. Rosenbrock: An automatic method for finding the greatest or least value of a function. *Comp. J.* **3** (1960) 175-184.
- [20] L.J. Fogel: Autonomous Automata. *Industrial Research* **4** (1962) 14-19.
- [21] L.J. Fogel: On the Design of Conscious Automata. Final Report under Contract No. AF49(638)-1651, AFOSR, Arlington, VA 1966, reference in [33].
- [22] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. J. Wiley, New York 1966.
- [23] J.R. Koza: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA 1992.
- [24] D.B. Fogel, L.J. Fogel, and V.W. Porto: Evolving Neural Networks. *Biological Cybernetics* **63** (6) (1990) 487-493.
- [25] D.B. Fogel and W. Atmar (eds.): *Proceedings of the First Ann. Conf. on Evolutionary Programming*. Evolutionary Programming Society, La Jolla, CA 1993.
- [26] D.B. Fogel and W. Atmar (eds.): *Proceedings of the Second Ann. Conf. on Evolutionary Programming*. Evolutionary Programming Society, La Jolla, CA 1993.
- [27] A.V. Sebald and L.J. Fogel (eds.): *Proceedings of the Third Annual Conference on Evolutionary Programming*. World Scientific Publishers, River Edge, NJ 1994.
- [28] J.R. McDonnell, R.G. Reynolds, and D.B. Fogel (eds.): *Evolutionary Programming IV: The Proceedings of the Fourth Annual Conference on Evolutionary Programming*. MIT Press, Cambridge, MA 1995.
- [29] L.J. Fogel, P.J. Angeline, and T. Bäck (eds.): *Evolutionary Programming V: The Proceedings of the Fifth Annual Conference on Evolutionary Programming*. MIT Press, Cambridge, MA 1996.
- [30] P.J. Angeline, R.G. Reynolds, J.R. McDonnell, and R. Eberhart (eds.): *Evolutionary Programming VI: The Proceedings of the Sixth Annual Conference on Evolutionary Programming*. Springer Verlag, Berlin 1997.
- [31] V.W. Porto, N. Saravanan, D. Waagen, and A.E. Eiben (eds.): *Evolutionary Programming VII: The Proceedings of the Seventh Annual Conference on Evolutionary Programming*. Springer Verlag, Berlin 1998.
- [32] D.B. Fogel (ed.): *Evolutionary Computation: The Fossil Record*, IEEE Press 1998.
- [33] D.B. Fogel: *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, New York 1999.
- [34] K. Miettinen, P. Neittaanmäki, M.M. Mäkelä, and J. Périaux (eds.): *Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming*. J. Wiley 1999.
- [35] L.J. Fogel: *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming*. J. Wiley 1999.
- [36] T. Bäck: *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York 1996.

9. Umelý život

9.1 Čo je to umelý život?

Otázka je, čo je to vlastne umelý život ("Artificial Life"). Pretože definícia tejto novej búrlivo sa rozvíjajúcej vednej oblasti nie je vôbec jasná, dá sa umelý život definovať skôr tým, čo doň rôzni ľudia zahŕňajú. Najčastejšie ide o simuláciu životných prejavov jednotlivcov alebo skupín. Umelý život zasahuje do viacerých už zavedených oblastí - okrem biológie a informatiky aj do fyziky, sociológie, psychológie, ekonómie (modelovanie situácií, kedy si firmy konkurujú alebo kedy spolupracujú, zapojenie teórie hier, riadenie priemyselných komplexov) a filozofie. Témou umelého života sa zaoberajú tak populárno-vedecké knihy [1-4], vedecké knihy [5,6], ako aj početné odborné konferencie [7-17] a časopisy [18-28]. Centrom aktivít skupín zaoberajúcich sa umelým životom je USA, presnejšie povedané inštitút v Santa Fe, laboratóriá v Los Alamos a MIT, a Japonsko. Aj v Európe existujú špecializované konferencie venované otázkam umelého života [29-32]. Na Slovensku sa tiež pracuje v oblasti umelého života, napr. Július Csontó v Košiciach [33,34] a v oblasti umelých agentov Jozef Kelemen [35].

V zásade sú dva prístupy k tejto téme.

Prvý je vizionársky - tvrdiaci, že umelý život znamená vytvorenie a vylepšovanie účinnosti *nových foriem života*, iba v inom médiu ako sú uhľovodíky, teda väčšinou na báze kremíkových čipov v počítačoch (ako napr. počítačové vírusy) alebo v "železe" predstavovanom (nano)robotmi. Proces biologickej evolúcie viedol ku genotypu vytvárajúcemu fenotyp, schopný manipulovať s vlastným genotypom priamo: kopírovať ho, meniť alebo vytvárať celkom nový. Tento prístup síce funguje zatiaľ iba pri umelom živote prezentovanom počítačovými programami, no pri terajšom prudkom rozvoji biotechnológií a génových manipulácií možno predpovedať, že v polovici 21. storočia už môže byť reálny aj "biologický" umelý život (teda pokiaľ ako taký nebudeme chápať už dnes vytvárané rastlinné druhy s geneticky vnesenou informáciou pre vytváranie protilátok proti škodcom, alebo produkujúce látky použiteľné ako vakcína proti cholere či malárii). S týmto prístupom sa prelína z opačnej strany aj "technizácia" ľudského života, od existujúcich transplantátov končatin reagujúcich na nervové povely alebo kardiostimulátorov v srdci až po dnes ešte vzdialenú možnosť priameho napojenia človeka na počítač, doteraz realizovanú iba primitívnymi prostriedkami virtuálnej reality.

Druhý, "prízemnejší" smer tvrdí, že cieľom umelého života je vypracovať nové metódy na *štúdium života*, vedúce k lepšiemu pochopeniu tohto biologického fenoménu. Umelý život poskytuje možnosť robiť experimenty, ktoré by boli v tradičnej biológii extrémne komplikované alebo nemožné. Výhodná je tu možnosť presne opakovať experiment a tiež rýchlosť simulácií v porovnaní s experimentami. Zatiaľ čo klasická biológia sa snaží život analyzovať, umelý život smeruje k syntéze aspoň niektorých čŕt života zo základných vlastností živej hmoty. Kľúčovým rysom tohto oboru je formulovanie hlavných pravidiel správania sa jednotlivých "organizmov" a ich replikácie, pričom cieľom je dosiahnuť evolúciu organizmov rovnako ako vzťahov medzi organizmami ("emergent behavior"), ako je napr. súperenie, spolupráca, parazitizmus, vzťah dravec - korisť alebo vývoj druhov. Veľa prác smeruje k počítačovej simulácii umelých bytostí a umelých svetov, v ktorých tieto bytosti žijú. Predmetom štúdia je teda nielen napodobňovanie pozemského života, ale aj simulácia, ako by život mohol vyzeráť za iných podmienok.

Zakladateľ umelého života Christopher Langton [7] definuje umelý život ako štúdium umelých systémov, ktoré vykazujú správanie charakteristické pre prírodné živé systémy: samoorganizáciu, adaptáciu, evolúciu, ko-evolúciu, metabolizmus atď. Je to snaha o

vysvetlenie fenoménu života v akejkolvek forme, bez ohľadu na špecifické príklady, ktoré sa vyvinuli na Zemi. To zahŕňa biologické a chemické experimenty, počítačové simulácie a čisto teoretické rozbor. Predmetom výskumu sú procesy prebiehajúce na molekulárnej, sociálnej a evolučnej úrovni. Cieľom je extrahovať základné princípy organizácie živých systémov a ich vývoja.

Vytvorenie týchto procesov v inom prostredí (inom ako v počítačoch) ich sprístupňuje novým druhom experimentálnej manipulácie a testovania. Pritom evolúcia hardvéru a syntetická evolúcia biomolekúl sú zaujímavé aj priamo pre prax. Na druhej strane populácie dátových štruktúr v počítači sa dajú použiť na reprezentovanie biologických jedincov (dravca a koristi, mravcov, buniek apod.). Vo väčšine príkladov umelého života populácie dátových štruktúr nereprezentujú priamo žiaden žijúci organizmus alebo proces, ale skôr sa podriaďujú umelo vytvoreným zákonom abstrahovaným zo zákonov platných pre živé organizmy. Najväčší nadšenci pritom tvrdia, že tieto dátové štruktúry sú fakticky živé. Steven Levy [1] dôvodí takto: "Život je dynamický fyzikálny proces a keď dokážete duplikovať tieto procesy na inak neživom materiáli - stvorili ste život. Ten môže byť nezávislý na materiáli. Môže dokonca vzniknúť v počítači." Vzhľadom na to, že dosiaľ neexistuje žiadna celkom kvalitná definícia života, je toto tvrdenie zdrojom búrlivých diskusií.

9.2 Definície života

Základný problém, pri ktorom je ťažko hľadať riešenie, je odpoveď na otázku *ako definovať život*? Na to, že už s touto otázkou budú problémy, prišli v 50-tych rokoch vedci, chystajúci sa hľadať život vo vesmíre, ktorí zrazu zistili, že nedokážu definovať, čo vlastne hľadajú. Samozrejme, bolo veľa definícií, no všetky mali svoje výnimky a obmedzenia. Napríklad: život musí byť schopný autoreprodukcie - mulica (sterilný kríženec kobyly a osla, ktorý nie je principiálne schopný ďalšieho rozmnožovania) teda nie je živá?

Vlastnosti definujúce život môžu byť napr. nasledujúce:

1. Život je skôr určitý vzor v čase a priestore ako špecifický materiálny objekt.
2. Je schopný sa reprodukovať, či už sám alebo s pomocou hostiteľského organizmu.
3. Obsahuje informáciu, pomocou ktorej je vybudovaný.
4. Obsahuje metabolizmus premeny jedného druhu hmoty na iný spojený s produkciou energie.
5. Funkčne interaguje s okolím.
6. Jeho časti sa navzájom funkčne dopĺňajú.
7. Je stabilný v meniacom sa prostredí.
8. Je schopný vyvíjať sa.
9. Rastie.

Obrazy organizmu ako stroja a stroja ako organizmu sú staré stáročia. No až teraz sa obidva tieto obrazy pomaly začínajú prekrývať.

Základnými vlastnosťami života, ktoré sa snažíme vytvoriť umelo, sú sebareplikácia, samostatné rozhodovanie = seba-riadenie, limitovaná seba-obnova, evolúcia a učenie. Týmto približujeme stroje živým organizmom. No zatiaľ sa nedarí napodobniť všetky tieto vlastnosti súčasne. Väčšina príkladov umelého života sa zameriava na simuláciu aspoň jednej vlastnosti charakteristickej pre život. Zo všeobecného pohľadu môže ako príklad výtvorov podobných životu slúžiť celosvetová telefónna sústava, vývoj počítačových vírusov, virtuálna realita v počítači alebo počítačové modely rozvoja ekonómie a/alebo ekológie. Priemysel smeruje k používaniu metód založených na biologických princípoch, tieto sú menej náročné na spotrebu materiálu a energie, zložitosť vytváraných vecí a procesov môže dosahovať hranice biologickej integrácie.

Na druhej strane aplikácia metód genetického inžinierstva, aj keď po stáročia v praxi využívaných šľachtiteľmi ovocia alebo domácich zvierat, robí život až teraz inžiniersky spracovateľným.

Živé systémy majú vnútornú štruktúru vykazujúcu niekoľko základných vlastností:

- absenciu striktného centrálného riadenia,
- čiastočnú autonómiu nižších podštruktúr,
- vysokú úroveň prepojenosti medzi podštruktúrami,
- nelineárne vzájomné ovplyvňovanie podštruktúr v sieti.

Z týchto vnútorných vlastností potom vyplývajú vonkajšie vlastnosti živých systémov:

- adaptabilita (v rámci učenia jedného živočícha, drobné "taktické" zmeny),
- vývoj (v rámci druhu, zmena základných génových vlastností alebo zmena celkového zamerania vývoja - z adaptácie génov k adaptácii kultúry),
- odolnosť spôsobená redundanciou,
- schopnosť narastania systému bez nutnosti meniť jeho základné vlastnosti (kŕdeľ vtákov alebo kobyliiek môže narastať rádovo bez nutnosti meniť systém správania sa - jediná topológia schopná neobmedzeného rastu a neriadeného učenia je sieť).

Vývoj je podporovaný prepojenosťou jednotlivcov s exponenciálnym rastom možných kombinácií tejto prepojenosti a zároveň variáciami jednotlivcov, akokoľvek väčšinou nevýhodných pre jednotlivcov, avšak v dlhodobom meradle výhodných pre druh ako celok - keď je 99% variácií neúspešných, zvyšné úspešné percento sa rozšíri do celého druhu.

No tento systém má zároveň aj neodstrániteľné nedostatky:

- Neoptimálnosť - vzhľadom na to, že je systém určený pre neustále sa meniace podmienky, neustále hľadanie výhodnejších ciest a kontrola možnej zmeny podmienok znižuje jeho výkonnosť vtedy, keď sú podmienky stabilné a systém našiel optimum.
- Neriaditeľnosť - zložité systémy sa nedajú pre svoju zložitosť riadiť priamo, ich riadenie je možné iba okľukou. Prítomnosť "emergentného" správania vylučuje priame riadenie systému.
- Nepredvídateľnosť - keďže systém nedokážeme dokonale chápať a riadiť, nemôžeme ani na 100% predpovedať jeho správanie.
- Dlhé a postupné budovanie systému - nutný vývoj potrebuje čas.

Živý systém využíva pri učení drobné chyby, čo mu umožňuje vyhnúť sa väčším zlyháním.

Štúdie zložitého správania sa vychádzajúceho z jednoduchých pravidiel sľubujú byť základom pre pochopenie komplikovaného kolektívneho správania sa zložitejších organizmov. Tieto štúdie sú schopné lepšie predpovedať uvedené emergentné javy.

Princípy pri tom objavené môžu byť užitočné aj pre praktické účely softvérovej evolúcie inteligentných robotov, nanotechnológie (robotiky a výroby počítačových komponent alebo napríklad lekárskeho "nástrojov" na molekulárnej úrovni) a robustných a adaptabilných počítačových programov pri využití života podobných procesov.

9.3 Umelá inteligencia a agenti

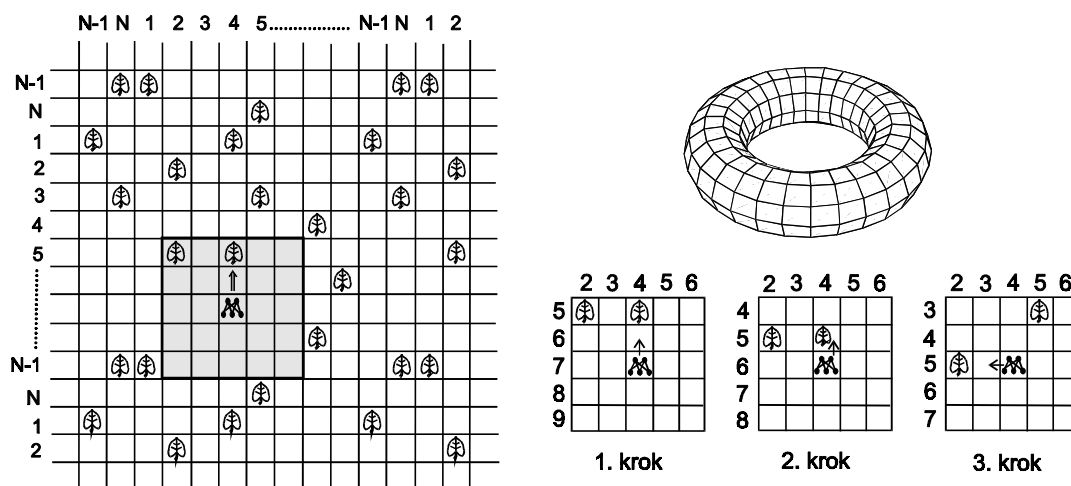
Vo vzťahu ku klasickej umelej inteligencii je umelý život orientovaný "opačným smerom". Umelá inteligencia postupuje "zhora nadol", základom je dopredu zadaný komplexný problém s presne stanovenými pravidlami ako hranie šachu, chápanie textu alebo diagnóza. Program musí byť inteligentný hneď od začiatku a závisí od programátora, ako to dosiahnuť. Umelý

život, na rozdiel od programov umelej inteligencie, vzniká "zdola nahor". Začína od najjednoduchších elementárnych jednotiek a postupne sa dostáva k vývoju a emergentným javom. Je určený na "prežitie" v komplexnom prostredí s meniacimi sa podmienkami. Umelý život sa snaží vytvoriť iba základné pravidlá, ktoré budú tak dobre vybrané, aby zložité správanie sa (a možno v budúcnosti aj istá "inteligencia") vzniklo (emergovalo) po čase spontánne.

Umelá inteligencia a umelý život majú k sebe najbližšie v type programov nazvaných *autonómny agent* [36,37]. To sú programy, ktoré obsahujú špecifický druh vnímania okolitého prostredia. Na základe týchto vstupných informácií ovplyvňujú vlastný stav a stav susedných agentov alebo prostredia samotného. Agenti pracujú v operačnom systéme, databáze alebo v počítačovej sieti. Agent buď zberá informácie alebo ovplyvňuje svoje okolie, individuálne aj "v tímovej práci". Po počiatočnom nastavení pracuje bez ďalšieho ovplyvňovania programátorom. Pod názvom "distribuovaná umelá inteligencia" tento prístup rieši praktické problémy, akými sú napr. riadenie letovej prevádzky letiska alebo prevádzky továrne, kedy každý stroj má svojho "agenta" starajúceho sa o dodávky energie, surovín a o odvoz hotových produktov.

Zatiaľ najrozšírenejšími príkladmi umelého života (aj keď mnohí by ich za umelý život nepovažovali) sú počítačové vírusy a bunkové (celulárne) automaty, ktoré sú v najjednoduchšom prípade reprezentované Conwayovou hrou "život".

Počítačový vírus [38,39] sa môže považovať aj za autonómneho agenta. V praxi sa počítačový vírus líši od autonómneho agenta tým, že je jednoduchší a má len jeden hlavný cieľ: rozmnožovať sa a rozširovať do ďalších počítačov. Okrem toho je tu otázka ničivých vedľajších účinkov vírusov. Pre umelý život sú vírusy špeciálne zaujímavé tým, že majú vlastnosti veľmi podobné biologickým vírusom. Akcie vírusov by teoreticky mohli byť aj pozitívne, napríklad udržiavanie integrity databázy.

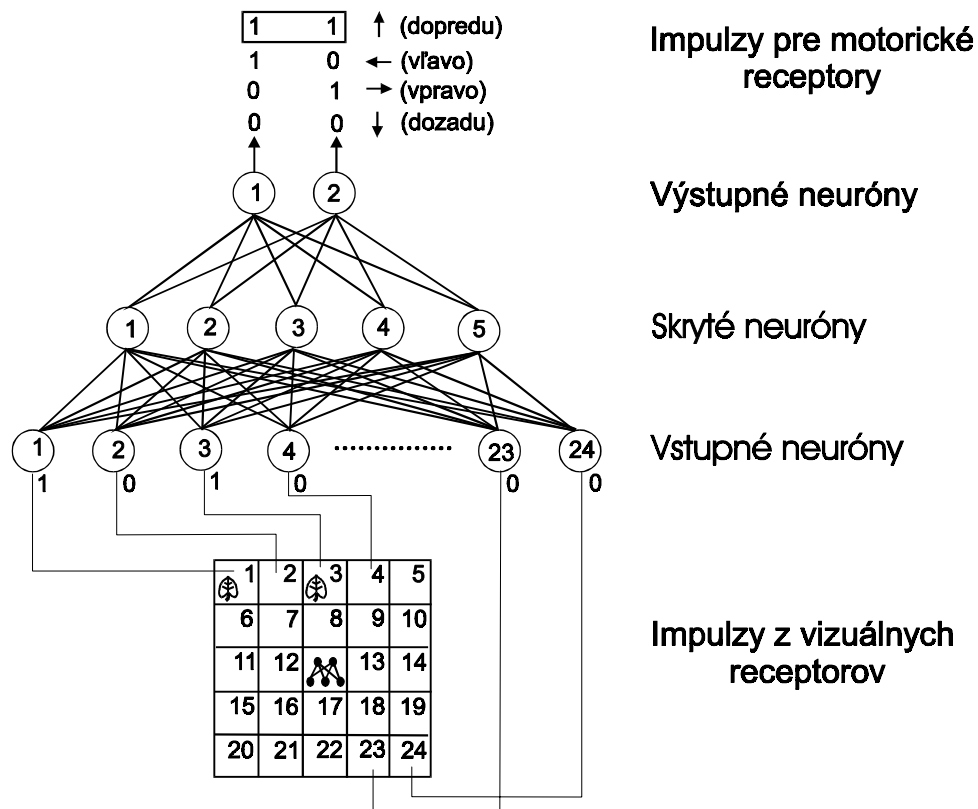


Obrázok 9.1. Ilustratívny príklad organizmu, ktorý je riadený neurónovou sieťou a nachádza sa na pravouhlej mriežke typu $N \times N$. Organizmus sa živí potravou znázornenou náhodne rozmiestnenými lístkami a má za cieľ pozbierať za svoj život (daný počtom horizontálnych alebo vertikálnych krokov na susedné pole) čo najväčší počet lístkov. Je schopný vnímať iba svoje najbližšie (vytieňované) okolie. Pri rekognoskácii prostredia musí určiť smer, kde je najviac najbližších lístkov, a pohybovať sa týmto smerom. Pravouhlá mriežka je stočená do toroidu - pneumatiky, aby sa nemohlo stať, že organizmus dôjde na okraj mriežky. Vpravo dole sú znázornené prvé tri kroky, pričom predpokladáme, že neurónová sieť vždy rozhodla správne a organizmus získal celkovo jeden lístok.

Základným dôvodom na to, aby sa počítačové vírusy považovali za živé, je ich schopnosť autoreplikácie. Počítačový vírus je sekvenciou strojového kódu, ktorá sa skopíruje na jeden či viac "hostovských" programov, keď sú tieto zaktivované. Keď sa tieto infikované programy spustia, kód vírusu sa aktivuje a vírus sa šíri ďalej. Podobne ako biologické vírusy ani počítačové vírusy nie sú schopné "žiť" samostatne, potrebujú hostiteľa.

Podobný typ programov, "červy" (worms, Internet Worm 1988) môžu žiť nezávisle a cestovať sieťou. Nepripájajú sa na cudzie programy a nemusia ani poškodzovať ostatné programy. Teoreticky môžu rôzne časti "červa" bežať na rôznych strojoch.

Základná vlastnosť, ktorá vírusom aj červom chýba, je schopnosť samostatne sa vyvíjať. Vírus, ktorý by toho bol schopný, by musel byť príliš veľký, a teda ľahko rozpoznateľný a zničiteľný antivírusovými programami.



Obrázok 9.2. Znáznomenie trojvrstvovej neurónovej siete organizmu z obr. 9.1. Táto riadiaca jednotka sa realizuje trojvrstvou neurónovou sieťou. Vstupné neuróny získavajú signály z vizuálnych receptorov pre okolité políčka (0 pre neprítomnosť lístku, inak 1). Skryté neuróny pomáhajú spracovať prijatú informáciu pre výstupné neuróny, ktoré rozhodujú o smere budúceho pohybu organizmu. Spojenie medzi neurónmi sú ohodnotené váhami, ktoré tak ovplyvňujú "inteligenciu" neurónovej siete. Takýchto organizmov je viac, ich úspešnosť v zbere potravy sa ohodnocuje, neúspešné zanikajú a tie najúspešnejšie sa množia, vymieňajú si vzájomne časti "dedičnej informácie" - váhy medzi neurónmi, takže organizmy sa stávajú stále efektívnejšími v zbere potravy.

Virtuálna realita [40] má tiež blízko k umelému životu. Na rozdiel od umelého života je vo virtuálnej realite užívateľ aktívnym účastníkom procesov v umelo vytvorenej (simulovanej) realite. Na rozdiel od virtuálnej reality v simuláciách umelého života jedinci - programy po

počítačnom nastavení zvyčajne už nekomunikujú s užívateľom, ktorý je len pozorovateľom dejov vo virtuálnom svete, do ktorého nezasahuje.

Základom väčšiny simulácií umelého života sú algoritmy dovoľujúce umelým "bytosťam" vyvíjať sa a/alebo meniť svoje prostredie. Každý z týchto algoritmov je vedným odborom sám osebe so širokými vedeckými a priemyselnými aplikáciami. Základné algoritmy spadajú do dvoch odvetví: učiace sa algoritmy reprezentované neurónovými sieťami a evolučné algoritmy reprezentované predovšetkým genetickými algoritmami.

Neurónové siete [41] majú hodnotu samy osebe ako univerzálne aproximátory a dá sa povedať, že samy predstavujú pole výskumu ďaleko rozsiahlejšie ako v súčasnej dobe umelý život. Pre autonómnych agentov potom neurónové siete predstavujú jednu z možných metód rozhodovania a súčasne učenia sa ako komunikovať s okolitým prostredím alebo rozpoznávať informácie.

Vela aplikácií umelého života vybavuje svoje organizmy neurónovou sieťou, ktorá slúži ako umelý mozog schopný učiť sa z príkladov s vopred zadaným výsledkom. Typická je neurónová sieť zostavená zo vstupných neurónov, ktoré sú napojené na informácie získavané z vonkajšieho prostredia, zo skrytých vrstiev neurónov, ktoré vykonávajú väčšinu výpočtov, a z výstupných neurónov, ktoré určujú ďalšiu akciu organizmu - agenta (viď. obr. 9.1 a 9.2). Trénovanie (učenie) neurónovej siete pozostáva z nastavovania sily spojenia medzi jednotlivými neurónmi. Toto trénovanie možno robiť rôznymi prístupmi, medzi iným aj genetickými algoritmami.

Zaujímavé je aj použitie simulátorov umelého života vo výučbe princípov biológie pre deti. Výrazný úspech tu dosiahol napr. Mitchell Resnick [42]. Najpopulárnejší v tomto ohľade je asi program SimLife [43].

Niektoré techniky z *evolučných algoritmov* (kríženie a mutácia), ktorým sme sa venovali v predchádzajúcich kapitolách, slúžia na prenos informácie medzi organizmami a zabezpečujú evolúciu celej generácie - populácie organizmov - agentov.

9.4 Evolúcia pravidiel a Lindenmayerove systémy

Okrem genetických algoritmov, programovania a ďalších metód existujú aj tzv. *Lindenmayerove systémy* [44] produkčných pravidiel používaných na modelovanie rastu a rozvoja organizmov. Tieto systémy sa vyskytujú v mnohých simulátoroch a tiež v počítačovej animácii. Pri simulácii týchto systémov zanedbávame vnútornú štruktúru skúmaných systémov a nahrádzame ju zjednodušenými pravidlami tak, aby výsledný efekt bol čo najbližší skúmanému biologickému efektu. Výsledkom aplikácie takýchto pravidiel môže byť fraktálová štruktúra. Príkladom je tvorba "kríka", kde je L-systém daný nasledovnými pravidlami:

Počiatkový bod F

Produkčné pravidlo $F \rightarrow F [+F][-F]$ (9.1)

F znamená nakresli čiaru (začíname kolmou čiarou z počiatkového bodu so súradnicami 0,0 do koncového bodu so súradnicami 0,10)

[znamená zapamätať si súradnice koncového bodu a zodpovedajúcu zmenu súradníc

$$dx = x_{\text{koncový}} - x_{\text{počiatkový}}, dy = y_{\text{koncový}} - y_{\text{počiatkový}}$$

] znamená vrátiť sa na zapamätané súradnice bodu a zodpovedajúce dx a dy

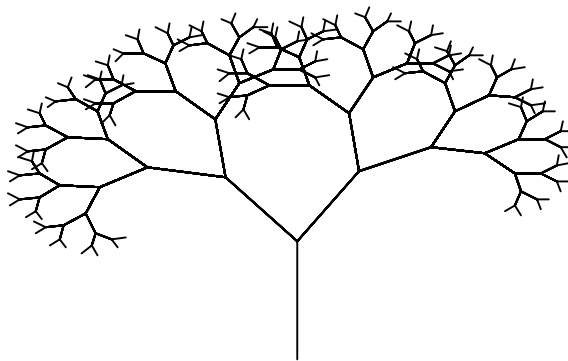
+ znamená otočiť sa doľava (použili sme prepis pre súradnice nového bodu)

$$\begin{aligned} x_{\text{koncový}} &= x_{\text{počiatkový}} + 0.7 (\cos(40^\circ) dx - \sin(40^\circ) dy) \\ y_{\text{koncový}} &= y_{\text{počiatkový}} + 0.7 (\sin(40^\circ) dx + \cos(40^\circ) dy) \end{aligned} \quad (9.2)$$

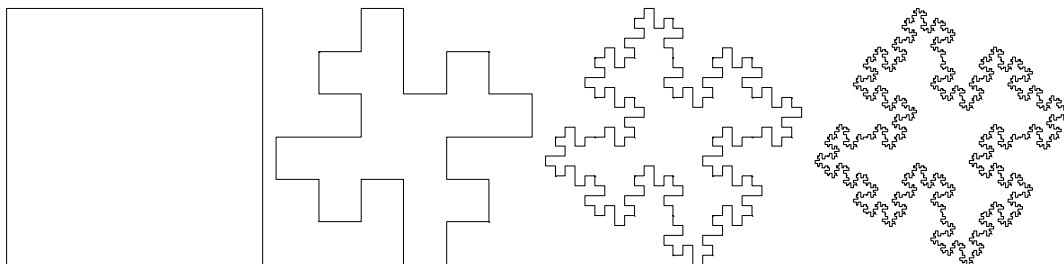
– znamená otočiť sa doprava (použili sme prepis pre súradnice nového bodu)

$$\begin{aligned}x_{\text{koncový}} &= x_{\text{počiatočný}} + 0.7 (\cos(33^\circ) dx + \sin(33^\circ) dy) \\y_{\text{koncový}} &= y_{\text{počiatočný}} + 0.7 (\sin(33^\circ) dx - \cos(33^\circ) dy)\end{aligned}\quad (9.3)$$

Na obrázku 9.3 sme použili rekúziu pravidiel do úrovne 7. Uhol otáčania 40° doľava v rovnici (9.2) a 33° doprava v rovnici (9.3) bol zvolený na začiatku náhodne. Fraktálovú štruktúru vytvorenú pomocou trocha pozmenených pravidiel môžeme vidieť na obr. 9.4.



Obrázok 9.3. Lindenmayerov systém daný rovnicou 9.1, interpretovaný ďalej uvedenými pravidlami a rovnicami 9.2 a 9.3, s rekúziou do úrovne 7.



Obrázok 9.4. Lindenmayerov systém pre počiatočný bod $F+F+F+F$, produkčné pravidlo $F \rightarrow F+F-F-F+F-F-F$. Uhol otáčania v rovniciach (9.2) a (9.3) bude 90° a zmenšenie sa použije iba pri rekúzii. Dostávame tzv. kvadratický Kochov ostrov [45], čo je fraktálová štruktúra (môžeme ísť s rekúziou ľubovoľne hlboko, a pokiaľ dostaneme zobrazený odsek obrázku bez udanej mierky, nie je možné povedať, aký veľký je to odsek). Je to podobné ako pri predchádzajúcom strome, pokiaľ dostaneme obrázok odrezaného konca vetvičky bez mierky, nie je možné povedať, či rekúzia na obr. 9.3 išla do 7. úrovne alebo do milióntej.

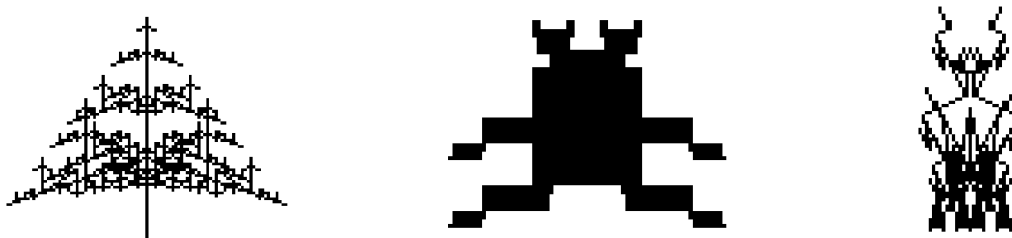
Úloha 9.1. Vytvorte obdobný trojrozmerný stromček ako na obrázku 9.3, kde sa jednotlivé vetvy nebudú skracovať faktorom 0.7, ale náhodným číslom s gaussovským rozdelením pravdepodobnosti so stredom napr. v 0.7, kde sa bude zmenšovať aj hrúbka kmeňa a bližšie vetvy budú vyzeráť hrubšie, kde uhly vetiev v priestore tiež budú dané náhodne. Vyskúšajte aj iné produkčné systémy, napr. počet vetiev bude daný náhodným číslom od 2 do 3.

Fraktály sa však nepoužívajú iba na animácie, ale aj na modelovanie štruktúr a javov v živých organizmoch, ako je sieť ciev a kapilár v ľudskom tele. Tá je taká hustá, že žiadna bunka nie je od nej vzdialená viac ako 3-4 bunky (počet kapilár je asi 10 miliárd), ale sieť nezaberá viac ako 5% objemu tela. Fraktálmi sa dá modelovať aj systém alveol (vzduchových komôrok) v ľudských pľúcach, ktorých je asi 300 000 000 a ktoré spôsobujú,

že 1 cm³ pľúc odpovedá aktívnej respiračnej ploche 300 cm². Podobný systém je tiež vo vnútrajšku ľadvín, ktoré spracujú okolo 1800 litrov krvi za deň pomocou asi 1 000 000 základných filtračných jednotiek, alebo v srdci u Purkyného vlákien elektrochemicky regulujúcich tep. Využitie fraktálov sa zdá byť jedným zo základných princípov morfogenézy a fraktály sa objavujú aj pri geometrickej interpretácii dynamických pochodov (podivné atraktory), čo sa dá využiť napr. pri biorytmoch na konštrukciu defibrilátorov na potlačenie infarktu [46,47]. Fraktálmi sa tiež dajú popísať štruktúry a javy na nižšej, molekulárnej úrovni, napr. u proteínov alebo DNA [48].

Netypickým prístupom pri výberovej funkcii evolučných algoritmov je estetické kritérium, ako ho napr. uplatňuje Karl Sims pri generovaní obrázkov [49]. Sims [50] tiež použil genetický systém na vývoj "umelých bytostí" zložených z blokov, spojených kĺbmi, poháňaných svalmi a riadených "obvodmi". Tieto umelé bytosti sa potom testovali na schopnosť pohybovať sa po povrchu, skákať, plávať a pod., pričom sa vyvinula veľká variabilita bytostí a nových typov pohybov.

V skutočnosti začal s podobným vývojom obrázkov "bytostí" zoológ Richard Dawkins [51] vo svojej populárnej knižke "Blind Watchmaker". Tieto "bytosti" sa volali "Biomorph" podľa surrealistických obrázkov Desmonda Morrisa podobných zvieratám. Pomocou náhodných mutácií sa vždy vytvoril súbor obrázkov, z ktorých si užívateľ vybral ten, čo sa mu najviac páčil, a z toho sa generoval ďalší súbor. Tieto stromom podobné štruktúry boli zakódované vo forme génov, kódujúcich uhol vetvenia, dĺžku vetiev, hĺbku rekurzie, tvar ako priamka, elipsa, obdĺžnik a pod. Takto vznikli napr. nasledujúce obrázky ihličnatého stromu, žaby alebo samuraja, (viď. obr. 9.5). Prehľad podobných programov nájdete napr. na internete [34].



Obrázok 9.5. Strom, žaba a samuraj, obrázky vytvorené algoritmom založeným na rekurzii a pravidlách podobných Lindenmayerovým systémom, kde však sú pravidlá zakódované chromozómami, ktoré prechádzajú drobnými zmenami. Užívateľ má napr. na výber z 8 možných obrázkov a ten, ktorý si vyberie, tvorí základ pre ďalších 8 obrázkov vytvorených pomocou náhodne pozmenených pravidiel. Po niekoľkých desiatkach cyklov výberov sa potom dajú získať zaujímavo vyzerajúce obrázky.

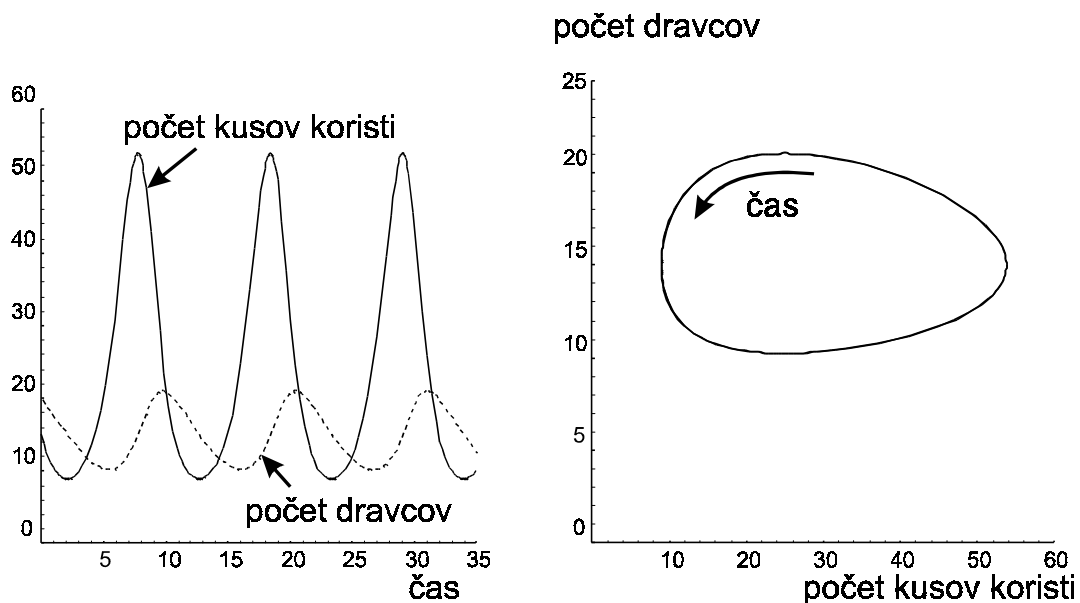
9.5 Evolučná dynamika a modelovanie krdľa

Skôr analytickému spôsobu riešenia problémov sa venuje ekologická a *evolučná dynamika*. Jedným z najznámejších príkladov je asi vzťah dravec - korisť vyjadrený Lotkovou-Volterrovou rovnicou [52-54]. Ide o vyjadrenie vzťahu medzi počtom dravcov a množstvom lovej zveri v závislosti od času. V ideálnom prípade by tieto počty mali byť stabilné, no pri určitej efektivite útokov dravcov a rozmnožovaní dravcov aj koristi nastávajú oscilácie množstva koristi, nasledované osciláciami počtu dravcov. Dá sa to vysvetliť nasledovne: pri počiatkovej chorobe sú dravci zdecimovaní a korisť sa premnoží. Dravci potom majú veľa koristi, bez problémov sa môžu (po zániku choroby) rozmnožovať a ich počet sa tak zväčší

ešte výraznejšie ako množstvo koristi. Teraz sú zasa premnožení dravci, ktorým sa podarí natoľko zredukovať množstvo koristi, že sami začnú umierať od hladu. Oscilácie množstva koristi a dravcov sa potom opakujú už bez nutnosti počiatočného vychýlenia počtu dravcov z rovnovážneho stavu nejakou chorobou. Takéto oscilácie možno napríklad pozorovať u počtov ulovených králikov a rysov vo viac ako storočie starých záznamoch Spoločnosti Hudsonovho zálivu, kde sú maximá počtov ulovených králikov vzdialené od seba vždy desať rokov nasledované maximami počtov ulovených rysov.

Nech $N_1(t)$ je počet kusov koristi (alebo hostiteľa) a $N_2(t)$ je počet kusov dravcov (alebo parazitov) v čase t . Predpokladajme, že za neprítomnosti dravcov sa korisť rozmnožuje rýchlosťou r_1 , zatiaľ čo za absencie koristi dravci hynú rýchlosťou r_2 . Nech b_1 zodpovedá schopnosti dravcov požírať korisť a b_2 vplyvu množstva koristi (a teda nasýtenia dravcov) na ich rozmnožovanie. Najjednoduchšie vyjadrenie je potom

$$\begin{aligned} dN_1(t)/dt &= N_1(r_1 - b_1 N_2) \\ dN_2(t)/dt &= N_2(-r_2 + b_2 N_1) \end{aligned} \quad (9.4)$$



Obrázok 9.6. Prvá závislosť ukazuje oscilácie počtu dravcov a množstva koristi v čase, druhý obrázok je trajektóriou sústavy diferenciálnych rovníc (9.4).

Nech $r_1 = 1.5$, $b_1 = 0.1$, $r_2 = 0.25$, $b_2 = 0.01$ a $N_1(0) = 15$ a $N_2(0) = 20$. Potom môžeme znázorniť priebeh zmien počtov dravcov a koristi v závislosti od času na jednom grafe (viď obr. 9.6), kde je počet dravcov vyznačený prerušovanou čiarou a počet kusov koristi plnou čiarou. Druhý graf zobrazuje vzájomnú závislosť počtu dravcov a množstva koristi, kde sa tieto počty pohybujú v čase stále dookola po zobrazenej trajektórii.

Takýto priebeh vzájomného ovplyvňovania dvoch živočíšnych druhov sa dá modelovať nielen diferenciálnymi rovnicami, ale aj počítačovým experimentom, kedy vytvoríme na mriežke dravcov a koristi, pričom korisť sa rozmnožuje stabilnou rýchlosťou, zatiaľ čo rozmnožovanie dravcov závisí od hustoty koristi.

Pre názornosť však najprv uvedieme príklad simulácie populačného procesu jedného druhu. Keď zanedbáme takmer všetko včítane plošného rozšírenia druhu, dostávame iba dva procesy - zrodenie a smrť.

Ak predpokladáme, že príliš veľká populácia má inhibičný efekt na ďalší rast a tento inhibičný efekt je lineárny, pre zmenu počtu populácie N dostávame pre kladné konštanty r a s Verhulstovu-Pearlovu logistickú rovnicu rozpracovanú v [54]. Verhulst túto rovnicu navrhol už r. 1838 [55], no bola ignorovaná až do roku 1920, kedy ju Pearl a Reed znovu objavili [56]

$$dN/dt = N(r - sN) \quad (9.5)$$

Táto rovnica by mala ideálne fungovať napr. pre kultúru kvasiniek, no prekvapujúco dobre funguje aj pre oveľa zložitejšie problémy. Túto rovnicu si môžeme rozdeliť na rýchlosť rastu $B(N)$ a vymierania $D(N)$, kedy

$$dN/dt = B(N) - D(N) = N[(a_1 - a_2) - (b_1 + b_2)N] \quad (9.6)$$

kedy $r = a_1 - a_2$ a $s = b_1 + b_2$ pre kladné konštanty a_1, a_2, b_1, b_2 a môžeme písať

$$B(N) = N(a_1 - b_1 N) \text{ a } D(N) = N(a_2 + b_2 N) \quad (9.7)$$

Pretože rýchlosť rastu populácie $B(N)$ nemôže byť záporná, vyplýva z toho, že $1 \leq N \leq a_1/b_1$. Čas s medzi jednotlivými udalosťami, teda zrodením alebo smrťou v populácii, je nepriamo úmerný súčtu rýchlostí rastu a vymierania. Pri simulácii sa generuje tiež pomocou náhodnej premennej Y_2 s uniformnou distribúciou z intervalu $(0,1)$, teda

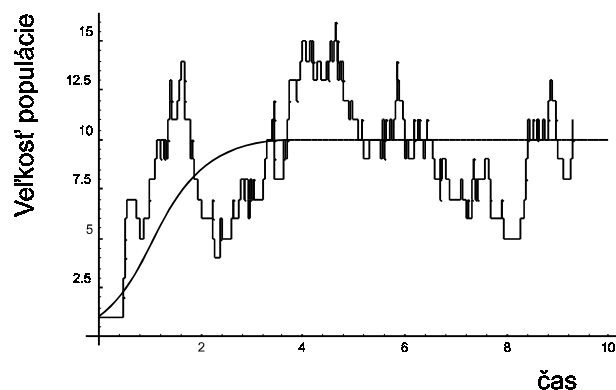
$$s = \frac{-\ln Y_2}{B(N) + D(N)} \quad (9.8)$$

Na rozdiel od deterministického modelu má síce stochastický model rovnovážny stav, no po čase stochastickými zmenami populácia vymrie. U deterministického modelu sa berie veľkosť populácie ako reálne číslo, ktoré pri daných parametroch rovnice nikdy nemôže klesnúť na nulu. U stochastického modelu je veľkosť populácie celé číslo, ktoré keď sa náhodou dostane na nulu, už sa z nej nikdy nedostane. Stochastický model je opísaný algoritmom 9.1, ktorého výsledok je na obr. 9.7.

```
function B(n:integer):real;
begin B:=n*(a1-b1*n); end;
function D(n:integer):real;
begin D:=n*(a2+b2*n); end;

a1=2.2; a2=0.2; b1:=0.1; b2:=0.1; n:=1; time:=0;
N(time):=1;
draw(N(time));
for step:=0 to step<200 do
begin if n=0 break;
  Y1=random; Y2=random;
  s=-ln(Y2)/(B(n)+D(n));
  time:=time+s;
  N(time):=n;
  draw(N(time));
  if 0≤Y1≤B(n)/(B(n)+D(n)) then n:=n+1;
  else n:=n-1;
  N(time):=n;
  draw(N(time));
end;
```

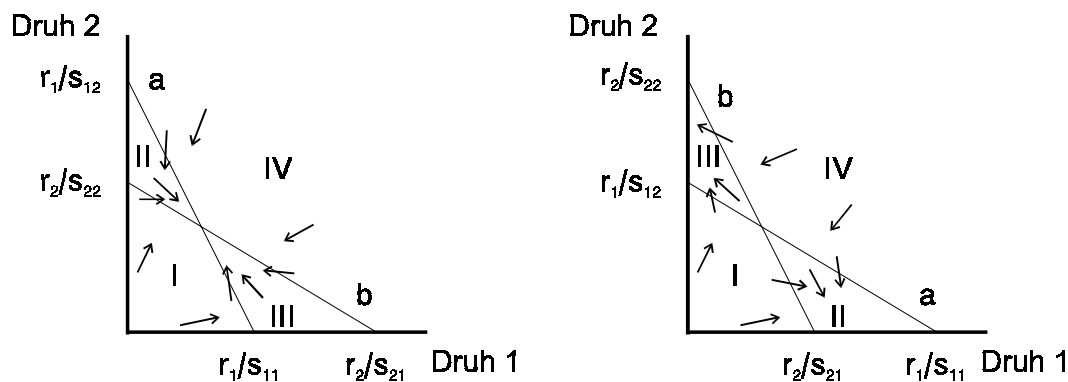
Algoritmus 9.1. Generovanie veľkosti populácie v závislosti od času pomocou stochastických zmien. Výsledok algoritmu je znázornený na obr. 9.7. Príkaz `draw(N(time))` vykreslí spojnicu súradníc posledného bodu s novým bodom so súradnicami $(time, N(time))$.



Obrázok 9.7. Zobrazenie stochastickej realizácie zmien veľkosti populácie v závislosti od času podľa rovnice 9.5 a algoritmu 9.1 (stupňovitá krivka) a súčasne deterministickej krivky podľa riešenia diferenciálnej rovnice $N'(t) = N(t) ((a_1 - a_2) - (b_1 + b_2) N(t))$ s okrajovou podmienkou $N(0) = 1$ (hladká krivka).

Ako ďalší príklad simulácie môžeme uviesť súťaženie dvoch druhov, kde nárast populácie jedného druhu je súčasne inhibovaný tak veľkosťou populácie tohto druhu, ako aj veľkosťou populácie druhého druhu. Na rozdiel od Volterrových-Lotkových rovníc by teda u každého druhu samostatne nastal nárast až do stabilného bodu, zatiaľ čo u Volterrových-Lotkových rovníc by sa koristi bez dravcov rozmnožovala donekonečna, zatiaľ čo dravci bez koristi by zahynuli. Zároveň nárast koristi spôsobuje nárast dravcov, zatiaľ čo pri súťaživom vzťahu je to naopak, druhy sú si navzájom dravcami, aj keď sa nepožierajú, stačí keď spotrebovávajú rovnaký druh obmedzených zdrojov. Keď $N_1(t)$ a $N_2(t)$ označíme množstvá prvého a druhého druhu v čase t , môžeme písať

$$\begin{aligned} dN_1/dt &= N_1(r_1 - s_{11}N_1 - s_{12}N_2) \\ dN_2/dt &= N_2(r_2 - s_{21}N_1 - s_{22}N_2) \end{aligned} \quad (9.9)$$



Obrázok 9.8. Diagramy stability pre súťaženie dvoch živočíšnych alebo rastlinných druhov. V prvom prípade dostávame stabilnú rovnováhu v mieste prekríženia dvoch priamok a a b , v druhom prípade dostávame nestabilnú rovnováhu v mieste prekríženia - pri akomkoľvek vychýlení jeden z druhov vymrie, záleží iba od smeru vychýlenia. V prípade, že by sa priamky a , b nepretínali a priamka a je nad b , vymrie druh 2, v opačnom prípade vymrie druh 1.

Môžeme rozlíšiť štyri prípady:

1. Keď $r_1/r_2 > s_{12}/s_{22}$ aj s_{11}/s_{21} , potom druh 1 vyhráva.
2. Keď $r_1/r_2 < s_{12}/s_{22}$ aj s_{11}/s_{21} , potom druh 2 vyhráva.
3. Keď $s_{12}/s_{22} < r_1/r_2 < s_{11}/s_{21}$, potom existuje stabilné rovnovážne riešenie s konštantnými veľkosťami populácií N_1 a N_2 pre prvý a druhý druh (viď prvý diagram z obr. 9.8).
4. Keď $s_{11}/s_{21} < r_1/r_2 < s_{12}/s_{22}$, potom existuje nestabilné rovnovážne riešenie. Keď sa veľkosti populácie vychýlia z tohto riešenia, začnú sa vychyľovať stále viac, až pokiaľ jeden z druhov nevyhynie. Ktorý z druhov vyhynie, to závisí na počiatočnom stave a pri stochastickej simulácii čiastočne aj na náhode (viď druhý diagram z obr. 9.8).

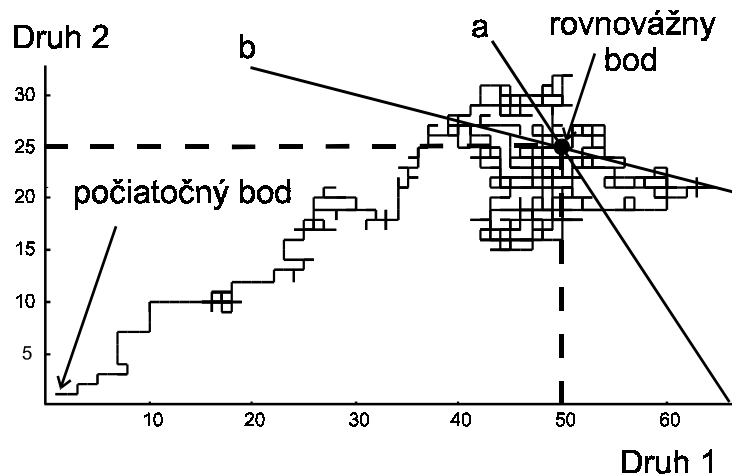
```
function B1(n:integer):real;
begin B1:=n*r1 end;
function D1(n1,n2:integer):real;
begin D1:= n1*(s11*n1 + s12*n2) end;
function B2(n:integer):real;
begin B2:=n*r2 end;
function D2(n1,n2:integer):real;
begin D2:= n2*(s21*n1 + s22*n2) end;

r1=2.0; s11=0.03; s12=0.02; r2=1.5; s21=0.01; s22=0.04;
n1:=1; n2:=1;
draw(1,1);
for step:=0 to step<1000 do
begin
Y1:=random;
R:=B1(n1)+D1(n1,n2)+B2(n2)+D2(n1,n2);
if 0≤Y1≤B1(n1)/R then n1:=n1+1 else
if Y1≤(B1(n1)+D1(n1,n2))/R then n1:=n1-1 else
if Y1≤(B1(n1)+D1(n1,n2)+B2(n2))/R then n2:=n2+1 else n2:=n2-1;
draw(n1,n2);
end;
```

Algoritmus 9.2. Generovanie stochastických zmien veľkostí dvoch súťažiacich populácií. Vzhľadom na počiatočné hodnoty konštánt r a s má systém stabilné riešenie $n_1=50$, $n_2=25$, okolo ktorého sa budú pohybovať veľkosti populácií. Výsledok algoritmu je znázornený na obr. 9.9. Príkaz `draw(n1,n2)` vykreslí spojnicu súradníc posledného bodu s novým bodom so súradnicami (n_1, n_2) .

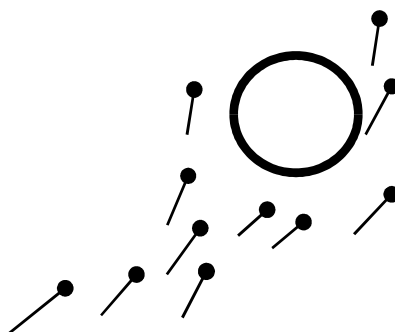
Načo modelovať vývoj pomocou stochastických procesov, keď pri vyššie uvedených príkladoch existuje aj jednoznačné analytické riešenie zodpovedajúcich diferenciálnych rovníc? Stačí jednoducho pridať priestorové vzťahy, t.j. umiestniť jednotlivé "živočích" na toroidálnu mriežku ako u celulárnych automatov (viď ďalej) a diferenciálne rovnice prestávajú platiť. Jednoduchým pridaním priestorovej interakcie sa problémy stávajú vo väčšine prípadov analyticky neriešiteľnými.

Úloha 9.2. Namodelujte priebehy množstva dravcov a koristi ukázané v predchádzajúcich obrázkoch 9.6, nie však pomocou diferenciálnych rovníc, ale vytvorením stochastického modelu, kde množstvá dravcov a koristi a ich prírastky/úbytky budú dané celými číslami; v prípade, že zmena nie je vyjadrená celým číslom, berte neceločíselnú časť ako pravdepodobnosť a rozhodnite o zaokrúhlení nahor/nadol na základe porovnania tejto pravdepodobnosti s náhodným číslom z intervalu $(0,1)$.



Obrázok 9.9. Modelovanie prvých 1000 zmien veľkostí populácií pri súťažení dvoch druhov pomocou algoritmu 9.2. Populácie prvého aj druhého druhu začínajú s veľkosťami $N_1=N_2=1$. Kľukatou čiarou sú zobrazené postupné zmeny veľkostí populácií, ktoré sa po približnom dosiahnutí rovnovážneho stavu "potulujú okolo". Žiaľ, nie je tu zobrazený časový priebeh a jednotlivé zmeny z rozdielnych časových odsekov sa prekrývajú, čím vytvárajú nepravidelnú mriežku. Obrázok odpovedá prvému diagramu stability z predchádzajúceho obr. 9.8 pre konštanty $r_1=2.0$; $s_{11}=0.03$; $s_{12}=0.02$; $r_2=1.5$; $s_{21}=0.01$; $s_{22}=0.04$.

Typickým cieľom mnohých simulačných programov je *modelovanie kolónie mravcov* [57]. Je veľa programov, ktoré takéto správanie napodobňujú. Mravce totiž pri relatívnej jednoduchosti svojich jednotlivcov vykazujú komplikované skupinové správanie. Sú tak schopné vykonávať úlohy, ktoré idú ďaleko nad rámec možností jednotlivého mravca. Robia tak bez vzájomného spojenia "telefónnou linkou", bez centrálného riadenia a pri častých poruchách informácie alebo zmenách prostredia. Mravčie kolónie stavajú cesty medzi hniezdom a zdrojom potravy, alebo vytvárajú živé mosty pri migrácii. Z opačného pohľadu sa dá takéto správanie využiť aj pre optimalizačné výpočty [58].



Obrázok 9.10. Zobrazenie "žubrienok" (čiernych bodov s "chvostíkom"), ktoré sa pohybujú smerom vpravo nahor a vyhýbajú sa prekážke (kruhu). Model je inšpirovaný správaním sa vtáčieho krdla podľa Craiga Reynoldsa [59].

Ďalším príkladom emergentných vlastností je správanie sa vtáčieho krdla bez centrálného riadenia skúmané v prácach Craiga Reynoldsa [59]. Ten vytvoril základný model "vtáka",

nazvaného "boid". Do počítačového modelu sa potom vložila populácia "vtákov" riadiacich sa tromi základnými vlastnosťami:

- vyhýbať sa kolízii s okolitými "vtákmi",
- prispôbiť rýchlosť okolitým "vtákom",
- snažiť sa držať sa blízko okolitých "vtákov",
- (môže sa pridať základný smer, ktorým vtáky letia, nutnosť vyhýbať sa prekážkam, pridať malé náhodné zmeny smeru a rýchlosti vtákov apod.).

Každý "vták" pritom vidí iba svojich najbližších susedov. Tieto základné vlastnosti sú dostačujúce na vytvorenie prirodzeného správania sa krdľa. Keď sa napr. objaví prekážka, krdel' sa bez akéhokoľvek centrálného riadenia rozdelí na dve časti, ktoré sa po obľetení prekážky spoja. Tieto princípy sa napr. použili na realistickú animáciu netopierov vo filmoch Batman sa vracia a Cliffhanger.

Úloha 9.3. Vytvorte program na animáciu pohybu žubrienok na obrázku 9.10, kde veľký kruh je prekážka, "hlavička žubrienok" predstavuje umiestnenie žubrienky v dvojrozmernom priestore a dĺžka a smer chvostíka určujú rýchlosť a smer pohybu žubrienky. Pridajte "dravca", teda inofarebnú žubrienku, ktorá sa pohybuje smerom k ostatným žubrienkam a ktorej sa ostatné žubrienky snažia vyhýbať.

9.6 Celulárne automaty

Celulárne automaty [60,61] môžu tvoriť samostatnú kapitolu umelého života. Sú to diskkrétne dynamické systémy zložené z rovnakého typu "buniek", ktorých správanie je úplne určené ich momentálnym vlastným stavom a stavmi "buniek" v ich najbližšom okolí. Čo je považované za okolie, závisí od definície. Čas, priestor a stav systému sú diskkrétne. Bunky nemajú pamäť a sú rozložené na pravidelnej mriežke (jedno- či viacrozmernej, pri dvojrozmernej môže ísť o štvorcovú, trojuholníkovú alebo šesťuholníkovú mriežku). Mriežka mení svoj vzhľad po etapách. Môžeme si predstaviť, že stavy všetkých buniek sa v priebehu jedného časového kroku menia súčasne. Stav bunky v mriežke sa mení z "generácie na generáciu" pomocou pravidiel berúcich do úvahy blízke okolie bunky. Bunky majú konečný počet stavov. Stav každej bunky je charakterizovaný zopár bitmi informácie; čas sa posúva v diskkrétnych krokoch, pričom nový stav bunky je definovaný napr. podľa "tabuľky" obsahujúcej všetky možné stavy "centrálnej" bunky a jej susedov v predchádzajúcom časovom kroku. Každému prípadu v tabuľke zodpovedá nový stav "centrálnej" bunky. Všetky bunky používajú rovnakú prechodovú funkciu.

Zákony zmien sú teda lokálne a uniformné. Na základe veľmi jednoduchých zákonov však vznikajú veľmi zložité štruktúry a vzory správania vyšších celkov (zložitých vzorov) jednoduchých buniek. Ako jeden z prvých sa takýmito "hrami" zaoberal matematik Stanislaw M. Ulam v Los Alamos.

Evolúcia je základným kameňom výskumu v umelom živote. Jeden z najväčších otvorených problémov dneška je vznik života. Ako vznikli sebareplikujúce organizmy, nutné na evolúciu? Ulam navrhol John von Neumannovi, aby skonštruoval na "nekonečnej šachovnici" systém na štúdium strojovej reprodukcie operujúci iba so základnými strojovými príkazmi. Každá štvorcová bunka mohla byť v akomkoľvek stave zodpovedajúcom častiam "stroja". Stroj bol potom vzorom zloženým z takýchto častí. Pravidlá správania sa častí stroja boli vytvorené na základe zjednodušenej fyziky. Takýto celulárny automat "zostrojil" von Neumann koncom štyridsiatych rokov [62]. Jeho reprodukčný automat pozostával z obdĺžnikovo usporiadaných buniek nasledovaných dlhým "chvostom". Vnútri obdĺžnika existovalo množstvo "podorganizmov". Spracovateľské podorganizmy spracovávali materiál z okolitého prostredia, podorganizmy zabezpečujúce duplikáciu čítali súbor inštrukcií a

kopírovali ich. Každá bunka mala 29 možných stavov; existoval súbor pravidiel prechodov z jedného stavu do druhého. Jeden z hlavných von Neumannových prínosov bol, že reprodukčný proces používal inštrukcie na spracovanie dvoma rôznymi spôsobmi: ako interpretovaný kód v priebehu vlastnej stavby a ako kopírované dáta - kopírovanie príkazov na stavbu do potomka. Neskôr sa dokázala existencia tohto rozdelenia aj v prírode: Inštrukcie na spracovanie (DNA) sa používajú na vytvorenie vlastného "systému" - proteínov. V celulórných automatoch sa môžu objekty chápať aj ako pasívne dáta, ktoré sa podrobujú operáciám, aj ako výpočtové prostriedky. To využil Langton [63] vo svojej výrazne jednoduchšej verzii samoreprodukujúcich sa tzv. Q-slučiek v rámci dvojdimenzionálneho celulórneho automatu.

Na tom istom mieste - University of Michigan - začal potom používať John Holland celulórne automaty na problémy adaptácie a optimalizácie.

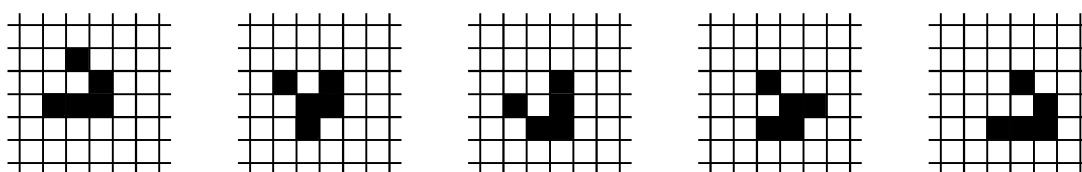
Najpopulárnejším celulórnym automatom je hra "Život" (Life), navrhnutá r. 1970 Johnom Hortonom Conwayom, mladým matematikom z Gonville and Caius College v Cambridgi, a rozšírená "rubrikami" Martina Gardnera v časopise Scientific American [64,65]. Von Neumannov automat bol síce univerzálnym počítačom (mohol simulovať akúkoľvek opísateľnú funkciu akéhokoľvek počítača pomocou súboru logických pravidiel), no bol zbytočne zložitý. Conway zjednodušil počet možných stavov bunky z von Neumannových 29 iba na 2: bunka mohla byť alebo živá, alebo mŕtva.

Pravidlá: Hráč tu vlastne nerobí nič, iba pozoruje štruktúry, ktoré vytvára počítač.

Conway vyskúšal veľa pravidiel pre vznik a prežitie bunky. Chcel pritom dosiahnuť, aby veľmi jednoduché vzory automaticky nerástli donekonečna, no niektoré s "divokým" a nepravidelným správaním by mohli rásť do nekonečna a mali by existovať aj stabilné štruktúry. Conwayove bunky sú rozmiestnené na štvorcovej mriežke a za susedov štvorčeka (bunky) považujeme štvorčeky nielen priamo susediace hranou, teda horný, dolný, vľavo a vpravo, no aj ďalšie štyri susediace "rohom".

Pravidlá:

1. STABILNÝ STAV: Keď má daná bunka dvoch živých susedov (nech už je sama živá či "mŕtva"), ostáva v rovnakom stave ako predtým aj v ďalšej generácii.
2. RAST: Keď má bunka presne troch živých susedov, bude v ďalšej generácii živá bez ohľadu na jej momentálny stav.
3. SMRŤ: Keď má bunka počet susedov 0, 1, 4-8, bude v nasledujúcej generácii "mŕtva". Bunka teda "zomrie na podchladenie alebo na prehriatie, keď má prí málo alebo priveľa susedov".



Čas 1

Čas 2

Čas 3

Čas 4

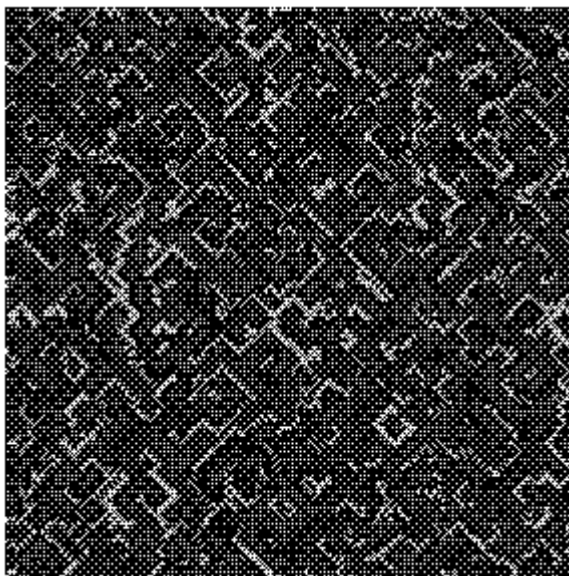
Čas 5

Obrázok 9.11. Štruktúra "klzák" z hry "Life" (primitívneho celulórneho automatu). Táto štruktúra sa vždy po štyroch krokoch opakuje, pričom sa pohybuje doprava nadol. Existuje veľa takýchto objektov, ktoré vzájomnými zrážkami môžu zanikať alebo tvoriť iné druhy štruktúr. "Klzák" si možno predstaviť ako bit 1 (jeho neprítomnosť ako bit 0) a smer jeho pohybu ako drôt. Z takýchto a podobných štruktúr potom možno zostaviť celý virtuálny počítač.

Populácia buniek sa mení z generácie na generáciu a je možné tvoriť najrôznejšie štruktúry. Conway sa zaujímal o možnosť rastu donekonečna. Ako jednu z hypotetických

možností navrhol tzv. "klzákovú pušku" (glider gun) - stabilný objekt, ktorý by "vystreľoval" vzory nazývané "klzák" (glider, viď obr. 9.11). R.W. Gosper na základe výzvy v populárnom časopise Scientific American vytvoril na počítači takúto štruktúru a tiež veľa ďalších.

Pri použití "klzákov" ako reprezentácie bitov bol Conway schopný vytvoriť ekvivalenty "and"-brán, "or"-brán a "not"-brán, sčítač, ako aj analóg vnútornej pamäte počítača. Tak sa preukázalo, že hra Life je vlastne Turingovým počítačom [66,67], rovnako ako ním môžu byť aj iné celulárne automaty [68].



Obrázok 9.12. Modelovanie vzorov celulárnym automatom.

Príklad iného typu správania sa je na obrázku 9.12, kde z náhodného rozloženia buniek na toroide (zobrazenom ako štvorec) pri štarte vznikajú pozdĺžne, kruhové alebo špirálovito sa rozširujúce vzory. Takéto "vlny" potom zostávajú na mieste alebo cestujú, rozširujú sa alebo zanikajú. Aj keď pravidlá sú deterministické, výsledkom je stav pripomínajúci chaos. Tento obrázok vznikol z náhodného stavu, kde každá z buniek bola v jednom zo 7 stavov (0-6). Za susedné bunky tu považujeme bunky susediace hranou. Pravidlá v poradí ich uplatňovania: Bunka susediaca s bunkou v stave 0 prechádza na stav 0. Bunka v stave c prechádza na stav $c+1$, pokiaľ je niektorý z jej susedov v stave $c+1$. Inak bunka zostáva v pôvodnom stave. Podobné systémy sa používajú na modelovanie fibrilácií srdca alebo oscilačných chemických reakcií.

Úloha 9.4. Zobrazte súbor stavov jednorozmerného celulárneho automatu, kde sú bunky iba v jednom riadku (formálne spojenom do cyklu, t.j. začiatok riadku susedí s koncom) a možné stavy buniek 0 = biela, a 1 = čierna. Prechodové pravidlá pre stav buniek berú do úvahy iba momentálny stav bunky (v prostriedku) a jej ľavého a pravého suseda. Pravidlá sú zadané takto: $1,1,1 \Rightarrow 0$; $1,1,0 \Rightarrow 0$; $1,0,1 \Rightarrow 1$; $1,0,0 \Rightarrow 0$; $0,1,1 \Rightarrow 1$; $0,1,0 \Rightarrow 1$; $0,0,1 \Rightarrow 0$; $0,0,0 \Rightarrow 1$. Napr. pravidlo $1,1,0 \Rightarrow 0$ znamená, že čierna bunka s čiernym ľavým a bielym pravým susedom bude v ďalšom časovom kroku biela. Zobrazte súbor riadkov v časovej postupnosti, t.j. riadok v čase 0, pod ním riadok v čase 1 atď. Počiatočný stav riadku v čase 0 má byť náhodne generovaný.

Úloha 9.5. Zostrojte model celulárneho automatu so 4 stavmi:

1. Prázdna bunka 2. Mravec
3. Ihličie 4. Mravec s ihličím

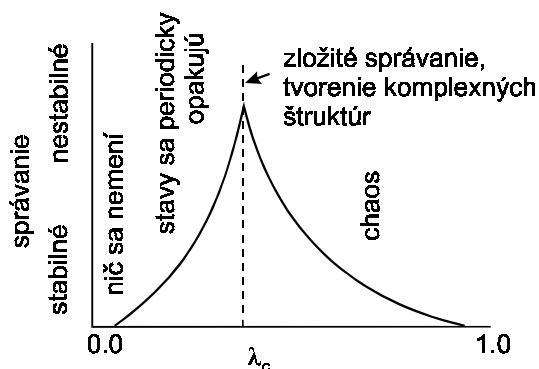
Pravidlá: Na počiatku obsahuje dvojrozmerná mriežka (na okrajoch spojená do toroidu, aby neboli problémy s ohraničením) náhodne rozmiestené ihličie a mravce. Mravec sa pohybuje vždy náhodným smerom, no tak, aby dva mravce neboli v jednej bunke. Keď narazí na ihličie, zoberie ho. (Môže tak aj z "kôpky" odoberať.) Keď mravec s ihličím narazí na iné ihličie, odloží svoje ihličie a nezdvihne hneď susedné. Pozorujte, ako sa takýmto distribuovaným procesom budú vytvárať "kôpky".

Uvedené celulárne automaty vyzerajú ako hračky, no sú spojené s hlbokými filozofickými otázkami, napr. ako je možné, že zložité systémy, ako sú živé organizmy, môžu vzniknúť vo svete, kde základným zákonom je zvyšovanie entropie, teda "dezorganizácie". Dynamika zmien riadených informáciou v živých systémoch prevzala kontrolu nad dynamikou zmien určených tokom energie.

9.7 Vznik komplexných javov

Celulárne systémy sú priestorovo rozložené nelineárne dynamické systémy schopné vykazovať stabilitu, cyklické správanie aj chaos. Zároveň sú schopné modelovať univerzálny počítač. Pritom sú jednoznačne definované pomocou "look up table". Zároveň do značnej miery môžu predstavovať fyzikálny model. Kedy u takýchto systémov začne prevažovať dynamika informácie vykazujúca komplexné správanie sa?

Christopher Langton považoval niektoré z celulárnych automatov za "nudné", pretože po niekoľkých generáciách všetky bunky vymreli, alebo sa usporiadali do jednoduchých pravidelných štruktúr - boli vysoko usporiadané. Iné automaty zase boli príliš chaotické a nepredpovedateľné, nedali sa rozpoznať od náhodne vygenerovaného šumu. Niektoré celulárne automaty však vykazovali zaujímavé komplikované správanie sa na hranici medzi poriadkom a úplným chaosom. Langton definoval parameter nazvaný lambda, pomocou ktorého sa dá predpovedať, či daný celulárny automat bude usporiadaný, chaotický alebo na hranici medzi poriadkom a chaosom [69].

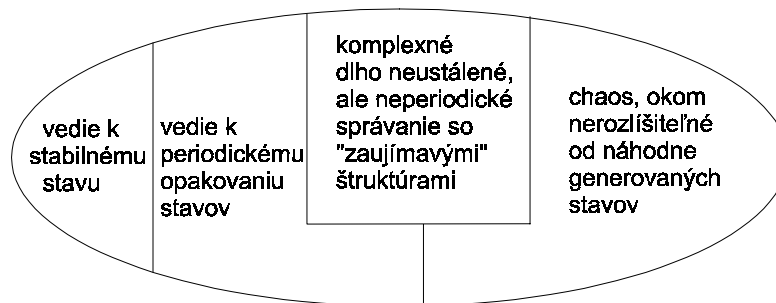


Obrázok 9.13. Schematické znázornenie závislosti komplexnosti od parametra λ v priestore celulárnych automatov, ukazujúce vzájomný vzťah medzi Wolframovými triedami [70] (idúcimi od nemenného stavu cez periodicitu a komplexné správanie po chaos) a fázovým priechodom. Treba poznamenať, že "meradlo stability" celulárneho automatu pred a za fázovým priechodom označeným λ_c sa líši spôsobom merania.

Nech je ďalšie správanie bunky určené funkciou so vstupom stavov buniek v okolí s počtom buniek N (včítane bunky samotnej) a výstupom funkcie je nový stav bunky. Nech celkovo existuje K možných stavov bunky. Vyberme si jeden stav bunky, ktorý nazvime "neaktívny". Nech počet pravidiel funkcie, ktoré vedú k tomuto neaktívnemu stavu, je n . Nech ostatných $K^N - n$ prechodových pravidiel funkcie je vybraných náhodne a uniformne zo zvyšných $K - 1$ stavov bunky. Potom je parameter lambda

$$\lambda = \frac{K^N - n}{K^N} \quad (9.10)$$

Keď je $n = K^N$, lambda je 0, všetky bunky okamžite vyhynú. Pre $n=0$ je lambda rovné 1 a žiadna bunka nehynie. Keď sú prechody na všetky stavy v pravidlách funkcie rozložené rovnomerne, potom $\lambda=1.0-1/K$. Medzi lambda rovnou nule (usporiadaným "mŕtvym" stavom) a rovnomerne rozloženými stavmi v pravidlách (chaosom) leží oblasť zaujímavého správania - na hranici chaosu.



Obrázok 9.14. Schematické znázornenie priestoru pravidiel celulárnych automatov (podľa Langtona [69]).

Parameter lambda dobre opisuje zmeny dynamických režimov iba pre väčšie K a N , pre úlohu 9.4 sú tieto hodnoty príliš malé, aby sa dosiahol pravidelný prechod ukázaný v grafe 9.13. Okrem toho je na obrázku 9.13 na osi y "čas na dosiahnutie stabilného stavu", ale stabilný stav je rôzne definovaný pre hodnoty pred a za fázovým prechodom. Pred fázovým prechodom je dosiahnutie stabilného stavu definované tak, že všetky bunky sú v stave, ako boli už v niektorom z predchádzajúcich časových odsekov, a teda sa začína situácia periodicky opakovať (s väčšou či menšou periódou). Naproti tomu pre lambda po fázovom prechode je stabilný stav štatisticky vyjadrený pomocou histórie tak, že priemerná "hustota" obsadenia buniek sa usadila tak, že sa nelíši viac ako o 1 percento od svojho dlhodobého priemeru. Treba poznamenať, že parameter lambda nie je dokonalou charakteristikou pravdepodobnosti vzniku komplexného správania. Začnime z náhodného stavu buniek a so všetkými pravidlami vedúcimi k "mŕtvemu" stavu; postupne náhodne po jednom modifikujeme pravidlá tak, že budú viesť k "živému" stavu namiesto mŕtveho, a vždy znovu spustíme automat a zisťujeme jeho správanie. Uvidíme, že správanie niekedy "preskočí" z periodického na chaotické správanie bez fázy tvorby komplexných štruktúr, a inokedy sa môže objaviť komplexné správanie aj mimo "fázového prechodu" (schematicky na obr. 9.14).

Treba zdôrazniť, že oblasť komplexného správania je v skutočnosti veľmi malá v porovnaní s množstvom chaoticky sa správajúcich automatov, alebo s periodicky sa opakujúcimi stavmi. (Treba tiež zdôrazniť, že vzhľadom na konečnosť pravidiel a automatu sa každý automat správa periodicky, otázka je iba, či sa rovnaký stav opakuje hneď v ďalšej iterácii, za desať iterácií alebo za milión iterácií. Zaujímavé štruktúry sú také, kde aj keď nepoznáme pravidlá a poznáme iba momentálny stav, sme schopní odhadnúť budúci krok s nenáhodnou, no nie príliš veľkou pravdepodobnosťou.)

Je otázkou, či sa celulárne automaty nemôžu použiť okrem modelovania výsledného správania nášho sveta aj na modelovanie základných zákonov fyziky. Špeciálne upravené celulárne automaty môžu zachovávať aj najzákladnejšie rysy fyziky, ako je vratnosť. Celulárne automaty sa použili ako extrémne jednoduché modely obyčajných diferenciálnych rovníc, ako tepelné a vlnové rovnice [71] a Navierove a Stokesove rovnice [72,73].

Celulárne automaty môžu tiež slúžiť ako diskkrétne modely na štúdium dynamických systémov vykazujúcich emergentné javy spôsobené "kolektívnymi" vzťahmi, ako je usporiadávanie, turbulencia, chaos, fraktály [74,75], alebo modelovanie biologických fenoménov, ako sú hmyzie spoločenstvá, bunkové kolónie, sieťnica (vnútorný obal očnej gule) alebo imunitný systém [76].

Evolúcia celulárnych automatov môže slúžiť aj na návrh hardvérových riešení [77-80]. Tzv. *evolvér* (evolware) je systém založený na celulárnom programovaní, v ktorom sa paralelné celulárne automaty (s implementovaným genetickým algoritmom, kde nie všetky pravidlá musia byť rovnaké pre všetky bunky) používajú na riešenie výpočtových problémov. Takéto systémy sú už pokusne implementované aj hardvérovo. Výhodou takých systémov je okrem iného aj väčšia tolerancia k vstupným chybám aj k chybám pri výpočte, ktoré pri stovkách tisícok výpočtových prvkov ("mikroprocesorov") nemusia byť až také zanedbateľné.

Problémom spojeným so "sieťovými" systémami je, že pri určitej úrovni zložitosti pravdepodobne nie je žiaden matematický model dopredu schopný presne predpovedať správanie systému, jeho vynárajúce sa (emergentné) vlastnosti (ako je vytvorenie "kôpok"). Zacyklené kľbká spojení sú natoľko zamotané a formálne modely, ktoré ich opisujú, natoľko nešikovné, že jediným možným spôsobom ako zistiť správanie modelu je "spustiť" ho. Zdá sa, že sa systém väčšinou "bráni" riešeniu pomocou spriahnutých nelineárnych rovníc. Z fyzikálno-chemického charakteru molekuly vody je ťažké odhadnúť vznik vírov, zo zrnka piesku existenciu lavín a z jednej molekuly užitočnosť pojmu teplota (skupinovej vlastnosti molekúl), zo semena nemožno odhadnúť konečnú podobu rastliny. Najrýchlejším spôsobom ako zistiť "výstup" semena je nechať ho vyklíčiť. Podobné problémy spojené s "motýlím" efektom, kde malá zmena vo vstupe môže spôsobiť veľkú zmenu výstupu (z vulgarizovane, "zamávanie krídel motýľa môže spôsobiť uragán na inom kontinente"), sa vyskytujú všade v "sieťovej matematike". Do tohto pojmu môžeme zahrnúť paralelné distribuované procesy, boolovské siete, neurónové siete, spinové sklá, celulárne automaty, klasifikačné systémy, genetické algoritmy a "rojové" (swarm) výpočty. Všetky obsahujú vzájomný vplyv tisícok simultánne interagujúcich funkcií, kde ide o masívne paralelné systémy často s lokálnym prepojením a relatívne primitívnymi základnými prvkami. V aplikáciách sa systémy tohto typu využívajú napr. vo fyzike na simuláciu kritických stavov, turbulencie alebo toku kvapaliny cez pórovité médiá, difúzie energie v plynach, v biológii na modelovanie biologických rytmov, rastu nádorov, epidémií, ekologických rovnováh, premiestňovania živočíchov, v chémii pri polymerizáciách, oscilačných chemických reakciách, agregáciách koloidov alebo vzniku kryštálov, difúzných reakciách, v geológii na modelovanie zemetrasenia, sopiek a pretvárania zemského povrchu, v ekonomike pri fluktuáciách cien, v sociológii na modelovanie šírenia správ, imitačného správania, rastu populácie, vývoja mesta alebo dopravných zápch.

V robotike sa pokladajú za budúcnosť mikroroboty - rýchle, lacné a neriadene. Využitie: planetárny výskum, baníctvo a žatva, výstavba "na diaľku".

V tradičnej *robotike* sa programátori pokúšajú predpovedať a riadiť každý aspekt akcie robota. Takéto systémy sú potom bezradné, keď je robot postavený pred nepredvídanú situáciu. Na rozdiel od klasického prístupu je adaptívna kontrola pohybu robota schopného sa učiť v premenlivom prostredí typickou praktickou aplikáciou "umelého života" [81-84].

Laboratórium Rodneyho Brooka pre výskum robotov stanovilo tieto zásady [85,86]:

1. Začať s jednoduchými úlohami.
2. Naučiť sa ich robiť bezchybne.
3. Pridávať ďalšiu úroveň riadenia nad výsledkami jednoduchých úloh.

4. Nemeniť jednoduché základné veci, vyššie úrovne môžu maximálne potlačiť výstup nižších vrstiev.
5. Naučiť novú vrstvu vykonávať veci tak dobre, ako to len bude možné.
6. Opakovať dookola.

Naproti tomuto sú zatiaľ priemyselné roboty direktívne riadené. Pokusy vytvoriť "inteligentného" robota pomocou klasickej umelej inteligencie zlyhávajú - "úspešné" roboty sú po dvadsaťročnom úsilí vo výskume nanajvýš schopné pohybovať sa po rovnej ploche s prekážkami v tvare kvádra. "Geneticky" riadené roboty zatiaľ predstavujú maximálne desaťcentimetrové hračky, ktoré sú vzhľadom na podstatne kratší čas ich vývoja schopné zvládať približne podobné "úlohy".

Zásadou je postupné budovanie zložitého systému, nie budovanie zložitého systému naraz. Príkladom je napríklad ľudský mozog; prvotné sú základné funkcie ako riadenie dýchania a potom možno pridať funkcie vyššej úrovne ako myslenie. Riadenie musí byť čo najviac decentralizované a komunikácia musí obsahovať iba najnutnejšie príkazy - podobne ako je to pri objektovo orientovanom programovaní.

Inteligentné riadenie vyzerá ako slobodné správanie. Jediný spôsob, ako je možné inteligentne riadiť mechanizmy, je dať im slobodu.

9.8 Kooperácia alebo súťaženie?

Evolúcia je adaptácia zameraná na naplnenie vlastných potrieb. Koevolúcia je vo väčšej miere adaptácia na naplnenie potrieb navzájom medzi druhmi. Koevolučná ekológia je zbierkou organizmov, ktoré fungujú ako svoje životné prostredie. Je proti záujmu ako dravej zveri, tak aj koristi, aby bol druhý druh vyhubený. Aj keď pre korisť na prvý pohľad vyzerá zánik nepriateľa priaznivo, dravá zver v skutočnosti väčšinou likviduje najslabšie kusy koristi a tým prispieva k zdokonaľovaniu genofondu a zabráňuje degenerácii. Ako vyplýva z teórie hier v ekonómii, aplikovanej tiež často na analýzu pretekov v zbrojení medzi štátmi, zjednodušenej zo skúmania "problému väzňov" (prisoner's dilemma, pozri [87,88]), kooperácia sa môže vynoriť zo sebeckého záujmu, keď je výhodná pre obidvoch. Väzenské dilemma vzniká, keď sa dvaja väzni dohodnú na úteku a niečo sa pokazí. Keď obidvaja spolupracujú - zapierajú, vyjdú z toho relatívne dobre - vyjadrené ohodnotením plus X bodov. Keď jeden z nich zradí druhého, zradca dostane plus Y bodov, prezradený nula bodov. A keď obidvaja zradia, dostanú každý W bodov. (V tabuľke na obr. 9.15 sú tieto hodnoty $X=3, Y=5$ a $W=1$.) Ide teda v podstate o štyri prípady kombinácie dvojíc K (kooperuje - spolupracuje so spoluväzňom) a Z (zrádza spoluväzňa, angl. defects), teda KK, KZ, ZK a ZZ , kde prvé písmeno vždy označuje, ako sa zachoval prvý spoluväzeň, a druhé písmeno označuje správanie druhého spoluväzňa.

Pri iba jednej iterácii by samozrejme bolo vždy výhodné zradiť, no keď sa iterácie s tým istým spoluväzňom opakujú a vopred nie je známy počet iterácií, metóda ako sa zachovať nie je jasná. Väčšinou sa výhodne uplatňuje "tit-for-tat" - (približne "ako ty mne, tak ja tebe") - prvýkrát spolupracovať a prestať spolupracovať s tým, kto nespupracuje. Metóda "tit-for-tat" sa dá vyjadriť na základe správania dvojice spoluväzňov pri predchádzajúcej príležitosti, čo teraz určuje správanie prvého spoluväzňa. Teda: $KK \Rightarrow K, KZ \Rightarrow Z, ZK \Rightarrow K, ZZ \Rightarrow Z$. Pokiaľ sú predchádzajúce prípady správania jednoznačne usporiadané, potom možno toto správanie vyjadriť reťazcom pravých strán $KZKZ$. Samozrejme, pred to musíme ešte pridať písmeno určujúce, aká je stratégia (K alebo Z) pri prvom stretnutí dvojice spoluväzňov. Keď zoberieme stratégiu pamätajúcu si správanie pri troch predchádzajúcich príležitostiach, máme 64 možností: $KK KK KK$ (prípád 1), $KK KK KZ$ (prípád 2), $KK KK ZK$ (prípád 3), ... , $ZZ ZZ ZK$ (prípád 63), $ZZ ZZ ZZ$ (prípád 64). Stratégia ako sa po prvých troch stretnutiach zachovať nabudúce by teda bola zakódovaná reťazcom 64 písmen $KZKKKZ...$. Pri plánovaní stratégie

na základe predchádzajúcich skúseností s protihráčom sa často používa genetický algoritmus. Táto aplikácia genetického algoritmu prvýkrát vznikla zo spolupráce politológa Roberta Axelroda s Johnom Hollandom [89] a rozvíjala sa ďalej pre zložitejšie modely medzinárodnej bezpečnosti odpovedajúce napr. na otázku, koľko z rozpočtu vydávať na obranu. Axelrod zbral zakódovanie stratégie reťazcom ako chromozóm genetického algoritmu, nechal každú stratégiu súťažiť s ostatnými stratégiami, a súčet výsledkov pre jednotlivú stratégiu vždy odpovedal jej ohodnoteniu. Na základe ohodnotenia sa reťazce vyberali a potom dochádzalo k ich kríženiu a mutácii analogickej binárnemu reťazcu (pretože máme vždy iba dva prípady K a Z, išlo vlastne o binárny reťazec). V časovom priebehu je potom zaujímavé sledovať vývoj kooperatívnych a nekooperatívnych stratégií. Kooperatívne stratégie sa spočiatku ukázali menej výhodné, no v neskorších štádiách prevládali.

	Väzeň č. 2 kooperuje (K)	Väzeň č. 2 zrádza (Z)
Väzeň č. 1 kooperuje (K)	odmeny väzňov: 3, 3 obidvaja sú "odmenení" za vzájomnú spoluprácu prípady K,K	odmeny väzňov: 0, 5 väzeň č. 1, ktorý sa snažil spolupracovať, nedostane nič, väzeň č. 2, ktorý zradil, bude dozorcami odmenený, prípady Z,K
Väzeň č. 1 zrádza (Z)	odmeny väzňov: 5, 0 väzeň č. 1, ktorý zradil, bude dozorcami odmenený, väzeň č. 2, ktorý sa snažil spolupracovať, nedostane nič, prípady K,Z	odmeny väzňov: 1, 1 obidvaja sa navzájom zradili a sú mierne potrestaní, prípady Z,Z

Obrázok 9.15. "Payoff matrix", teda výplatná tabuľka pre problém dvoch väzňov závislá od ich rozhodnutia.

Evolúcia nie je "hra s nulovým súčtom výhier", kedy výhoda jedného prináša nevýhodu druhému - vzájomná kooperácia môže priniesť výhodu obom druhom bez toho, že by nejakému tretiemu druhu škodila.

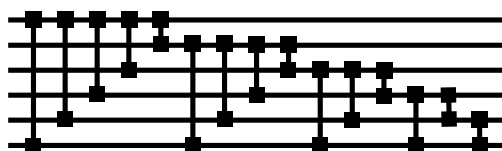
Výhoda kooperácie nemusí byť hneď vo všeobecnosti zrejmá - keď v zvieracích rodinách "ujovia a tety" nezaložia vlastnú rodinu, ale pomáhajú súrodencom s výchovou synovcov a neterí, pomáhajú tým podporiť rozšírenie svojho genofondu (lebo ich príbuzní majú časť genetickej informácie zhodnú) rovnako, ako keby sebecky založili vlastnú rodinu.

Takéto zdanlivo biologické koncepty ako koevolúcia sa môžu s výhodou použiť napr. pri evolúcii programov. Hillis [90] sa pokúšal vyvíjať triediace algoritmy pre zoznamy čísel, kde algoritmy boli ohodnocované podľa toho, koľko zoznamov zoradili správne. Takto vyvinuté algoritmy boli však ďaleko od optima. Keď sa však zároveň nechali vyvíjať aj zoznamy, ktoré boli ohodnotené tým lepšie, čím viac algoritmov ich nezoradilo správne, výsledné algoritmy boli oveľa efektívnejšie. O podobný prístup sa pokúsil aj Olsson [91] pri optimalizácii triediacich sietí.

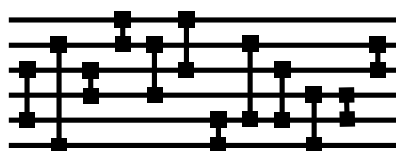
Triediaca sieť opisuje algoritmus na zoradovanie sady čísel podľa veľkosti od najmenšieho (hore) k najväčšiemu (dole). Každá vodorovná čiara v sieti reprezentuje umiestnenie čísla v zozname. Každá zvislá úsečka zakončená štvorčekmi predstavuje porovnanie dvoch čísel v daných umiestneniach a ich prípadnú výmenu, pokiaľ je v "hornej pozícii" väčšie číslo ako v "dolnej". V prípade, že je mechanizmus usporiadania korektný, mal by správne zoradiť všetky možné permutácie danej veľkosti (viď. obr. 9.16). Keď je sieť minimálna, neexistuje korektný algoritmus s menším počtom porovnávaní.

Testovanie všetkých možných triediacich sietí, ktoré dokážu korektne porovnať iba 16 čísel pomocou povedzme 64 porovnaní, by vyžadovalo vyskúšať 10^{154} sietí. Preto sa skúša

použitie evolučných algoritmov na vyhľadanie takýchto sietí. Ako je vidieť z obr. 9.16, triediaca sieť vyhľadaná pomocou evolúcie môže byť lepšia ako sieť získaná zápisom algoritmu, no v žiadnom prípade nepripomína pravidelnosť zvyčajne obsiahnutú v algoritme.



Bublinkové triedenie



Výsledok evolúcie

Obrázok 9.16. Korektné siete na triedenie sady čísel podľa veľkosti. Bublinkové triedenie (bubble sort) dokáže zoradiť 6 čísel s použitím 15 porovnaní, zatiaľ čo jednej z minimálnych korektných sietí napravo stačí iba 12 porovnaní. Takýchto minimálnych korektných sietí existuje viac (môžeme napríklad prehodiť prvé dve porovnania).

Život je charakterizovaný *neprestajnou zmenou* spôsobovanou autoorganizáciou. Časť zákonov riadiacich tieto zmeny a ich smerovanie bola síce v princípe vysvetlená Darwinovou evolučnou teóriou, avšak veľa vecí zostáva ešte nevyjasnených.

Život je charakterizovaný nerovnováhou, odklonom od tendencie smerovania systému k väčšej entropii (samozrejme, za zmenšenie entropie v živých organizmoch sa platí zvýšením entropie inde). Pseudostabilný nerovnovážny stav je charakteristický pre "živé" systémy, napr. v ekonómii, v prírodnom ekosystéme, v zložitej počítačovej simulácii, v imunitnom alebo evolučnom systéme. David Layzer v knihe *Cosmogenesis* [92] tvrdí, že centrálnou vlastnosťou života nie je reprodukcia nemennosti, ale *reprodukováaná nerovnováha*. Živé organizmy musia vyrovnávať tendenciu "investovania do budúcnosti" - t.j. množstvo mutácií a inovačných trendov s tendenciou využívať už vyskúšané a fungujúce. Keď sa takéto zmeny premietnu do kooperácie alebo vzťahu dravec/koristi s ostatnými druhmi, vzniká koevolúcia.

Chyby chránia koevolučný vzťah pred príliš tesným spojením. Takéto spojenie by mohlo pri zániku jedného druhu spôsobiť lavínovitý zánik celého spoločenstva. Udržiavaním určitého stupňa "univerzálnosti" (napr. schopnosti živiť sa viacerými druhmi koristi alebo byť opelený viacerými druhmi hmyzu) sa spoločenstvo stáva odolné voči zmenám životných podmienok (napr. podnebia), aj keď nie je ideálne "optimalizované" pre dané životné podmienky.

Výsledok mutácií sa môže prirovnávať k bielemu šumu, zatiaľ čo aplikácia vyskúšaných pravidiel vedie k opakovaniu vzorov pravidelnosťou podobných kryštalickej štruktúre. Zaujímavá oblasť typická pre "život" sa nachádza medzi týmito dvoma extrémami. Zo svojich experimentov s modelmi roje Chris Langton zo Santa Fe odvodil tzv. parameter lambda, ktorý mu pomáha určovať nastavenie systému tak, aby sa prejavovali "životu podobné" vlastnosti, t.j. nepravidelné opakovanie vzorov. Tento jav sa dá prirovnávať k fázovému prechodu (napr. z kvapaliny na paru alebo na ľad) v termodynamike. Zaujímavé je, že systém má automatickú tendenciu zdržať sa v takýchto oblastiach čo najdlhšie. Takýto systém má najlepšiu schopnosť sa vyvíjať, a pritom neskĺznuť do "anarchie", ktorou je v termodynamickom zmysle smrť.

Roku 1990 Kristian Lindgren [93], pracujúci v Inštitúte Nielsa Bohra v Kodani, rozšíril koevolučný experiment "váženského problému" na populáciu 1000 hráčov, zaviedol do hry náhodný šum a nechal evolúciu prebiehať až do 30000 generácií. V takejto mierke sa napríklad prejavili krivky typické z ekologických oscilácií dravca a koristi, kde množstvá jedincov v populácii oscilujú a oscilácie sú späté - maximum koristi je nasledované maximumom dravcov - kde je korisť predstavovaná kooperujúcimi väzňami a dravci zrádzajúcimi väzňami.

Okrem toho sa prejavili aj ďalšie efekty ako parazitizmus, symbióza a dlhodobá kooperácia medzi skupinami - "zvieracími druhmi". Táto práca zaujala tiež biológov, lebo sa tu prejavili obdobia dlhej stability, nasledované krátkymi obdobiami nestability, kde veľa "druhov" vyhynulo a vytvorili sa nové. Obdobný vývoj, čiastočne oponujúci zjednodušenej darwinovskej predstave stabilného postupného vývoja, je známy zo sledovania skamenelín.

9.9 Vznik života evolúciou neživého

Virtuálny svet Toma Raya z University of Delaware nazvaný *Tierra* [94] je prechodom od kódov počítačových vírusov k experimentálnemu štúdiu evolúcie umelých organizmov na úrovni genómu. Snaží sa poskytnúť prostredie, v ktorom môže darwinovská evolúcia prebiehať v počítači bez priameho riadenia alebo intervencie človeka. Ray vytvoril zjednodušený "počítač" v počítači s menším množstvom inštrukcií v umelom kóde napodobňujúcom strojový kód. V tomto kóde Ray potom napísal 80 bytový "vírus", ktorý bol schopný rozmnožovania. Takýto organizmus je lineárnym reťazcom inštrukcií a žije postupným vykonávaním postupnosti svojich príkazov. Avšak Ray urobil aj niečo navyše - pri kopírovaní vírusu pridal s určitou pravdepodobnosťou náhodné preklopenie niekoľkých bitov, čo odpovedá mutácii v prírode. Základný súbor strojových príkazov je vhodne vybraný tak, že pri akejkoľvek mutácii zostane organizmus vykonávateľným programom. Organizmy sa teda môžu vyvíjať pomocou genetických operátorov. Každému "vírusu" bol riadiacim operačným systémom pridelený určitý strojový čas na beh, a teda rozmnožovanie. Na druhú stranu Ray pridal vek, takže staršie vírusy boli operačným systémom odstránené s väčšou pravdepodobnosťou. V *Tierre* nie je presne definovaná účelová funkcia. Vírusy súťažia o "prírodné zdroje" svojho prostredia, teda o čas CPU a pamäť. *Tierra* sa tak stáva umelým prostredím, v ktorom sa organizmy riadia vlastnými zákonmi. Organizmy, ktorým sa nejakým spôsobom podarí zabráť väčšinu týchto zdrojov, majú vyššiu schopnosť prežitia. Keď sa pamäť zaplní organizmami, operačný systém "zabíja" najmenej schopné organizmy, teda napríklad tie, ktorých kód v nejakom kontexte generuje chybu. "Svet" bol do tej miery umelo vytvorený, že pokiaľ si vírus napr. vytvoril nekonečnú slučku, bol umelo zastavený. Čoskoro sa v programoch objavili lepšie "vírusy", ako bol schopný sám Ray zostrojiť, a dokonca sa objavil aj "parazitizmus" - vírus bol krátky a využíval časť kódu iného vírusu. Žiaľ, implementácia vyšších funkcií ako inteligencia, vnímanie a komunikácia je v organizmoch *Tierra* spojená s ťažkosťami. Získanie týchto vlastností prirodzeným vývojom *Tierra* by trvalo veky. Zatiaľ sa nepokročilo významne ani v programovej verzii viacbunkových organizmov. Roku 1996 Andrew Pargellis vytvoril podobný virtuálny svet s ešte obmedzenejším počtom inštrukcií, v ktorom sa vytvoril sebareplikujúci sa "vírus" sám, bez nutnosti vytvoriť prvý vírus "božským" zásahom človeka [95]. Podobný systém ako *Tierra* je *Avida* [6] vytvorená Christophom Adamim. *Avida* poskytuje lepšie riadiace prostriedky, jej prostredie je však tvorené dvojrozmernou mriežkou a interakcie sú lokálne.

Umelá evolúcia sa však neobmedzuje iba na nepraktické pokusy s počítačovými vírusmi. V súčasnej dobe sa v praxi využíva na hľadanie nových typov *liečiv vo farmaceutickom priemysle*. Aktívny mechanizmus, ktorý má droga odštartovať, väčšinou spočíva v spustení reakcie molekulou, ktorá bude zapadať do receptora a reagovať s jeho aktívnymi miestami. Nestačí iba mať v liečivu jednotlivé aktívne skupiny zodpovedajúce skupinám v akceptore - biomolekule, treba ich aj priestorovo rozmiestniť. Organických molekúl je veľmi veľa, no spôsobov, akým je možné jednotlivé molekuly pohybovať okolo "otáčavých" väzieb v priestore, je faktoriálovo viac. Dlhé reťazce aminokyselín sa dajú poskladať do takého kľbka, že z neho trčia tie správne aktívne skupiny na správnych miestach.

V minulosti sa hľadanie liečiva riešilo metódou pokusu a omylu. Teraz, v tzv. *kombinatorickej chémii* [96-98], sa vyskúšajú milióny variácií molekúl - úspešná je tá

molekula, ktorá sa zachytí aspoň na jednom mieste. Táto úspešná molekula sa odseparuje od ostatných molekúl a pripraví sa jej variácie - jedna z nich sa možno zachytí na dvoch aktívnych miestach. Tá sa vezme, pripraví sa jej variácie ... atď., dokiaľ nemáme molekulu, ktorá sa perfektne hodí na aktívne miesta receptoru.

Cesty evolúcie

Vzhľadom na to, že evolúcia je jedným zo základných predmetov skúmania umelého života a zároveň praktické aplikácie jej princípov umožňujú vytvárať optimalizačné algoritmy, je vhodné zaoberať sa evolúciou aj zo všeobecného hľadiska [99,100].

Základná dogma v biológii súvisiaca s evolúciou je, že informácia cestuje z génu do živého organizmu, no nikdy opačne - lamarkovský typ evolúcie [99], kedy zjednodušene povedané kováčove deti by mali silné svaly, lebo kováč si ich musel vypracovať - je pre biológov neprijateľný. Veľmi nepravdepodobné je tiež priame uplatnenie náhodnej mutácie - neexistuje gén, ktorý by "napumpoval" kováčove svaly - za jednu vlastnosť je zodpovedných viac génov, rovnako ako každý gén je zodpovedný za viacero vlastností. Preto je v prírode výhodnejšie mať adaptabilné "telo" (myslené aj tela rastlinné alebo prvokov), ako mať "telo" presne zoptimalizované na momentálne životné prostredie a čakať, že pri zmene životného prostredia sa objaví vhodná mutácia, ktorá "telo" hneď zoptimalizuje na nové podmienky.

No spôsobiť zmenu "životného prostredia" môže aj zmena správania - Baldwinov efekt [99] - napríklad keď si ľudia obľúbia hory a ostanú v nich viac generácií, budú sa viac rozmnožovať zvýhodnení jedinci s vyššou produkciou hemoglobínu. Keď zoberieme ľudskú kultúru ako sebariadiaci systém, ktorý "konzumuje" ľudské zdroje výmenou za zefektívnenie niektorých činností (napr. efektívny prenos informácií rečou namiesto posunkov, alebo uchovávanie informácií v knihách namiesto v pamäti), kultúra môže po dlhšom čase spôsobiť genetické zmeny (napr. vývin hlasiviek, zmenšenie zubov prechodom na tepelné spracovanie jedla), aj keď takáto priama závislosť nebola dosiaľ jednoznačne preukázaná.

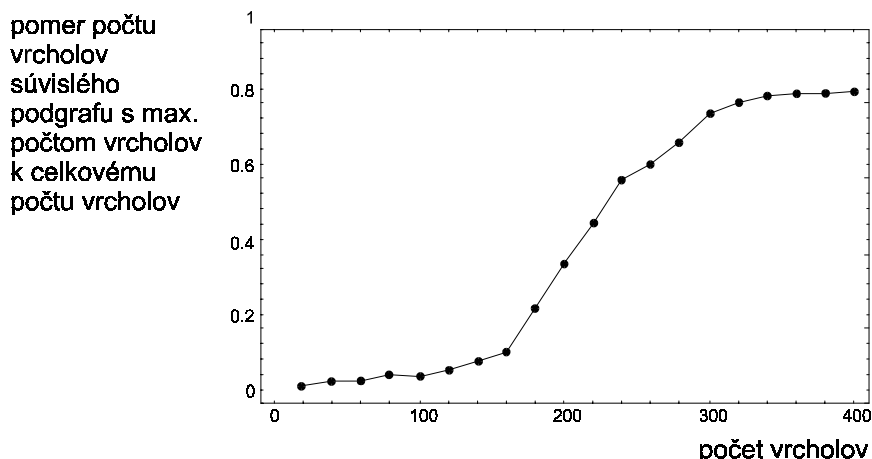
Evolúcia je štruktúra organizovanej zmeny, ktorá sama podlieha vývoju. Z časového hľadiska možno rozlíšiť nasledujúce typy evolúcie:

1. Sieť "do cyklu" prepojených chemických reakcií vytvárajúcich určitý druh stability (Stuart Kaufmann [101,102] verí, že komplikované systémy majú tendenciu k samoorganizácii a že organizovaná štruktúra môže vzniknúť spontánne).
2. Autoreplikácia.
3. "Učenie" genetickej informácie mutáciami, krížením a prežitím najschopnejších.
4. Zmeny spôsobené spoluúčasťou na tvorbe svojho "životného prostredia" výberom niky - miesta výskytu a vzťahmi tak vnútrodrohovými, ako aj mimodrohovými.
5. Evolúcia myšlienok ("memetic culture") v sieti ľudských mozgov - lamarkovská kultúrna evolúcia.
6. Evolúcia spôsobov a cieľov evolúcie - vyhľadávanie stratégií a spätnoväzbových mechanizmov, optimalizujúcich samých seba - napr. genetické inžinierstvo.

Moderným teóriám vzniku života predchádzal pokus chicagského študenta Stanleyho Millera [103], ktorý jednoducho zatvoril do 5-litrovej fľaše vodu, metán, amoniak a vodík, fľašu týždeň zahrieval a nechal vo vnútri generovať elektrické iskry simulujúce blesky. Výsledkom bola zmes komplikovaných zlúčenín aminokyselín potrebných na vznik života. Nemecký biofyzikálny chemik Manfred Eigen spolu s Petrom Schusterom potom vytvorili teóriu hypercyklov [104] vysvetľujúcu vznik života pomocou siete replikujúcich sa molekúl RNA. Problémom tu je, že RNA dosť ťažko môže vytvoriť svoju kópiu bez pomoci enzýmov, ktoré by mali byť produkované pomocou RNA - a RNA sama sa normálne vytvára prepisom z DNA.

Trocha pozmenenú teóriu vzniku života vytvoril jeden zo zriedkavých špičkových biológov, ktorí sa zaoberajú umelým životom, Stuart Kaufmann zo Santa Fe inštitútu, študujúci procesy samoorganizácie sietí spôsobené komplexným prepojením lokálnych reakcií [101,102]. Tento prístup sa dá použiť na "zapínanie" a "vypínanie" génov pri diferenciácii buniek, na správanie

imúnneho systému, na evolučný proces, ale predovšetkým na modelovanie vzniku života z hypercyklov reakcií jednoduchých aminokyselín a iných zlúčenín prítomných v "prebiotckej polievke". Z týchto jednoduchých zlúčenín potom vznikajú navzájom sa spájajúce a katalyzujúce polyméry, ktoré vytvoria stabilnú "sieť reakcií". Nemalo by teda ísť iba o zlúčeniny RNA.

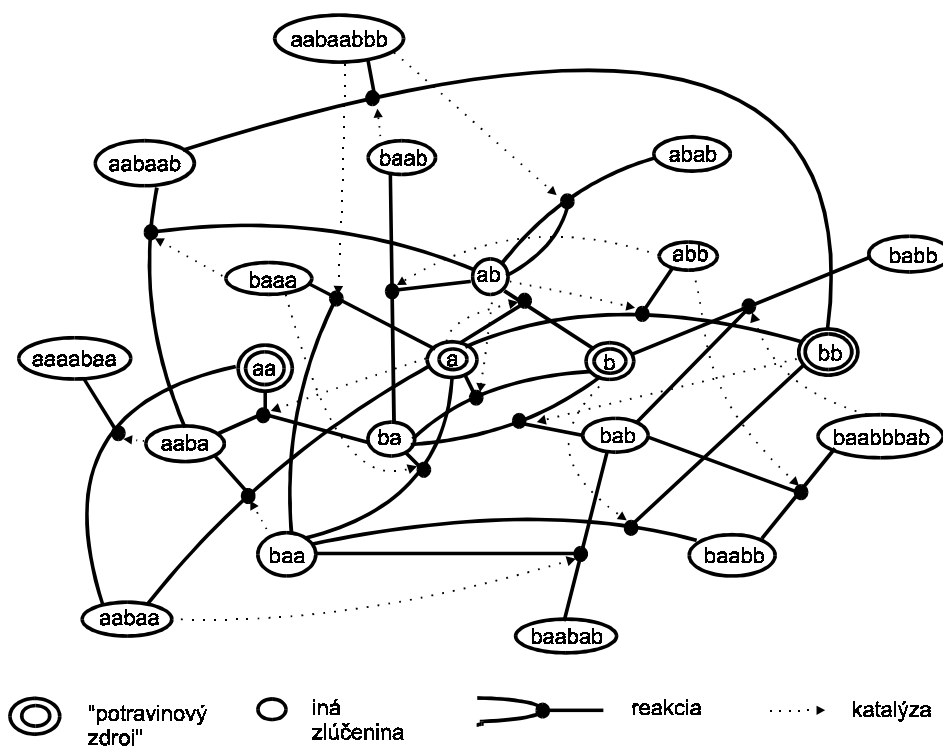


Obrázok 9.17. Generujeme náhodne grafy (teda sústavu vrcholov, kde dvojice vrcholov môžu byť spojené hranami). Pravdepodobnosť výskytu hrany medzi dvoma vrcholmi je vždy rovnaká a zodpovedá výskytu 500 hrán u grafu so 400 vrcholmi. Spočítajme počet vrcholov maximálneho súvislého podgrafu vygenerovaného grafu a zobraziť pomer počtu týchto vrcholov k celkovému počtu vrcholov. Aj napriek tomu, že pravdepodobnosť výskytu hrán v grafoch je stále rovnaká, krivka má sigmoidálny charakter, pretože počet možných hrán sa zvyšuje kvadraticky s počtom vrcholov. V prípade, že vrcholmi sú zlúčeniny a hranami reakcie, daná simulácia napovedá, že pri určitej hranici rôznorodosti prítomných zlúčenín sa väčšina z nich zapojí do siete reakcií - nastáva tvorba hypercyklov.

Tieto predstavy si môžeme priblížiť pomocou jednoduchého modelu. Predstavme si, že máme súbor vrcholov, ktoré náhodne spojíme s určitou pravdepodobnosťou hranami. Pokiaľ budeme zobrazovať pomer počtu vrcholov najväčšieho súvislého grafu (teda takého, kde sa dá po hranách vykonať cesta medzi ľubovoľnými dvoma vrcholmi) k počtu všetkých vrcholov v závislosti od počtu vrcholov, dôjdeme k obrázku 9.17, ktorý pripomína náhlu zmenu štruktúry (takú ako pri zmene ľadu na vodu). Vzniknutá závislosť má nelineárny sigmoidálny tvar; keď počet vrcholov presiahne určitú hranicu, relatívna veľkosť najväčšieho súvislého kusu grafu sa prudko zvýši. To je spôsobené tým, že aj keď pravdepodobnosť výskytu hrany medzi dvoma vrcholmi je stále rovnaká (v našom prípade pre najväčší graf bolo 400 vrcholov a 500 hrán), počet možných hrán stúpa kvadraticky spolu s lineárnym zvýšením počtu vrcholov. Pri určitom množstve vrcholov je ich potom väčšina spojená do jedného celku. Tento príklad môže slúžiť ako analógia pre sieť chemických zlúčenín (vrcholov) spojených reakciami (hranami), kedy od určitej veľkosti bude koncentrácia určitej zlúčeniny závislá od koncentrácií väčšiny ostatných zlúčenín.

Keď si zoberieme množinu všetkých možných zlúčenín, aj pri obmedzení molekulovou hmotnosťou, stabilitou a použitím iba zlomku z chemických prvkov, budeme stále dostávať astronomické čísla. Je isté, že chemické zlúčeniny môžu vznikať mnohými spôsobmi (najčastejšie z dvoch zlúčenín vzniká tretia zlúčenina, prípadne sa zlúčenina transformuje pomocou katalyzátora) a takto je možné pospájať dvojice, trojice alebo štvorice zlúčenín, zúčastňujúcich sa jednej reakcie. Keď máme malý súbor zlúčenín, bude sa medzi nimi vyskytovať aj málo reakcií uspokojivo prebiehajúcich za normálneho tlaku a teploty. No pri

zväčšovaní počtu zlúčenín nastane situácia, kedy väčšina zo zlúčenín bude navzájom prepojená do komplikovanej siete reakcií. Táto sieť sa po čase ustáli do rovnovážneho stavu, čo by malo zodpovedať základnej podmienke pre vznik komplikovanejších štruktúr, takých ako je bunka (viď obr. 9.18).



Obrázok 9.18 Typický príklad malej autokatalytickej siete podľa Kaufmanna [101,102]. Reakcie sú reprezentované štepnými produktmi (alebo eduktmi, smer reakcie závisí od koncentrácií) spolu so zodpovedajúcim väčším polymérom. Bodkované čiarky označujú katalýzu reakcie a smerujú od katalyzátora danej reakcie. Monoméry a diméry A a B predstavujú dodávaný “zdroj živín” (základné zloženie prebiotického polievke).

Podobné zjednodušené príklady napodobujúce princípy chemických reakcií dali vzniknúť aj samostatnému odboru v rámci umelého života, ktorý sa volá *umelá* alebo *algoritmická chémia* [105]. Prehľadný úvod do tejto problematiky urobil Peter Dittrich z Univerzity v Dortmunde [106]. Umelá chémia je definovaná súborom objektov - "molekúl" a súborom pravidiel - reakcií určujúcich spôsob interakcie týchto "molekúl", spolu s popisom aplikácie týchto pravidiel definujúcich dynamiku "populácie molekúl". Pomocou takého systému sa dajú modelovať nielen chemické reakcie, ale napríklad aj správanie sa robota riadeného symbolicky formulovaným ekvivalentom difúzie [107]. Na rozdiel od celulárnych automatov "molekuly" nemusia byť na mriežke a počet objektov (ktorý by inak odpovedal stavom bunky) nemusí byť konečný. Objektmi môžu byť napr. čísla, lambda výrazy, binárne a znakové reťazce a iné dátové štruktúry. Reakcie potom môžu predstavovať jednoduché aritmetické operácie, lambda výpočty, maticovú manipuláciu, zlučovanie reťazcov, spájanie do boolovských sietí alebo prepájanie obvodov v procesoroch. Dynamika sa potom môže riadiť napr. explicitnými simuláciami kolízií, obyčajnými diferenciálnymi rovnicami alebo celulárnym automatom.

Okrem už zmieneného modelovania vzniku života a evolúcie a RNK a DNK je možné týmto prístupom modelovať aj sociálne správanie sa alebo paralelné procesy v modulárnych systémoch. Z praktických aplikácií je možné zmeniť logické spracovanie informácie pri automatickom riadení, optimalizáciu algoritmov v súvislosti s genetickým programovaním alebo výpočty a programovanie pomocou reálnych molekúl. Jedným z prvých implicitne definovaných príkladov konštruktívnej umelej chémie je tzv. *typogenetika* vymyslená Douglasom Hofstadterom, ktorá modeluje vznik enzýmov z genetickej informácie v jadre a spätnú konštrukciu nových kópií reťazcov DNA a RNA pomocou nimi skonštruovaných enzýmov [108].

Jednoduchým príkladom ilustrujúcim skutočnosť, že umelá chémia môže mať so skutočnou chémiou len veľmi málo spoločného, je konštruktívna "čísla deliaca chémia", produkujúca populáciu prvočísel z náhodnej počiatkovej populácie kladných celých čísel. Objekty sú v nej prirodzené čísla okrem jednotky, $S=\{2,3,\dots\}$. Reakčným pravidlom je $s_1 + s_2 \rightarrow s_1 + s_3$, kde $s_3 = s_2/s_1$ pre $s_2 \bmod s_1 = 0$ a $s_2 \neq s_1$, v opačnom prípade $s_3 = s_2$; toto reakčné pravidlo vyjadruje funkcia `reaction( $s_1, s_2$ )` dávajúca výsledné s_3 . Proces začína náhodným generovaním populácie $P[1], P[2], \dots, P[M]$ čísel z množiny S . Potom nasleduje cyklus daný v algoritme 9.3. Funkcia `randomInteger(1, M)` produkuje náhodné celé čísla od 1 po M .

```
while not podmienka_ukončenia_cyklu do
begin
  i1:=randomInteger(1,M);
  s1:=P[i1];
  i2:=randomInteger(1,M);
  s2:=P[i2];
  s3:=reaction(s1, s2);
  if s3≠0 then P[i2]:= s3;
end;
```

Algoritmus 9.3. Konštrukcia prvočísel pomocou konštruktívnej čísla-deliacej chémie. Pre veľkosti populácie $M=100$, počet cyklov cez 5000 a prvú populáciu P generovanú z obmedzeného súboru čísel od 2 do 100 konverguje k populácii obsahujúcej takmer iba prvočísla.

Biologicky orientovaný prístup k evolúcii sa dá rozdeliť na fylogénzu, ktorá sa zaoberá evolúciou prírodného druhu v čase (mutácie a kríženie v genetickom algoritme), epigenézu, ktorá zodpovedá učeniu jednotlivého organizmu v priebehu života (neurónové siete) a ontogenézu, ktorá sa zaoberá vývojom mnohobunkových organizmov zo základnej bunky (autoreprodukujúce sa automaty).

Evolučné inovácie:

1. Symbióza: bakteriálna symbióza ako počiatok tvorby predchodcov buniek ([109]). Namiesto výmeny genetickej informácie krížením tak vzniká v konečnom dôsledku zlúčenie genetickej informácie. Symbióza môže byť vo väčšom meradle zodpovedná aj za zásadnejšie zmeny organizmu, ktoré by inak vyžadovali mnohonásobné simultánne inovácie. V tzv. endosymbiotickej hypotéze ide o vznik eukaryotických buniek (pokročilých buniek s jadrom a zložitými štruktúrami, napr. mitochondriami) integráciou symbiotického systému prokaryotických buniek (primitívnych buniek) do jedného celku. V mikrobiálnej ríši, ktorá tvorí väčšinu života na Zemi, je táto forma "spolupráce na vyššej úrovni" častá. Napokon, bunkové mitochondrie zabezpečujúce výrobu energie v živých organizmoch boli kedysi najskôr tiež samostatnými bunkami.

Táto evolučná zmena je v rozpore s klasickým darwinizmom, lebo znižuje výhody jednotlivca - pohltený jedinec zaniká ako samostatný druh. "Strom života" sa vykresľuje so stále sa rozvetvujúcimi konármi. No symbióza môže znamenať spájanie vetiev.

Okrem pohltenia alebo kríženia je možná výmena informácie aj priamym prenosom génov medzi druhmi, pričom sa často používajú vírusy ako sprostredkovatelia výmeny.

2. Smerované mutácie - keď si predstavíme náhodnú mutáciu spôsobenú kozmickými lúčmi, ťažko ich asi smerovať. No smerovať možno ich opravu. Opravný mechanizmus v bunke pracuje iba nad najdôležitejšími časťami kódu, ktorých zmena by prakticky určite znamenala smrť organizmu. No v menej závažných oblastiach tieto korekčné mechanizmy nemusia byť pre svoju náročnosť vyvinuté. Tým sa zvyšuje pravdepodobnosť zmeny v niektorých smeroch a znižuje v iných, podobne ako je to v Schwefelových evolučných stratégiách. Väčšina genetického kódu u organizmov vyšších ako prvky je totiž nezmyselná (aspoň podľa našich súčasných znalostí) a nikdy sa nepretransformuje do tvorby proteínov. Okrem toho u niektorých proteínov zmena aminokyseliny v niektorých pozíciách nespôsobí zmenu funkčnosti proteínu, takže opravný mechanizmus by bol zbytočný [99]. Tieto zmeny sú v niektorých častiach DNA také pravidelné v priebehu dlhých časových odsekov, že sa podľa nich dá zhruba určovať, kedy sa navzájom oddelili jednotlivé druhy organizmov. Okrem opravného mechanizmu publikoval v r. 1988 Harvardský genetik John Cairns dôkaz, že za určitých podmienok zmeny prostredia bola baktéria *E. coli* sama schopná sa zmutovať v priamej odpovedi na potrebu spôsobenú zmenou prostredia (samozrejme sa predpokladá, že mechanizmus pôsobiaci cieľnú mutáciu vznikol evolúciou [110,111]).
3. Tzv. horizontálny prenos genetickej informácie, kedy sa časť informácie prenáša z jednej "dospelej" bunky na druhú "dospelú" bunku. Môže ísť tak o proces riadený samotnými bunkami, ako aj o proces spôsobený vírusom. Tento prenos môže u baktérií sčasti nahradzovať sexuálny prenos informácie.
4. Saltationismus (z Lat. saltare - skok) je teória snažiaca sa vysvetliť fosílné nálezy, ktoré ukazujú náhly vznik úplne vyvinutého druhu, ktorý potom zostáva viac-menej stabilný až do svojho zániku. Tento jav pomenoval Stephen Jay Gold ako prerušovanú rovnováhu (punctuated equilibrium [112,113]). Skokovú teóriu odmieta väčšina biológov, pretože nie je jasný presný dôvod týchto skokov, okrem podozrenia na zmeny spôsobené v rannom vývoji embrya a možno zmeny "implementácie" genetickej informácie, pre ktorú je rovnako dôležitá chromozomálna DNA ako všetky proteíny, hormóny a nechromozomálna DNA spolu s fyzikálnymi podmienkami, ako je teplota, pH prostredia a pod. Podstatná zmena druhu sa môže umožniť združovaním funkcií, hierarchickou úrovňou riadenia a adaptívnym procesom meniacim celý súbor funkcií súčasne.
5. Samo-organizácia: gény sú spojené do siete vzťahov a podstatná zmena jedného z nich spôsobí nutnosť usadenia sa v inom stabilnom centre. No proti tejto zmene bojuje snaha populácie zostať stabilnou. Aj tá najvýhodnejšia mutácia bude nanič, pokiaľ nebude takýto jedinec prijatý kolektívom, čím bude daná možnosť ďalšieho rozšírenia mutácie rozmnožením potomkov. Stuart Kaufmann [101] na základe výpočtov tvrdí, že za určitou hranicou vzrastajúca hustota spojení medzi prvkami systému (agentmi) zmrazuje adaptáciu. Mutácia navyše musí spĺňať základné požiadavky - byť prijateľná z hľadiska molekulárnej úrovne genómu a potom aj z hľadiska splnenia fyzikálnych požiadaviek na stavbu tela. Tým sú možnosti vzniku živých foriem obmedzené. No vzhľadom na to, že poznáme iba výsledný "výber" - existujúce živé organizmy, a nepoznáme, z koľkých možností boli vyberané, nedá sa odhadnúť "hustota možných živých foriem", koľko bolo možností evolúcie. Fosílné nálezy a súčasné formy života na Zemi sú iba jednou z možných realizácií evolúcie. Odpoveď na otázku počtu možných ciest evolúcie môže naznačiť práve disciplína umelého života.

Trendy evolúcie:

1. Ireverzibilita - evolúcia sa nevracia naspäť, nové formy živočíchov sa nevracajú k vývojovo starším formám (veľryba je síce cicavec, ktorý sa vrátil do mora, no to je skôr výnimka ako pravidlo) a život všeobecne už neopúšťa raz obsadenú niku.
2. Narastajúca zložitost' (neexistuje však presná definícia toho, čo zložitost' je, a nie je daný ani jednoznačný dôvod jej narastania, aj keď existujú hypotézy: zložitost' je spôsobená rôznorodosťou prostredia, ktorej prispôsobené prvky sa skladajú na vytvorenie komplexného systému, je termodynamicky výhodnejšia, je spôsobená špecializáciou hnanou princípom prežitia najsilnejšieho vnútri druhu, atď.).
3. Narastajúca rôznorodosť - táto otázka však nie je celkom jasná, lebo dnes síce máme veľa rôznych druhov hmyzu, no nemáme veľa principiálne rozdielnych rodov. Táto situácia je v rozpore napr. s cambrijským obdobím, kedy existovalo veľmi veľa zásadne rôznych druhov designu, z ktorých prežilo iba zopár.
4. Narastanie počtu jedincov.
5. Narastajúca špecializácia - raz špecializované bunky alebo živočíšne druhy sa väčšinou nevracajú k väčšej flexibilitosti.
6. Narastajúca vzájomná závislosť od iných živočíšnych druhoch a od spoločenstva tvoreného vlastným druhom - zatiaľ čo lišajníky "požierajú priamo kameň" a sú veľmi závislé od fyzického prostredia (kyslý dážď), človek môže žiť takmer kdekoľvek, no potrebuje na prežitie spoluprácu s ostatnými ľuďmi.
7. Narastajúca schopnosť evolúcie. R. 1987 cambridgský zoológ Richard Dawkins na prvom seminári o umelom živote prezentoval článok *Evolúcia schopnosti vývoja* [114]. Navrhol prírodný výber vyššieho druhu, ktorý nepreferuje iba jedincov, ale tendenciu vyvíjať sa určitým smerom, alebo tendenciu vyvíjať sa vôbec. Biológ Leo Buss [115] z Yale University sa zamýšľala nad tým, ako sa vyvíja pojem jedinca. Zo začiatku to mohol byť stabilný systém reakcií polymérov, potom bunka, neskôr organizmus. Ako ďalšia jednotka môže slúžiť celý druh. Kvalita a schopnosť prežitia druhu nemusí bezprostredne závisieť od kvality jednotlivcov, ktorá sa porovnáva vnútrodrohovo. Kvalita druhu môže byť napríklad posudzovaná podľa schopnosti vyvíjať sa.

Programy budúcnosti si možno predstaviť ako pestované neriadené monštra, ktoré 99% času strávia nad vlastnými nezmyselnými operáciami a iba jedno percento venujú na "rozumný" výpočet z hľadiska užívateľa. Táto neefektívnosť by sa mala vyvážiť schopnosťou adaptovať sa na nové udalosti, pre ktoré dosiaľ neexistuje naučená odpoveď systému. Vzhľadom na stále zlacňovanie počítačového času to bude výhodnejšie ako najat' programátora, ktorý vymyslí perfektný malý program. Tento trend je čiastočne badateľný na rozširovaní neefektívnych, no k laikom prívetivých programov firmy Microsoft. Tie sú síce robené ľuďmi, no pravdepodobne vzhľadom k neukončenej evolúcii - malej komunikácii medzi "sieťou" autorov majú skôr vlastnosti neriaditeľného "monštra" s náhodne generovanými "features" (špecifikami, programátormi mimo Microsoftu nazývanými chybami).

Možnosti *samovoľného vzniku inteligencie v programoch umelého života* patria zatiaľ do oblasti sciencie fiction. Základné vlastnosti života ako pohyblivosť v meniacom sa prostredí, zmyslové vnímanie okolia a ostatné vlastnosti nutné na udržanie a rozvíjanie života sa vyvíjali v priebehu miliárd rokov, zatiaľ čo abstraktné myslenie a jazyk iba desiatky tisíc rokov. To by naznačovalo, že vytvorenie umelej inteligencie by malo byť oveľa jednoduchšie ako vytvorenie "živého" reagujúceho a reprodukovajúceho sa organizmu. No podľa mienky autorov je inteligencia vystavaná na základe živého organizmu, a vytvoriť ju na inom základe vyžaduje porovnateľne zložitú "základňu" ako je živá hmota, čo pri celej svojej zložitosti počítače ešte zďaleka neumožňujú.

Literatúra

- [1] S. Levy: *Artificial Life: The Quest for a New Creation*. Pantheon, New York 1992.
- [2] K. Sigmund: *Games of Life: Exploration in Ecology, Evolution and Behaviour*. Oxford University Press, New York 1993.
- [3] C. Emmeche: *The Garden in the Machine: The Emerging Science of Artificial Life*. Princeton University Press, Princeton, NJ 1994.
- [4] K. Kelly: *Out of Control: The Rise of Neo-Biological Civilization*. Addison-Wesley, Reading, MA 1994.
- [5] C. Langton: *Artificial Life: An Overview*. MIT Press, Cambridge, MA 1995.
- [6] C. Adami: *Introduction to Artificial Life*. Springer Verlag, New York 1998.
- [7] C.G. Langton (ed.): *Artificial Life*. SFI Studies in the Sciences of Complexity, vol. 6. The Proceedings of the 1987 Workshop in Los Alamos, NM. Addison-Wesley, Redwood City, CA 1989.
- [8] C. Langton, C. Taylor, D. Farmer, and S. Rasmussen (eds.): *Artificial Life II*. Addison-Wesley, Redwood City, CA 1991.
- [9] C. Langton (ed.): *Artificial Life III*. SFI Studies in the Sciences of Complexity, vol. 17, Addison-Wesley, Redwood City, CA 1993.
- [10] R. Brooks and P. Maes (eds.): *Artificial Life IV*. MIT Press, Cambridge, MA 1994.
- [11] C.G. Langton and T. Shimohara (eds.): *Artificial Life V*. Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems (Complex Adaptive Systems), MIT Press, Cambridge, MA 1997.
- [12] C. Adami, C. Taylor, and R.K. Belew (eds.): *Artificial Life VI*. Proceedings of the Sixth International Conference on Artificial Life, Bradford Books, MIT Press, Cambridge, MA 1998.
- [13] J.-A. Meyer and S.W. Wilson (eds.): *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior* (1990, Paris, France). MIT Press, Cambridge, MA 1991.
- [14] J.-A. Meyer, H.L. Roitblat, and S.W. Wilson (eds.): *From animals to animats II: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*. MIT Press, Cambridge, MA 1993.
- [15] D. Cliff, P. Husbands, J.-A. Meyer, and S.W. Wilson (eds.): *From animals to animats III: Proc. of the 3rd Int'l Conf. on Simulation of Adaptive Behavior*, MIT Press, Cambridge, MA 1994.
- [16] P. Maes, M.J. Mataric, J.-A. Meyer, J. Pollack, and S.W. Wilson (eds.): *From Animals to Animats IV: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*. MIT Press, Cambridge, MA 1996.
- [17] R. Pfeifer, B. Blumberg, J.-A. Meyer, and S.W. Wilson (eds.): *From Animals to Animats V: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*. MIT Press, Cambridge, MA 1998.
- [18] *The Artificial Life Journal*, MIT Press Journals, 55 Hayward Street, Cambridge, MA 02142-1399, USA.
- [19] *Adaptive Behavior*, MIT Press Journals, 55 Hayward Street, Cambridge, MA 02142-1399, USA.
- [20] *Evolutionary Computation*, MIT Press Journals, 55 Hayward Street, Cambridge, MA 02142-1399, USA.
- [21] *Complex Systems*, Complex Systems Publications, Inc., P.O. Box 6149, Champaign, IL 61821-8149, USA.
- [22] *Artificial Life and Robotics*, Springer Verlag, Springer-Auslieferungsgesellschaft GmbH & Co KG, Haberstr. 4, D-69126 Heidelberg Germany.

- [23] *Advances in Complex Systems* (formerly Journal of Complex Systems), HERMES Science Publications, 8, Quai Du Marche Neuf , 75004 Paris, France.
- [24] *Biological Cybernetics*, Springer Verlag, Springer-Auslieferungsgesellschaft GmbH & Co KG, Haberstr. 4, D-69126 Heidelberg Germany.
- [25] *Complexity*, John Wiley & Sons, Inc., 605 Third Avenue, New York, NY 10158-0012.
- [26] *Journal of Complexity*, Academic Press, 6277 Sea Harbor Drive, Orlando, Florida 32887-4900.
- [27] *Journal of Social and Evolutionary Systems*, JAI, Elsevier Science, P.O. Box 211, 1000 AE Amsterdam, The Netherlands.
- [28] *Journal of Theoretical Biology*, Academic Press, 6277 Sea Harbor Drive, Orlando, Florida 32887-4900.
- [29] F.J. Varela and P. Bourguin (eds.): *Toward a practice of autonomous systems. Proceedings of the First European Conference on Artificial Life*. MIT Press/Bradford Books, Cambridge, MA 1992
- [30] F. Moran, A. Moreno, J.J. Merelo, and P. Chacon (eds.): *Advances in Artificial Life: Proceedings of the Third European Conference on Artificial Life*. LNAI 929, Springer Verlag, Berlin 1995.
- [31] P. Husbands and I. Harvey (eds.): *Fourth European Conference on Artificial Life*. MIT Press, Cambridge, MA 1997.
- [32] D. Floreano, J.-D. Nicoud, and F. Mondada (eds.): *Advances in Artificial Life. The Proceedings of 5th European Conference*. Ecal'99, Lausanne, Switzerland, Springer Verlag, Berlin 1999.
- [33] J. Csontó: *Umělý život*. V knihe V. Mařík, O. Štěpánková, J. Lažanský (editori): *Umělá inteligence 3*, Academia, Praha 2000 - v tlači.
- [34] <http://alife.tuke.sk/>
- [35] J. Kelemen: *Strojovia a agenty*. Archa, Bratislava 1994.
- [36] E. Hillebrand and J. Stender: *Many-Agent Simulation and Artificial Life*. IOS Press 1994.
- [37] J. Ferber: *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley 1999.
- [38] J. Jalůvka: *Moderní počítačové viry: podstata, prevence, ochrana*. Computer Press, Praha 1996.
- [39] M. Danilák: *Svet počítačových vírusov*. Grada, Praha 1992.
- [40] N. Magnenat Thalmann and D. Thalmann: *Artificial Life and Virtual Reality*. J. Wiley, New York 1995.
- [41] V. Kvasnička, L. Beňušková, J. Pospíchal, I. Farkaš, P. Tiňo, A. Král: *Úvod do teórie neuronových sietí*. IRIS, Bratislava 1997.
- [42] M. Resnick: *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. Complex Adaptive Systems, MIT Press, Cambridge, MA 1997.
- [43] *SimLife* - Maxis (DOS, Windows), Maxis 2121 N. California Blvd., Suite 600, Walnut Creek, CA 94596-3572.
- [44] P. Prusinkiewicz, A. Lindenmayer: *The Algorithmic Beauty of Plants*. Springer Verlag, New York 1990.
- [45] H.O. Peitgen, H. Jürgens, and D. Saupe: *Chaos and Fractals: New Frontiers of Science*. Springer Verlag, New York 1992.
- [46] T.F. Nonnenmacher, G.A. Losa, and E.R. Weibel (eds): *Fractals in Biology and Medicine*. Birkhauser, Boston 1994.
- [47] G.A. Losa, D. Merlini, T.F. Nonnenmacher, and E.R. Weibel (eds): *Fractals in Biology and Medicine, Vol II*. Birkhauser, Boston 1998.
- [48] T.G. Dewey: *Fractals in Molecular Biophysics*. Oxford University Press, Oxford 1998.

- [49] K. Sims: Artificial Evolution for Computer Graphics. *Computer Graphics* (Siggraph '91 proceedings), **25**(4) (1991) 466-476.
- [50] K. Sims: Evolving 3D Morphology and Behavior by Competition. In R. Brooks and P. Maes (eds.): *Artificial Life IV*. MIT Press, Cambridge, MA 1994, pp. 28-39.
- [51] R. Dawkins: *The Blind Watchmaker*. E.E. Norton, New York 1987.
- [52] A.J. Lotka: *Elements of Physical Biology*. Williams and Wilkins, Baltimore 1925.
- [53] V. Volterra: Fluctuations in the abundance of a species considered mathematically. *Nature* **118** (1926) 558-560.
- [54] E. Renshaw: *Modelling Biological Populations in Space and Time*. Cambridge University Press, Cambridge 1991.
- [55] P.F. Verhulst: Notice sur la loi que la population suit dans son accroissement. *Corr. Math. et Phys. publ. par A. Quetelet* **T.X** (1838) 113-121.
- [56] R. Pearl and L.J. Reed: On the rate of growth of the population of the United States since 1790 and its mathematical representation. *Proceedings of the National Academy of Sciences* **6** (1920) 275-288.
- [57] R.J. Collins and D.J. Jefferson: AntFarm: Towards Simulated Evolution. In C. Langton, C. Taylor, D. Farmer, and S. Rasmussen (eds.): *Artificial Life II*. Addison-Wesley, Redwood City, CA 1991, pp. 579-602.
- [58] E. Bonabeau, M. Dorigo, G. Theraulaz: *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, Oxford 1999.
- [59] C. Reynolds: Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics* **21** (1987) 25-34.
- [60] H. Gutowitz: *Cellular Automata: Theory and Experiment*. MIT Press/Bradford Books, Cambridge, MA 1991.
- [61] S. Wolfram (ed.): *Theory and Applications of Cellular Automata*. World Scientific, Singapore 1986.
- [62] J. von Neumann: *Theory of Self-Reproducing Automata*. Completed and edited by A.W. Burks. University of Illinois Press, Champaign-Urbana 1966.
- [63] C.G. Langton: Self-Reproduction in Cellular Automata. *Physica D* **10** (1984) 135-144.
- [64] M. Gardner: The fantastic combination of John Conway's new solitaire game "Life". *Scientific American* **223** (1970) 120-123.
- [65] M. Gardner: Mathematical games: on cellular automata, self-reproduction, the Garden of Eden and the game "Life". *Scientific American* **224**(2) (1971) 112-117.
- [66] E.R. Berlekamp, J.H. Conway, and R.K. Guy: *Winning Ways for your Mathematical Plays, volume 2*. Academic Press, New York 1982, Chapter 25.
- [67] W. Poundstone. *The Recursive Universe*. William Morrow and Company, New York 1985.
- [68] J. Thatcher: Self-Describing Turing Machines and Self-Reproducing Cellular Automata. In A. Burks (ed.): *Essays on Cellular Automata*. University of Illinois Press, Urbana 1970, pp. 103-131.
- [69] C.G. Langton: Life at the Edge of Chaos. In C. Langton, C. Taylor, D. Farmer, and S. Rasmussen (eds.): *Artificial Life II*. Addison-Wesley, Redwood City, CA 1991, pp. 41-91.
- [70] S. Wolfram: Universality and Complexity in Celular Automata. *Physica D* **10** (1984) 1-35.
- [71] T. Toffoli: Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics. *Physica D* **10** (1984) 117-127.
- [72] U. Frisch, B. Hasslacher, and Y. Pomeau: Lattice-gas automata for the Navier-Stokes equation. *Physical Review Letters* **56** (1986) 1505-1508.

- [73] J. Hardy, O. De Pazzis, and Y. Pomeau: Molecular dynamics of a classical lattice gas: Transport properties and time correlation functions. *Physical Review A* **13** (1976) 1949-1960.
- [74] C. Bennett and G. Grinstein. Role of irreversibility in stabilizing complex and nonenergetic behavior in locally interacting discrete systems. *Physical Review Letters* **55** (1985) 657-660.
- [75] G. Vichniac: Simulating physics with cellular automata. *Physica D* **10** (1984) 96-115.
- [76] G.B. Ermentrout and L. Edelstein-Keshet: Cellular automata approaches to biological modeling. *Journal of Theoretical Biology* **160** (1993) 97-133.
- [77] M. Sipper: *Evolution of Parallel Cellular Machines, The Cellular Programming Approach*. LNCS 1194, Springer Verlag, Berlin 1997.
- [78] M. Sipper, D. Mange, and A. Peres-Urbe (eds.): *Evolvable Systems: From Biology to Hardware*. The Proceedings of the Second International Conference, Ices 98 Lausanne, Switzerland. Springer Verlag, Berlin, 1998.
- [79] A. Thompson: *Hardware Evolution: Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution*. Springer Verlag, London 1998.
- [80] D. Mange and M. Tomassini (eds.): *Bio-Inspired Computing Machines*. Presses Polytechniques et Universitaires Romandes, Lausanne 1998.
- [81] P. Maes. Designing autonomous agents: Theory and practice from biology to engineering and back. *Robotics and Autonomous Systems* **6** (1990) 1-2.
- [82] T. Gomi (ed.): *Evolutionary Robotics: From Intelligent Robots to Artificial Life*. Proceedings of ER'97, Ontario (Canada), AAI Books 1998.
- [83] P. Husbands (ed.): *Evolutionary Robotics*. Proceedings of First European Workshop, EvoRobot'98, Paris, France, April 16-17, 1998. LNCS 1468, Springer Verlag, Berlin 1998.
- [84] S. Nolfi and D. Floreano: *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines (Intelligent Robotics and Autonomous Agents)*. MIT Press, Bradford Books, 2000.
- [85] R.A. Brooks: Intelligence without representation. *Artificial Intelligence* **47** (1991) 139-160.
- [86] R.A. Brooks: Artificial life and real robots. In F.J. Varela and P. Bourguine (eds.): *Toward a practice of autonomous systems. Proceedings of the First European Conference on Artificial Life*, MIT Press/Bradford Books, Cambridge, MA 1992, pp. 3-10.
- [87] R. Axelrod: *The evolution of cooperation*. Basic Books, New York 1984.
- [88] R. Axelrod: *The Complexity of Cooperation: Agent-Based Models of Conflict and Cooperation*. Princeton University Press, Princeton 1997.
- [89] R. Axelrod: The evolution of strategies in the iterated prisoner's dilemma. In D. Davis: *Genetic algorithms and simulated annealing*. Pitman, London 1987, pp. 32-43.
- [90] D.W. Hillis: Co-evolving parasites improve simulated evolution as optimization procedure. In Langton, C., C. Taylor, J.D. Farmer, and S. Rasmussen (eds.): *Artificial Life II*, Santa Fe Institute Studies in Complexity, vol. X, 313-324. Addison-Wesley, Redwood City, CA 1991.
- [91] B. Olsson: Optimization Using a Host-Parasite Model with Variable-Size Distributed Population. *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*. Nagoya 1996.
- [92] D. Layzer: *Cosmogenesis: The Growth of Order in the Universe*. Oxford University Press, New York 1990.
- [93] K. Lindgren: Evolutionary phenomena in simple dynamics. In C. Langton, C. Taylor, D. Farmer, and S. Rasmussen (eds.): *Artificial Life II*. Addison-Wesley, Redwood City, CA 1991, pp. 295-312.

- [94] T.S. Ray: An approach to the synthesis of life. In C. Langton, C. Taylor, D. Farmer, and S. Rasmussen (eds.): *Artificial Life II*. Addison-Wesley, Redwood City, CA 1991, pp. 371-408.
- [95] A.N. Pargellis, The spontaneous generation of digital "Life". *Physica D* **91** (1996) 86-96.
- [96] J. Singh, M.A. Ator, E.P. Jaeger, M.P. Allen, D.A. Whipple, J.E. Solowej, S. Chowdhary, and A.M. Treasurywala: Application of Genetic Algorithms to Combinatorial Synthesis: A Computational Approach to Lead Identification and Lead Optimisation. *J. American Chemical Society* **118** (1996) 1669-1676.
- [97] D.E. Clark and D.R. Westhead: A Review of Evolutionary Algorithms in Computer-Aided Molecular Design. *J. Computer-Aided Molecular Design* **10** (1996) 337-358.
- [98] S. Miertus and G. Fassina (eds.): *Combinatorial Chemistry and Technology: Principles, Methods, and Applications*. Marcel Dekker, New York 1999.
- [99] A. Markoš: *Povstávání živého tvaru*. Vesmír, Praha 1997.
- [100] J. Maynard Smith: *Evolution and the Theory of Games*. Cambridge University Press, Cambridge 1982.
- [101] S. Kauffman: *Origins of Order: Self-Organization and Selection in Evolution*. Oxford U. Press, New York 1993.
- [102] S. Kauffman: *At Home in the Universe: The Search for Laws of Self-Organization and Complexity*. Oxford U. Press, N.Y. 1995.
- [103] S.M. Miller and L.E. Orgel: *The Origins of Life on Earth*. Prentice-Hall, Englewood, CA 1967.
- [104] M. Eigen and P. Schuster: *The hypercycle: A Principle of Natural Self-organization*. Springer Verlag, Berlin 1979.
- [105] W. Fontana: *Algorithmic chemistry*. In C. Langton, C. Taylor, D. Farmer, and S. Rasmussen (eds.): *Artificial Life II*. Addison-Wesley, Redwood City, CA 1991, pp. 159-209.
- [106] <http://ls11-www.informatik.uni-dortmund.de/achem/>
- [107] J. Ziegler, P. Dittrich, and W. Banzhaf : Towards a Metabolic Robot Control System. In M. Holcombe and R. Paton (eds.): *Proceedings of Information Processing in Cells and Tissues IPCAT 97*. Plenum Press, New York 1998, pp. 305-318.
- [108] D.R. Hofstadter: *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books, Inc., New York 1979.
- [109] L. Margulis and R. Fester (eds.): *Symbiosis as a Source of Evolutionary Innovation: Speciation and Morphogenesis*. The MIT Press, Cambridge, MA 1991.
- [110] P.L. Foster and J. Cairns: Mechanisms of Directed Mutation. *Genetics* **131** (1992) 783-789.
- [111] G.B. Hall: Spontaneous Point Mutations That Occur More Often When Advantageous Than When Neutral. *Genetics* **126** (1990) 5-16.
- [112] N. Eldredge and S.J. Gould: Punctuated Equilibria: An Alternative to Phyletic Gradualism. In T.J.M. Schopf (ed.): *Models in paleontology*. Freeman Cooper, San Francisco 1972.
- [113] S.J. Gould: *Wonderful Life: The Burgess Shale and the Nature of History*. Norton, New York 1989.
- [114] R. Dawkins: Evolution of Evolvability. In C.G. Langton (ed.): *Artificial Life*. SFI Studies in the Sciences of Complexity, vol. 6. The Proceedings of the 1987 Workshop in Los Alamos, NM. Addison-Wesley, Redwood City, CA 1989, pp. 201-220.
- [115] L.W. Buss: *The Evolution of Individuality*. Princeton University Press, Princeton 1987.

10. Neurónové siete

10.1 Historický úvod

Umelé neurónové siete [1-9] pôvodne vznikli ako modely mozgu resp. nervového systému a jeho činnosti. Postupne sa však rozvinuli do nástrojov strojového učenia, používaných napr. na automatické rozpoznávanie priestorových objektov alebo časových postupností signálov, zhukovú analýzu a pod. Neurónová sieť je v podstate súborom jednoduchých prvkov (uzlov, neurónov) vykonávajúcich jednoduché operácie. Spojenia neurónov sú zvyčajne ohodnotené reálnymi číslami - váhami. Vo svojej najrozšírenejšej podobe (viacvrstvové s dopredným šírením) neurónové siete dostanú príklady so správnou odpoveďou, ktoré sa "naučia", a potom dokážu odpovedať aj na "otázky" alebo klasifikovať vzory, ktoré nie sú celkom totožné s naučenými, ale sú im iba podobné. Takéto neurónové siete môžu fungovať ako univerzálny aproximátor, teda modelovať s požadovanou presnosťou akúkoľvek spojitú funkciu, pokiaľ dostanú správne vzory na učenie. V takom prípade sú vlastne univerzálnym prostriedkom regresnej analýzy.

Problém však je zvyčajne s učením, teda nastavením parametrov siete (napr. hodnôt váh alebo množstva neurónov [10-12]). Nastaviť parametre tak, aby sieť čo najlepšie fungovala, zodpovedá optimalizácii, kedy premenné sú parametre funkcie a minimalizujeme chybu predpovede siete. Najčastejšie sa takto optimalizujú parametre viacvrstvovej neurónovej siete so sigmoidálnym prenosom a dopredným šírením (viď ďalej obr. 10.3-10.5).

Častočne vedľajšou aplikáciou je použitie optimalizačných algoritmov na výber (=redukciu) tréningových (cvičných) dát so známymi správnymi odpoveďami, ktoré sa sieť učí predpovedať. Pri aplikácii neurónových sietí ako klasifikátorov a prediktorov je dôležité pre efektívnosť adaptačného procesu (učenia), ale aj pre interpretáciu výsledkov, dobre vybrať deskriptory (teda vstupné dáta) opisujúce vzory, ktoré sa má sieť naučiť interpretovať a/alebo predikovať. Objekt (vzor) môže byť opísaný vektorom rôznorodých dát, z ktorých iba časť môže byť významná pre klasifikáciu. Pretože naučenie neurónovej siete je zložité, môže sa pre výber dát podstatných pre klasifikáciu použiť iná, rýchlejšia, ale menej presná metóda a neurónová sieť sa potom iba používa pre dáta redukované s využitím tejto menej presnej metódy. Takou menej presnou metódou je napr. klasifikátor KNN (ang. K Nearest Neighbors) [1], ktorý ohodnocuje testovaný objekt na základe najbližších K objektov (vzdialenosť objektu od testovaného objektu sa určuje ako súčet absolútnych hodnôt rozdielov jednotlivých prvkov vektorov opisujúcich tieto dva objekty). Otázka je, ktoré prvky z opisných vektorov sa môžu pri výpočte tejto vzdialenosti vynechať. To sa dá opísať vektorom núl (vynechať) a jednotiek (započítať). Pri takomto výbere tréningových dát, vlastne optimalizácii binárneho vektora, sa ukázalo výhodné použiť genetický algoritmus [13, 14].

Podobne môžeme pomocou genetických algoritmov hľadať tiež oblasti (väčšinou obdĺžniky alebo hyperkvádre) v prehľadávanom priestore, v ktorých sa namerali požadované výstupné hodnoty. Z takto vytvorených pravidiel (keď sú vstupné dáta v istom rozmedzí, potom výsledok bude tiež v určitom rozmedzí) môžeme potom vytvoriť expertný systém. Žiaľ, namerané výstupné dáta pre nejaké body v prehľadávanom priestore často chýbajú, a tak sa nahradzujú neurónovou sieťou, ktorá je už naučená aproximovať tvar funkcie z ostatných vstupov a výstupov [15].

Ďalšou atypickou aplikáciou je hľadanie takých dát, ktoré zodpovedajú zadaným hodnotám výstupov už naučenej neurónovej siete [16, 17]. Genetický algoritmus sa takto využil na hľadanie hraničných, ťažko rozhodnuteľných prípadov (ohodnotenie 0.5), kedy neurónová sieť bola naučená diagnostikovať zo vstupných dát, či pacient má alebo nemá zápal slepého čreva (má = ohodnotenie 1, nemá = 0). Takéto výsledky môžu byť použiteľné

jednak na extrakciu pravidiel, podľa ktorých sa neurónová sieť rozhodla (pokiaľ s klasifikáciou 0.5 súhlasíme), alebo na rozhodnutie, či sa sieť naučila dobre (napr. pokiaľ s klasifikáciou nesúhlasíme).

Neurónové siete alebo ich vstupné dáta sú teda väčšinou objektom optimalizácie.

Zásadne odlišnou možnosťou prepojenia neurónových sietí s optimalizáciou je použitie špeciálne prispôbených neurónových sietí ako optimalizačných metód. Neurónové siete, keď už sa používajú na optimalizáciu (čo je skôr výnimka, väčšinou sa používajú namiesto polynomiálnej regresie), tak sa používajú predovšetkým na riešenie špecializovaných problémov. Takýmto problémom je napr. spracovanie obrazu Hopfieldovou neurónovou sieťou, čo je rekonštrukcia objektu zo zašumeného alebo rozmazaného obrazu. Na tento účel Hopfieldova sieť funguje vynikajúco [18-23], no ide o veľmi špecializovaný problém. Hopfieldova sieť však funguje aj pre klasické úlohy kombinatorickej optimalizácie [24], kedy si musíme zvoliť iba vhodnú reprezentáciu dát. Ide konkrétne o problémy najcennejšieho párenia (weighted matching), problém obchodného cestujúceho (Traveling Salesperson Problem) [25-27], alebo aj rozdelenie vrcholov grafov na dve podmnožiny (graph bipartitioning) [28]. Problém obchodného cestujúceho sa dá riešiť aj pomocou Kohonenovej neurónovej siete [29-31]. Problém je v tom, že naposledy menované problémy riešia neurónové siete nie veľmi efektívne a v praxi sa preto na tieto úlohy neurónové siete nevyužívajú, aj keď sa tieto algoritmy rôznymi úpravami postupne vylepšujú. Riešenia týchto problémov sa väčšinou publikujú iba pre teoretikov ako ilustračné príklady toho, že neurónovou sieťou možno riešiť aj takéto druhy úloh.

Ďalej sa teda budeme zaoberať tak optimalizáciou neurónových sietí, ako aj optimalizáciou neurónovými sieťami.

10.2 Viacvrstvové neurónové siete s dopredným šírením

10.2.1 Všeobecný klasifikačný problém

Zavedieme všeobecnú formuláciu klasifikačného problému pomocou pojmu zobrazenia—funkcie definovanej nad dvomi množinami A a B . Tento prístup bude užitočný pre interpretáciu neurónových sietí ako *klasifikátora* alebo *prediktora*. Nech $F(\mathbf{x})$ je funkcia definovaná nad množinou A , ktorá priradí každému prvku $\mathbf{x} \in A$ obraz—funkčnú hodnotu z množiny B , $\hat{\mathbf{x}} = F(\mathbf{x}) \in B$,

$$F : A \rightarrow B \quad (10.1)$$

Nech $G(\mathbf{x}, \mathbf{w})$ je funkcia, ktorej argumenty sú z konečnej podmnožiny $A_{train} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_r\} \subset A$ (nazývanej *tréninová množina*) a $\mathbf{w}$ je parameter (alebo parametre) zobrazenia G , potom $\hat{\mathbf{x}} = G(\mathbf{x}, \mathbf{w}) \in B_{train} \subset B$ (pozri obr. 10.1)

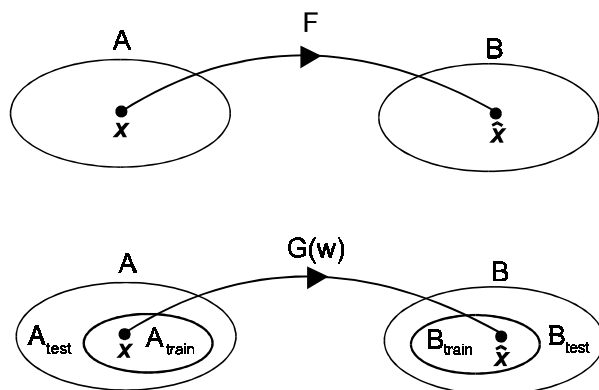
$$G(\mathbf{w}) : A_{train} \rightarrow B_{train} \quad (10.2)$$

Formálne môžeme povedať, že zobrazenie $G(\mathbf{w})$ je reštrikcia zobrazenia $F(\mathbf{x})$ nad množinou $A_{train} \subset A$. Komplement množiny A_{train} vzhľadom na množinu A je označený A_{test} (nazývaný *testovacia množina*), $A_{test} = A \setminus A_{train}$. Predpokladajme, že pre každé $\mathbf{x}_i \in A_{train}$ poznáme požadovaný obraz—funkčnú hodnotu $\hat{\mathbf{x}}_i$,

$$\mathbf{x}_1 / \hat{\mathbf{x}}_1, \mathbf{x}_2 / \hat{\mathbf{x}}_2, \dots, \mathbf{x}_r / \hat{\mathbf{x}}_r \quad (10.3)$$

Požadované funkčné hodnoty $\hat{\mathbf{x}}_i$ sa interpretujú ako obrazy funkcie F

$$\hat{\mathbf{x}}_i = F(\mathbf{x}_i) \quad (i = 1, 2, \dots, r) \quad (10.4)$$



Obrázok 10.1. Schematické znázornenie zobrazenia $F: A \rightarrow B$. Zúžením tohto zobrazenia na podmnožinu A_{train} dostaneme nové “modelové” zobrazenie $G(w)$, funkčný tvar tohto zobrazenia je určený parametrom (parametrami) w .

Cieľom našich úvah je nájsť taký parameter (alebo parametre) w funkcie $G(x, w)$, aby funkčné hodnoty argumentov z tréningovej množiny A_{train} boli čo najbližšie obrazom funkcie $F(x)$ (t.j. požadovaným hodnotám). Definujme *účelovú funkciu*

$$E(w) = \frac{1}{2} \sum_{i=1}^r (G(x_i, w) - \hat{x}_i)^2 \quad (10.5)$$

Táto funkcia vyjadruje súčet štvorcov odchýlok funkcie $G(x, w)$ od požadovaných hodnôt $\hat{x}$ braných z tréningovej množiny. Požiadavka, aby vypočítané hodnoty $G(x, w)$ boli “čo najbližšie” požadovaným hodnotám $\hat{x}$, sa realizuje pomocou požiadavky minimálnosti účelovej funkcie $E(w)$ vzhľadom na parameter w . Hovoríme, že funkcia $G(x, w)$ je *adaptovaná*, ak jej parameter w je vybraný tak, aby sa rovnal svojej optimálnej hodnote (t.j. pre ktorý má účelová funkcia globálne minimum). Nech $\bar{w}$ je optimálna hodnota parametra w určená nasledujúcim minimalizačným problémom

$$\bar{w} = \arg \min_{w \in W} E(w) \quad (10.6)$$

kde W je množina (priestor) prípustných hodnôt parametra w . Adaptovaná funkcia $G(x, \bar{w})$ simuluje pôvodnú funkciu $F(x)$ pre hodnoty argumentov z tréningovej množiny A_{train} na základe minimalizačného kritéria (10.6). Okrem toho sa adaptovaná funkcia $G(x, \bar{w})$ používa aj na predpoveď funkčných hodnôt zodpovedajúcich argumentom z testovacej množiny A_{test} , t.j. predpokladá sa, že adaptovaná funkcia dobre aproximuje pôvodnú funkciu $F(x)$ tiež mimo tréningovej množiny. Naše úvahy sa môžu jednoducho chápať ako klasický regresný problém, v ktorom sú parametre modelovej funkcie G optimalizované (adaptované) tak, aby vypočítané funkčné hodnoty boli blízke požadovaným (experimentálnym) funkčným hodnotám.

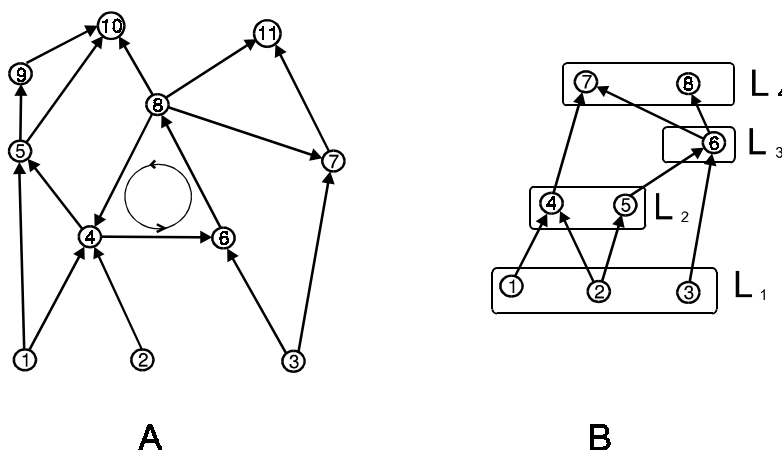
Jeden zo základných problémov v každej oblasti prírodných vied je *hľadanie vzťahu — funkcie medzi štruktúrou jej objektov a ich vlastnosťami*. Ideálom je konštrukcia tejto funkcie v analytickom tvare, ktorá vzťahuje vlastnosti objektov k ich štruktúre. V mnohých prípadoch je tento cieľ buď vôbec nerealizovateľný alebo len s veľkými ťažkosťami. Preto sa pomerne často používa prístup kombinujúci čiastočné znalosti problému s adaptáciou. Modelová funkcia G sa zostrojí na základe určitých úvah a jej voľne nastaviteľné parametre sa určia pomocou minimalizácie účelovej funkcie $E(w)$ (vzťah (10.6)). Takto adaptovaná modelová funkcia G sa potom berie ako analytický vzťah medzi štruktúrou prírodovedných objektov a ich vlastnosťami.

10.2.2 Definícia neurónovej siete

Formálne je neurónová sieť určená ako orientovaný graf $G=(V,E)$ (pozri obr. 10.2). Výrazy $V=\{v_1, v_2, \dots, v_N\}$ a $E=\{e_1, e_2, \dots, e_M\}$ označujú neprázdnu vrcholovú množinu resp. hranovú množinu grafu G obsahujúceho N vrcholov (neurónov) a M hrán (spojov). Každý spoj $e \in E$ sa interpretuje ako usporiadaná dvojica dvoch neurónov z množiny V , $e=(v, v')$. Hovoríme, že spoj e začína v neuróne v a končí v neuróne v' . Množina neurónov V je rozložená na disjunktné podmnožiny (pozri obr. 10. 2)

$$V = V_I \cup V_H \cup V_O \quad (10.7)$$

kde V_I obsahuje N_I vstupných (*input*) neurónov, ktoré sú susedné len s odchádzajúcimi hranami, V_H obsahuje N_H skrytých (*hidden*) neurónov, ktoré sú susedné súčasne s odchádzajúcimi ako aj s prichádzajúcimi hranami, a konečne V_O obsahuje N_O výstupných (*output*) neurónov, ktoré sú susedné len s prichádzajúcimi hranami. V našich nasledujúcich úvahách budeme vždy predpokladať, že množiny V_I a V_O sú neprázdne, t.j. neurónová sieť obsahuje vždy aspoň jeden vstupný a jeden výstupný neurón.



Obrázok 10.2. Neurónová sieť je definovaná ako orientovaný súvislý graf. Diagram A obsahuje orientovaný graf s jedným cyklom, a teda sa nemôže použiť na definíciu neurónovej siete s dopredným šírením. Diagram B ilustruje možnosť rozkladu vrcholov (neurónov) acyklického orientovaného grafu na vrstvy $L_1, \dots, L_4$.

V acyklických neurónových sieťach, ktoré neobsahujú orientované cykly (také ako v grafe A na obr. 10.2), môžu byť neuróny usporiadané do vrstiev (pozri graf B na obr. 10.2)

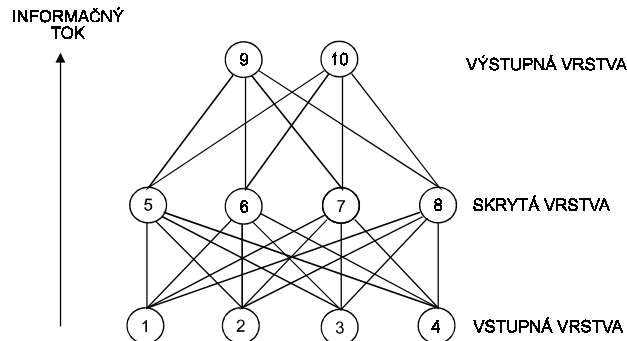
$$V = L_1 \cup L_2 \cup L_3 \cup \dots \cup L_t \quad (10.8)$$

kde $L_1=V_I$ je vstupná vrstva (obsahuje len vstupné neuróny), $L_2, L_3, \dots, L_{t-1}$ sú skryté vrstvy a L_t je výstupná vrstva. Vrstva L_i (pre $1 \leq i \leq t$) je určená nasledujúcim jednoduchým spôsobom

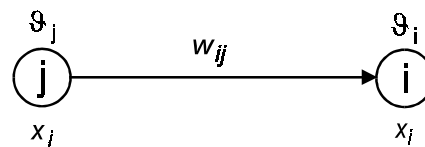
$$L_i = \{v \in V; d(v) = i + 1\} \quad (10.9)$$

kde vzdialenosť $d(v)$ sa rovná dĺžke maximálnej cesty, ktorá spája daný neurón so vstupným neurónom, potom musí platiť $d(v)=0$ pre $v \in V_I$. Neurónová sieť určená acyklickým grafom sa obvykle volí tak, že neuróny z dvoch susedných vrstiev sú poprepájané všetkými možnými spojmi (pozri obr. 10.3). Žiaľ, takýto rozklad množiny neurónov na vrstvy je možný len pre

neurónové siete reprezentované acyklickými grafmi, pre cyklické grafy vzdialenosť $d(v)$ môže nadobúdať ľubovoľnú kladnú celočíselnú hodnotu.



Obrázok 10.3. Znáznornenie trojvrstvovej neurónovej siete s dopredným šírením.



Obrázok 10.4. Neuróny a spoje neurónovej siete sú ohodnotené reálnymi číslami: neuróny sú ohodnotené dvoma rôznymi parametrami, a to prahovými faktormi ϑ_i a aktivitami x_i . Spoje neurónovej siete sú ohodnotené váhovými koeficientmi w_{ij} .

Neuróny a spoje sú ohodnotené reálnymi číslami (pozri obr. 10.4). Každý neurón v_i je ohodnotený *prahovým koeficientom* ϑ_i a *aktivitou* x_i . Podobne každý spoj (v_j, v_i) je ohodnotený váhovým koeficientom (alebo jednoducho váhou) w_{ij} . Postulujeme, že aktivity skrytých a výstupných neurónov sú určené vzťahom

$$x_i = t(\xi_i) \quad (10.10a)$$

$$\xi_i = \sum_{j \in \Gamma_i^{-1}} w_{ij} x_j + \vartheta_i \quad (10.10b)$$

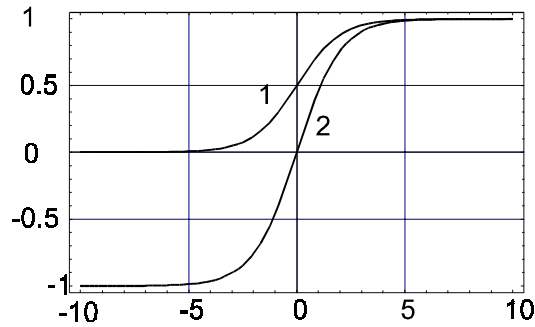
kde súčet ide cez neuróny, ktoré sú predchodcami neurónu v_i . Veličina ξ_i sa nazýva *potenciál* neurónu v_i . *Prechodová (aktivačná) funkcia* $t(\xi)$ z pravej strany (10.10a) je monotónne rastúca funkcia, ktorá vyhovuje nasledujúcim dvom asymptotickým podmienkam: $t(\xi) \rightarrow A$ pre $\xi \rightarrow -\infty$ a $t(\xi) \rightarrow B$ pre $\xi \rightarrow \infty$, kde $-\infty < A < B < \infty$. V teórii neurónových sietí sa často využíva nasledujúca "sigmoidálna" funkcia

$$t(\xi) = \frac{B + Ae^{-\xi}}{1 + e^{-\xi}} \quad (10.11)$$

Táto prechodová funkcia zobrazuje celú množinu reálnych čísel R na otvorený interval (A, B) , formálne $t: R \rightarrow (A, B)$. Najčastejšie sa prechodová funkcia (10.11) využíva pre hodnoty parametrov $A=0$, $B=1$ alebo $A=-1$, $B=1$ (pozri obr. 10.5). Prvý graf zodpovedá klasickej sigmoidálnej prechodovej funkcii, zatiaľ čo druhý graf je analógiou hyperbolického tangensu.

Sieť dostane ako vstup aktivačný vzor pre svoje vstupné jednotky (inak povedané, vstupné aktivity, ako napr. hodnoty pixelov v obrázku ručne písaného písmena A, sa pridajú vstupným neurónom). Aktivácie sa šíria dopredu zo vstupných jednotiek cez jednu alebo viac prostredných vrstiev "skrytých" neurónov spojeniami ohodnotenými váhami, až skončia vo

výstupných jednotkách. Aktivácie sú v priebehu šírenia z jednotky na jednotku násobené váhami spojenia, cez ktoré idú, a potom sú sčítané spolu s ostatnými vstupujúcimi aktiváciami. Tento súčet je spracovaný aktivačnou funkciou. Takto vytvorená aktivácia sa potom rovnakým spôsobom šíri ďalej až do výstupného neurónu. Aktivácie výstupných neurónov sú zakódovanou "odpoveďou" siete na vstup, napríklad ktoré písmeno bolo vstupom siete.



Obrázok 10.5. Priebeh aktivačnej funkcie definovanej vzťahom (10.11). Graf 1 zodpovedá štandardnej sigmoide ($A=0, B=1$), graf 2 je podobný funkcii hyperbolický tangens ($A=-1, B=1$).

V poslednej dobe sa používajú genetické algoritmy aj pre rekurentné neurónové siete, ktoré obsahujú aj spätné spojenia medzi vrstvami [32]. Topológia a váhy siete sa pritom optimalizujú súčasne.

Aktivity neurónov tvoria vektor $\mathbf{x}=(x_1, x_2, \dots, x_N)$. Tento vektor možno formálne rozložiť na tri podvektory obsahujúce vstupné, skryté a výstupné aktivity

$$\mathbf{x} = \mathbf{x}_I \oplus \mathbf{x}_H \oplus \mathbf{x}_O \quad (10.12)$$

Neurónovú sieť s fixovanými váhami a prahovými koeficientmi možno formálne chápať ako funkciu

$$G: R^{N_I} \rightarrow (A, B)^{N_O} \quad (10.13)$$

ktorá priradí vstupnej aktivite $\mathbf{x}_I$ (deskriptor) výstupný vektor $\mathbf{x}_O$ (klasifikátor) s hodnotami svojich zložiek z otvoreného intervalu (A, B)

$$G(\mathbf{x}_I) = \mathbf{x}_O \quad (10.14)$$

Skryté aktivity nie sú explicitne uvedené, hrajú len úlohu medzivýsledkov. Ešte niekoľko poznámok k výpočtu aktivít podľa (10.10a-b). Vstupné aktivity sú určené deskriptorom, preto ich pokladáme za fixované. Aktivity skrytých neurónov z druhej vrstvy L_2 môžeme teraz spočítať len s použitím vstupných aktivít z vrstvy L_1 . Vo všeobecnosti na výpočet aktivít z vrstvy L_i (kde $i > 1$) musíme poznať len aktivity z nižších vrstiev $L_1, L_2, \dots, L_{i-1}$. Týmto rekurentným spôsobom môžeme postupne spočítať aktivity všetkých neurónov; ako posledné sa počítajú aktivity výstupných neurónov. Vďaka tomu sa pre neurónové siete reprezentované acyklickým grafom zaužíval názov *neurónové siete s dopredným šírením* (*feed-forward neural networks*). Žiaľ, spomínaný jednoduchý postup výpočtu aktivít neurónov je aplikovateľný len na neurónové siete reprezentované acyklickým orientovaným grafom. V prípade, že graf obsahuje orientované cykly (ide o rekurentné siete), nie je tento postup použiteľný. Rovnice (10.10a-b) sú v tomto prípade spriahnuté a nelineárne. Preto ich riešenie (t.j. skryté a výstupné aktivity) môžeme získať len použitím iteračného postupu, a to tak, že štartujeme z počiatočných aktivít, pomocou týchto spočítame nové aktivity a tieto sa v

nasledujúcom iteračnom kroku použijú ako vstup pre výpočet nových aktivít. Tento iteračný postup sa opakuje dovtedy, kým nie je rozdiel medzi starými a novými aktivitami menší ako predpísaná presnosť.

Vo väčšine aplikácií sa sieť učí správne mapovanie medzi vstupnými a výstupnými vzormi pomocou učiaceho algoritmu. Nastavenie váhových koeficientov neurónových sietí bolo dlho veľkým problémom. Až Rumelhart so spolupracovníkmi [33] navrhli jednoduchý gradientový algoritmus (nazvaný metóda spätného šírenia - backpropagation) adaptácie viacvrstvových neurónových sietí s dopredným šírením. Typicky sú váhy spočiatku nastavené na malé náhodné hodnoty. Potom sa sieť postupne zadáva množina tréningových vzorov. Keď každý vstup (vzor) prejde sieťou a pre každý vzor sa vyprodukuje výstup, "učiteľ" porovná aktivácie jednotlivých výstupných jednotiek so správnymi hodnotami. Váhy siete sa potom nastavujú tak, aby sa čo najviac zmenšil rozdiel medzi výstupom siete a správnym vopred známym výstupom. Každá iterácia tejto procedúry je tréningovým cyklom, prebehnutie tréningových cyklov cez všetky vzory je tréningovou epochou. Takýchto epoch je na naučenie siete typicky potrebné rádovo vyše desať tisíc.

Okrem najjednoduchšej gradientovej metódy najväčšieho spádu sa často používajú aj iné metódy, napr. metóda spriahnutých gradientov, Levenbergova-Marquardtova metóda a pod. [34]. No najjednoduchšia metóda najväčšieho spádu sa ešte stále používa najčastejšie. Ostatné metódy sú síce rýchlejšie, no môžu sa jednoduchšie "preučiť" alebo sklznúť do lokálneho minima. "Preučenie" u neurónových sietí znamená niečo podobné ako použitie príliš veľkého stupňa polynómu u polynomiálnej regresie. Neurónová sieť sa v takom prípade naučí takmer perfektne produkovať požadované výstupy zo zadaných vstupov tréningovej množiny, no pre testovaciu množinu zlyháva. Podobný problém často nastáva aj pri použití evolučných optimalizačných algoritmov na nastavenie parametrov neurónovej siete.

10.2.3 Algoritmizácia neurónovej siete s dopredným šírením

Ďalej naznačíme základné princípy algoritmizácie neurónových sietí s dopredným šírením, ktoré obsahujú skryté neuróny. Pre jednoduchosť budeme uvažovať 3-vrstvovú neurónovú sieť, ktorá obsahuje jednu vrstvu skrytých neurónov, pričom neuróny zo susedných vrstiev sú prepojené všetkými možnými spôsobmi (pozri obr. 10.6).

Aktivity skrytých a výstupných neurónov sú určené vzťahmi (pozri vzťahy (10.10a-b))

$$z_i = t \left(\sum_{j=1}^n v_{ij} x_j + \Theta_i \right) \quad (\text{pre } i = 1, 2, \dots, p) \quad (10.15a)$$

$$y_i = t \left(\sum_{j=1}^p w_{ij} z_j + \vartheta_i \right) \quad (\text{pre } i = 1, 2, \dots, m) \quad (10.15b)$$

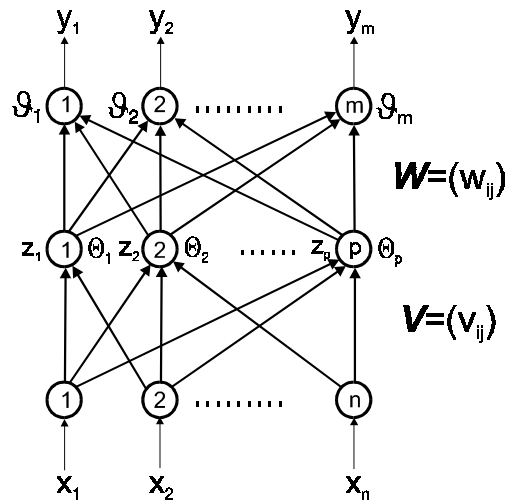
kde $t(\xi)$ je sigmoida určená pomocou (10.11) (s parametrami $A=0$ a $B=1$)

$$t(\xi) = \frac{1}{1 + e^{-\xi}} \quad (10.16)$$

Grafický priebeh tejto funkcie je znázornený na obr. 10.5, graf 1.

Výpočet aktivít neurónov pre dané váhové a prahové koeficienty sa nazýva aktívna fáza neurónovej siete. Tieto aktivity pre danú neurónovú sieť sa vypočítajú jednoduchým rekurentným postupom: Predpokladajme, že vstupné aktivity $x_1, x_2, \dots, x_n$ (deskriptory klasifikovaného objektu) sú známe, potom pomocou (10.15a) zostrojíme aktivity skrytých neurónov $z_1, z_2, \dots, z_p$. Následne, pomocou (10.15b) zostrojíme aktivity výstupných neurónov $y_1, y_2, \dots, y_m$. Uvedený rekurentný spôsob výpočtu aktivít postupuje neurónovou sieťou zdola

nahor. Táto skutočnosť sa odráža v názve týchto sietí ako neurónových sietí s dopredným šírením signálu. Algoritmizácia tohto postupu je formou pascalovského pseudokódu uvedená v algoritme 10.1.



Obrázok 10.6. Trojvrstvová neurónová sieť s dopredným šírením. Vstupná vrstva obsahuje n vstupných neurónov s aktivitami $x_1, x_2, \dots, x_n$. Skrytá vrstva obsahuje p skrytých neurónov, ich aktivity resp. prahové koeficienty sú označené $z_1, z_2, \dots, z_p$, resp. $\theta_1, \theta_2, \dots, \theta_p$. Výstupná vrstva obsahuje m výstupných neurónov, ich aktivity resp. prahové koeficienty sú označené $y_1, y_2, \dots, y_m$ resp. $\vartheta_1, \vartheta_2, \dots, \vartheta_m$. Váhové koeficienty medzi vstupnou a skrytou vrstvou, resp. skrytou a výstupnou vrstvou tvoria maticu $V=(v_{ij})$, resp. maticu $W=(w_{ij})$.

```

procedure activities(input:  $\Theta, V, \vartheta, W, x$ ; output:  $z, y$ );
begin for  $i:=1$  to  $p$  do
    begin  $\xi:=\Theta[i]$ ;
        for  $j:=1$  to  $n$  do  $\xi:=\xi+v[i,j]*x[j]$ ;
         $z[i]:=t(\xi)$ ;
    end;
    for  $i:=1$  to  $m$  do
        begin  $\xi:=\vartheta[i]$ ;
            for  $j:=1$  to  $p$  do  $\xi:=\xi+w[i,j]*z[j]$ ;
             $y[i]:=t(\xi)$ ;
        end;
    end;
end;

```

Algoritmus 10.1. Algoritmizácia v pascalovskom pseudokóde aktívnej fázy neurónovej siete s dopredným šírením, ktorá obsahuje jednu vrstvu skrytých neurónov. Vstupnými parametrami procedúry activities sú vstupné aktivity a váhové a prahové koeficienty, výstupnými parametrami sú skryté a výstupné aktivity. Reálna funkcia $t(\xi)$ je prechodová funkcia definovaná vzťahom (10.16).

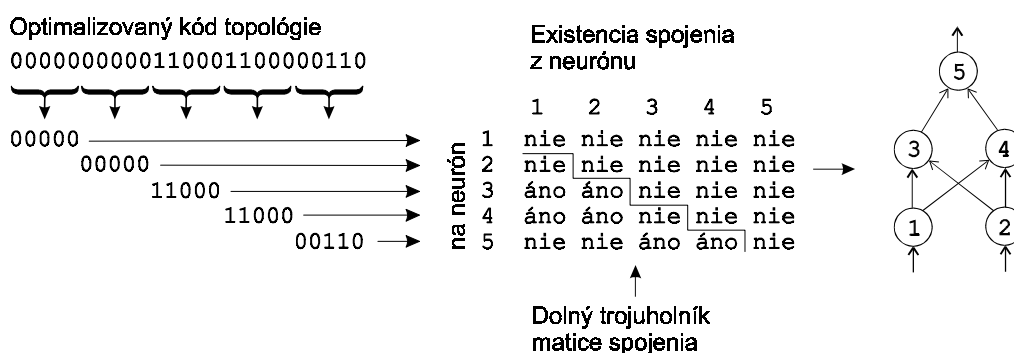
10.2.4 Optimalizácia neurónových sietí s dopredným šírením

Evolučné algoritmy sa používajú pri stochastickej optimalizácii neurónových sietí. Tieto algoritmy využívajú informáciu z predchádzajúceho riešenia pri návrhu novej, lepšej siete.

Snažíme sa pritom, aby súčet štvorcov odchýlok výstupov z neurónovej siete od vopred zadanych hodnôt bol čo najmenší. Tam, kde sa budeme zaoberať optimalizáciou topológie dopredných viacvrstvových neurónových sietí, bude učiacu stratégiu tvoriť ten najznámejší algoritmus - „*backpropagation*“, teda metóda adaptácie pomocou spätného šírenia chýb. Toto zjednodušenie je oprávnené z toho dôvodu, že optimalizácia pomocou stochastických evolučných algoritmov sa v prevažnej väčšine používa práve pre tento základný typ neurónovej siete.

V optimalizácii neurónových sietí sa vyskytujú dve základné úlohy: optimalizácia topológie a optimalizácia váh.

■ **Optimalizácia topológie neurónovej siete** spočíva v určení počtu skrytých vrstiev, počtu neurónov v týchto vrstvách, existencie spojení medzi nimi (obr. 10.7) [35,36], prípadne aj parametrov prechodovej funkcie neurónov alebo parametrov na učenie siete pomocou spätného šírenia.



Obrázok 10.7. Príklad zakódovania topológie spojenia piatich neurónov. Optimalizácia topológie neurónovej siete je pretransformovaná na optimalizáciu úvodného binárneho reťazca, pričom prvé dva neuróny sú definované ako vstupné a posledný piaty neurón je výstupný (podľa [35]).

Uvedený príklad ukazuje iba zlomok z toho, čo všetko sa dá optimalizovať. Aj tak je tento príklad príliš zjednodušený. Pokiaľ by sme nechceli zakaždým kontrolovať, či je sieť skutočne „feed forward“, teda či náhodou niektoré spojenia netvoria cyklus, potom je najjednoduchšie vyplniť iba dolný trojuholník matice spojení. V každom prípade však treba pred učením vyradovať neprijateľné architektúry (napr. také, kde nevedie cesta zo vstupu do výstupu) a vyradovať tiež vnútorné neuróny a ich spojenia, ktoré neležia na žiadnej ceste zo vstupu do výstupu.

Tento typ zakódovania topológie sa nazýva „nízkoúrovňovým“, pretože kóduje priamo topológiu. Pri väčších sieťach je možné nekódovať priamo spojenia, ale je potrebné len percento pravdepodobnosti spojenia dvojice neurónov z dvoch skupín neurónov [37]. U ešte zložitejších typov zakódovania, tzv. „vysokoúrovňových“, môže byť architektúra siete špecifikovaná pravidlami rastu alebo vetami formálneho jazyka. Tento typ je vhodnejší pre optimalizáciu rozsiahlejších sietí (na ktorú v našich podmienkach však väčšinou nemáme dostatočne rýchly hardvér). Vysokoúrovňové zakódovania väčšinou predstavuje tzv. gramatické kódovanie, kedy evolučné algoritmy nevyvíjajú priamo architektúru siete, ale skôr pravidlá formálnych gramatík, ktoré sa potom používajú pre generovanie vlastnej topológie siete.

Tak „nízkoúrovňové“, ako aj „vysokoúrovňové“ zakódovanie musí byť ohodnotené, teda zo zakódovania musí byť vyrobená sieť, ktorá sa inicializuje malými náhodnými hodnotami váh a potom optimalizuje väčšinou štandardným algoritmom „backpropagation“. Správne by sme asi mali pre každé takéto zakódovanie urobiť niekoľko štartov s rozdielnymi počiatočnými váhami a potom zobrať buď najlepší výsledok, alebo priemer pre ohodnotenie

“chromozómu” opisujúceho zakódovanie siete. V praxi sa však robí pre každú sieť iba jeden štart, a to sa ešte väčšinou snažíme čo najviac redukovať počet cyklov algoritmu backpropagation.

Gramatické zakódovanie môžeme predstaviť prácou Hiroaki Kitana [38]. Ten argumentuje tým, že počet možných spojení neurónov rastie kvadraticky s počtom neurónov, čo za použitia priameho kódovania môže pre veľké siete predstavovať problém. Gramatické kódovanie je schopné efektívne zakódovať aj niektoré opakujúce sa alebo zahniezdené podštruktúry siete. Pravidlá formálnych gramatík sa optimalizujú genetickými algoritmi, no po každej zmene musí nasledovať vytvorenie siete podľa nových pravidiel a jej ohodnotenie. Toto ohodnotenie potom slúži ako podklad pre funkciu fitness daného súboru pravidiel gramatiky.

Gramatiku tak možno nazvať “genotypom”, analógiou genetického kódu živočíchov alebo rastlín, zatiaľ čo samotná sieť sa môže stotožniť s fenotypom, teda príkladom živočicha alebo rastliny vytvorenej podľa genetického kódu.

Vytváranie “slov” podľa určitej gramatiky sa robí postupným prepisovaním reťazcov znakov podľa pravidiel gramatiky. Generujúca gramatika obsahuje dve disjunktné konečné abecedy, počiatočný symbol S a prepisovacie pravidlá. Prvá abeceda sa volá množina neterminálov (premenných), druhá abeceda množina terminálov; prepisovacie pravidlá pre bezkontextovú gramatiku prepisujú vždy jednu premennú na reťazec vytvorený z množiny premenných a terminálov. Výsledným slovom je potom podľa týchto pravidiel vytvorený reťazec, ktorý obsahuje už iba terminály. Najjednoduchším takým prípadom by bola abeceda premenných tvorená jediným symbolom S , abeceda terminálov tvorená dvoma symbolmi 0 a 1 a produkčné pravidlá $S \rightarrow 0S1$, $S \rightarrow 01$.

Z počiatočného znaku S tak možno vytvoriť každé “slovo” obsahujúce v prvej polovici samé nuly a v druhej samé jednotky, napr.

$S \rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 0000S1111 \Rightarrow 00000S11111 \Rightarrow 000000S111111 \Rightarrow 00000001111111$

Kitanov produkčný systém obsahoval abecedu premenných $A-Z$ a abecedu šesťnástich terminálov $a-p$. Gramatika generujúca grafy potom na pravej strane obsahuje namiesto jednorozmerného reťazca maticu 2×2 . Šesťnásť terminálov potom jednoznačne odpovedá všetkým možným maticiam 2×2 zloženým z núl a jednotiek. Z počiatočného symbolu sa pomocou pravidiel vygeneruje matica terminálov, z tej matice susednosti a z nej potom orientovaný graf. Pravidlá sú pritom dané chromozómom (viď obr. 10.8). Tri neuróny zodpovedajúce predposledným trom riadkom v matici nie sú spojené s ničím, preto nie sú do grafu zakreslené.

V zjednodušenej Kitanovej verzii je predpísaný iba prvý symbol S , ostatné sa môžu meniť (mutovať). Keď sa niektorá z premenných $A-Z$ objaví v pravidlách viackrát, berie sa pri konštrukcii matice susednosti do úvahy iba prvé z pravidiel. Na pravej strane sa v tejto konkrétnej gramatike objavujú iba terminálne symboly, takže môžeme skonštruovať iba maticu 8×8 . V prípade, že daný graf má aj rekurentné spojenia, tieto sa nebudú brať do úvahy a budú sa brať iba grafy s vopred danými počtami vstupných a výstupných vrcholov, ktoré zodpovedajú riešenému problému. Výsledné grafy sa potom učia pomocou “backpropagation” pre tréningovú množinu a ohodnotia súčtom chýb pre vzory z testovacej množiny. Kitano použil klasickú selekciu úmernú fitness pomocou “rulety”, kríženie s rozsekaním na niekoľko podreťazcov a následnou výmenou týchto podreťazcov a ako mutáciu použil náhradu náhodne zvoleného symbolu symbolmi z abecied $A-Z$ a $a-p$. Pravdepodobnosť mutácie potomkov bola nepriamo úmerná Hammingovej vzdialenosti (počtu rozdielnych prvkov na zodpovedajúcich si pozíciách) pre reťazce rodičov. Kitanove výsledky boli lepšie pre gramatické kódovanie ako pre priame kódovanie, no nie sú veľmi presvedčivé, pretože použité príklady boli príliš jednoduché. Ďalšie rozšírenie tohto prístupu, kde je integrovaný vývoj architektúry spolu s nastavením váh, možno nájsť v [40].

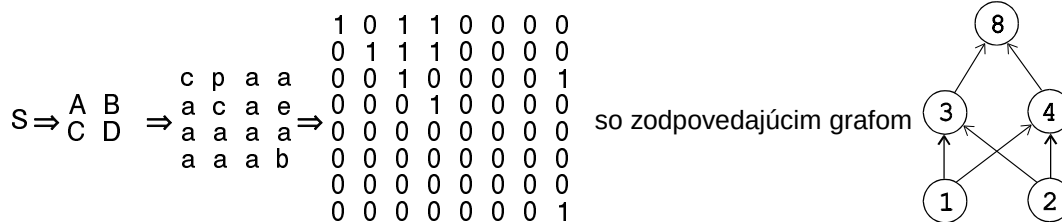
Príklad pravidiel generujúcej gramatiky

$$S \rightarrow \begin{matrix} A & B \\ C & D \end{matrix} \quad A \rightarrow \begin{matrix} c & p \\ a & c \end{matrix} \quad B \rightarrow \begin{matrix} a & a \\ a & e \end{matrix} \quad C \rightarrow \begin{matrix} a & a \\ a & a \end{matrix} \quad D \rightarrow \begin{matrix} a & a \\ a & b \end{matrix},$$

kde prepis terminálov (už nepatriacich do gramatiky) bude

$$a \rightarrow \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix} \quad b \rightarrow \begin{matrix} 0 & 0 \\ 0 & 1 \end{matrix} \quad c \rightarrow \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \quad e \rightarrow \begin{matrix} 0 & 1 \\ 0 & 1 \end{matrix} \quad p \rightarrow \begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix}$$

a s pomocou týchto pravidiel vznikne nakoniec matica susednosti



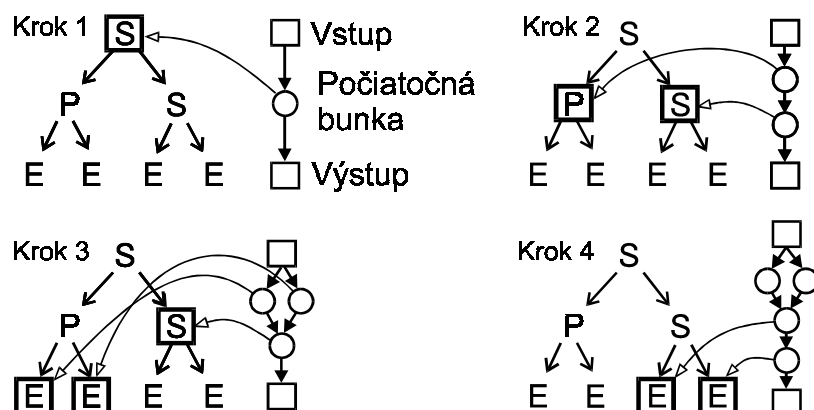
Chromozóm zodpovedajúci týmto pravidlám

S	A	B	C	D	A	c	p	a	c	B	a	a	a	e	C	a	a	a	a	...
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

Obrázok 10.8. Príklad pravidiel generujúcej gramatiky podľa Kitana [38].

O zložitejšie zakódovanie sietí sa pokúsili Grunau, Bellew a Whitley [39,41,42], ktorí používajú na zakódovanie siete tzv. celulárne kódovanie. Takto bolo vyvinuté napr. riadiace zariadenie pre šesťnohého robota alebo balansovanie "tyče". Základom celulárneho kódovania je tzv. gramatický strom. Každá bunka ukazuje na určité miesto v tomto strome a podľa toho, čo na tejto pozícii je, sa bunka rozdelí na dve bunky, zmení svoje vnútorné parametre (napr. prahový faktor neurónu), alebo premiestni svoj ukazovateľ na inú pozíciu v strome, čo umožní robiť rekúziu (treba si pamätať hĺbku rekúzie). Vývoj začína od jednej počiatočnej bunky ukazujúcej na vrchol stromu. Táto bunka je spojená so vstupom a výstupom. V *sekvenčnom* rozdelení označenom S sa rodičovská bunka rozdelí na dve dcérske bunky tak, že prvá dcérska bunka zdedí všetky vstupy rodičovskej bunky a druhá dcérska bunka zdedí všetky výstupy. Pritom spojenie povedie z prvej dcérskej bunky na druhú. Na obrázku 10.9 je druhá dcérska bunka vždy umiestnená pod prvou. Pri *paralelnom* rozdelení označenom P sa rodičovská bunka rozdelí na dve nespojené dcérske bunky, z ktorých každá zdedí vstupy aj výstupy rodičovskej bunky. Na obrázku 10.9 sú tieto bunky umiestnené vedľa seba. Uzly S aj P zároveň vedľa seba znamenajú, že prvá dcérska bunka posunie svoj ukazovateľ do uzla vľavo dole, zatiaľ čo druhá dcérska bunka bude ukazovať do uzla vpravo dole. Symbol E je potom symbol ukončenia. Je možné aj označenie uzla R, kedy sa ukazovateľ presunie na niektorý predchádzajúci uzol, čím je možné zaviesť rekurzívnosť. Podobne sa dá zdefinovať aj analógia Automaticky Definovaných Funkcií (ADF) z [43] pri genetickom programovaní. Gramatické stromy sa podobne ako funkcie pri Kozovom genetickom programovaní [44] optimalizujú pomocou genetických algoritmov, kedy ohodnotenie chromozómu (gramatického stromu) je odvodené z účelovej (chybovej) funkcie zodpovedajúcej neurónovej sieti.

Stochastické optimalizačné algoritmy sú v podstate jediným systematickým prístupom k optimalizácii topológie siete. Žiaľ, tieto algoritmy potrebujú vygenerovať a ohodnotiť rádovo tisícky (alebo viac) možných topológií neurónových sietí, aby dospeli k dobrému výsledku. Vzhľadom na to, že ohodnotenie každej topológie predstavuje vlastne naučenie jednej neurónovej siete pomocou tisícok iterácií spätného šírenia (alebo tisícok generácií genetických algoritmov), ide o výpočtovo veľmi náročnú úlohu.



Obrázok 10.9. Vytváranie siete z celulórneho kódovania. Naľavo je vždy tzv. gramatický strom, napravo sieť s neurónmi (bunkami) označenými krúžkom. Šípky od neurónov na P, S, a E v strome určujú ďalšiu akciu neurónu. S znamená vertikálne rozdelenie neurónu na dva nové, P horizontálne rozdelenie a E ukončenie práce s neurónom (podľa Gruaua [39]).

Kvôli výpočtovej zložitosti sa namiesto systematického prístupu k návrhu topológie stále bežne používa skôr intuícia a príklady optimalizácie topológie neurónovej siete existujú väčšinou iba pre jednoduchšie problémy. Pri návrhoch sietí sa bežne používajú heuristiky ako “pre zložitejšie problémy je potrebné viac skrytých jednotiek” a metódy “pokusu a omylu”. Návrh siete pre zložitejšie problémy pritom často sprevádza neprípustné zjednodušovanie optimalizačných algoritmov. Ich hlavná sila, ktorou je prehľadávanie mnohorozmerného priestoru s viacerými minimami a schopnosť dostať sa z lokálnych miním a skončiť v globálnom minime, sa potom znižuje. Zvyšuje sa tým pravdepodobnosť, že topológia skončí niekde v lokálnom minime – teda nebude ideálna. Žiaľ, výpočtové nároky týchto metód neumožňujú lepšie prehľadávanie možných topológií.

Okrem počtu vrstiev, neurónov a ich spojení môže byť sieť optimalizovaná na dosiahnutú presnosť a na veľa ďalších vlastností. Ani optimalizovaná funkcia nemusí byť iba súčtom odchýlok chýb predpovedí od už známych faktov. Sieť sa môže optimalizovať napr. na rýchlosť učenia alebo na malý počet spojení medzi neurónmi alebo na malý počet vnútorných neurónov. Existuje aj špeciálne zakódovanie potenciálnej siete, aby sa ľahko menili počty neurónov a vrstiev (viď [37]), ale tento postup má zasa iné nedostatky a nezakladá sa na dostatočne hlbokéj teórii, preto ho tu nebudeme uvádzať.

Atypickým príkladom je práca Davida Chalmersa [45], ktorý optimalizoval kompletne prepojenú sieť iba so vstupnou a výstupnou vrstvou neurónov. Keď označíme aktivity vstupných neurónov x_i , výstupných neurónov y_j , ideálne výsledné aktivity pre daný vstup $\hat{y}_j$ a váhu spojenia neurónov i a j ako w_{ij} , potom existuje známe Widrowovo-Hoffovo pravidlo “delta” pre najvhodnejšiu zmenu váh siete

$$\Delta w_{ij} = \eta(\hat{y}_j - y_j)x_i, \quad (10.17)$$

kde η je rýchlosť učenia (číslo z intervalu (0,1)). Chalmers sa pokúsil nahradiť toto pravidlo lineárnou funkciou všetkých premenných a ich párových násobkov.

$$\Delta w_{ij} = k_0(k_1 w_{ij} + k_2 x_i + k_3 y_j + k_4 \hat{y}_j + k_5 w_{ij} x_i + k_6 w_{ij} y_j + k_7 w_{ij} \hat{y}_j + k_8 x_i y_j + k_9 x_i \hat{y}_j + k_{10} y_j \hat{y}_j) \quad (10.18)$$

Chalmers sa pokúsil optimalizovať konštanty k_0 - k_{10} . Približne v jednom z desiatich pokusov skutočne dostal Widrowovo-Hoffovo pravidlo “delta”. Samozrejme, získanie účinnejšieho

algoritmu, ako je backpropagation, nie je pravdepodobné, ale uvedený prístup by mohol slúžiť pre učiace pravidlá pri učení bez učiteľa alebo pre rekurentné siete.

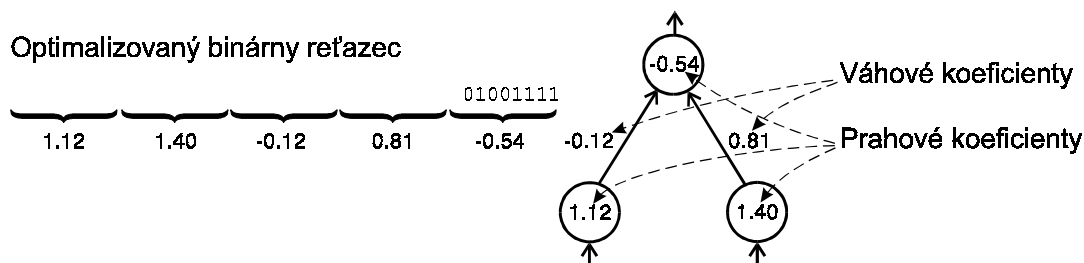
Jednotlivcami v evolučnom algoritme môžu byť aj jednotlivé neuróny [46].

Úloha 10.1. Naprogramujte sieť uvedenú na obr. 10.7 a optimalizujte pomocou genetického algoritmu váhy a prahové parametre tejto siete tak, aby s čo najmenšou chybou produkovala tabuľku 10.1, teda XOR, kde z numerických dôvodov namiesto 0 a 1 budeme brať 0.1 a 0.9.

Tabuľka 10.1. Upravená tréningová množina pre XOR

vstup do neurónu		výstup z neurónu
č. 1	č. 2	č. 5
0	0	0.1
0	1	0.9
1	0	0.9
1	1	0.1

■ **Optimalizácia váhových a prahových koeficientov** spojení v neurónových sieťach (obr. 10.10), ktorá nahradzuje metódu spätného šírenia, je ďalšou úlohou často riešenou pomocou evolučných metód [47]. Evolučné metódy však nie sú pre túto optimalizáciu najtypickejšie, pretože klasická metóda spätného šírenia poskytuje dobré výsledky a je výpočtovo menej náročná. Klasické algoritmy optimalizujúce súčet štvorcov odchýlok založené na derivácii "chybovej" (účelovej) funkcie automaticky končia v najbližšom lokálnom optime. Našťastie, hyperplocha optimalizovanej funkcie väčšinou nemá príliš mnoho lokálnych miním a výsledky metódy spätného šírenia sú väčšinou prijateľné. Pokiaľ sa optimalizácia nedarí ani po viacerých spusteniach s rôznymi náhodne generovanými počiatočnými váhami, stačí často pridať niekoľko skrytých neurónov. Keď však chceme vytvoriť efektívnu sieť s minimom vnútorných neurónov a spojov, sú evolučné algoritmy nenahraditeľné. Používajú sa tiež pri problémoch, kde nie sú k dispozícii korektné odpovede jednotlivých výstupných jednotiek siete na daný výstup, čo sa vyskytuje väčšinou pri problémoch riadenia [48,49] alebo navigácie robotov v neznámom prostredí. (V týchto prípadoch je však potrebné používať vždy iné štartovacie stavy, aby boli uniformne rozložené. Treba si tiež dávať pozor nielen na rýchlosť učenia, ale aj na schopnosť zovšeobecňovania, aby sieť nefungovala iba na zopár naučených príkladoch.) Evolučné metódy dokážu relatívne rýchlo vyhľadať váhové a prahové koeficienty blízke optimálnym, trvá im však dlho prechod od takmer optimálnych váh k optimálnym váham [50-53]. Tento nedostatok sa často nahradzuje spojením evolučného algoritmu s metódou spätného šírenia — evolučný algoritmus nájde približné hodnoty optima a spätné šírenie dokončí optimalizáciu do globálneho optima [54,55].



Obrázok 10.10. Zobrazenie prevodu binárneho reťazca na váhové a prahové koeficienty, teda reálne čísla. Namiesto vektora tvoreného reálnymi číslami je potom možné optimalizovať binárny reťazec.

Jednou z hlavných podmienok úspešného použitia evolučných algoritmov je rýchlosť výpočtu hodnôt účelovej funkcie v danom bode. Špeciálne táto posledná podmienka podstatne limituje úspešné použitie stochastických optimalizačných metód na optimalizáciu neurónových sietí; ich relatívna jednoduchosť v porovnaní s gradientovými metódami je "kompenzovaná" náročnosťou na výpočtový čas.

Pre optimalizáciu topológie neurónových sietí sa zatiaľ prakticky výhradne používajú genetické algoritmy, pre optimalizáciu váhových a prahových koeficientov sa používajú genetické algoritmy a simulované žihanie.

Nevýhodou genetických algoritmov pri optimalizácii topológie neurónových sietí sú problémy so zmysluplnou výmenou informácií pri krížení. Konkrétne ide o váhy v neurónovej sieti. Povedzme, že máme štyri vstupné a štyri skryté neuróny. Podarilo sa nám vygenerovať sieť, v ktorej sú spojené vstupné neuróny nenulovými váhami iba s prvými dvoma skrytými neurónmi, teda druhé dva skryté neuróny môžeme zanedbať. Vzhľadom na to, že sieť je v podstate symetrická, môže existovať rovnako dobrá topológia zanedbávajúca prvé dva skryté neuróny a používajúca iba druhé dva skryté neuróny. Pri krížení potom môže nastať situácia, že prvý potomok bude používať všetky štyri skryté neuróny a druhý potomok bude mať všetky váhy nulové. Existujú rôzne spôsoby, ako predchádzať tomuto problému krížením iba viacmenej podobných jedincov - sietí, ale žiaden z nich sa nedá považovať za uspokojivý. Pre n skrytých neurónov totiž existuje $n!$ permutácií ich postavení a teda $n!$ ekvivalentných sietí. Okrem tohto druhu symetrie existuje i symetria pri váhach skrytých neurónov. Keď je prechodová funkcia nepárna (čo väčšinou je), potom môžeme nahradiť všetky znamienka vstupných a výstupných váh skrytého neurónu opačnými znamienkami a výstupy siete sa nezmenia. Pre n skrytých neurónov tak máme 2^n štruktúrne odlišných, ale funkčne identických sietí generovaných takýmto prehodením znamienok. Pri krížení týchto sietí potom dochádza k nelogickosti, pretože sa môže ľahko stať, že polovicu váh neurónu vezmeme z jednej siete, polovicu z druhej siete (kde boli opačné znamienka) a výsledkom je niečo, čo nebolo ani v jednej sieti. Dochádza tak skôr k mutácii ako k výmene informácií. Celkovo je teda priestor váh $2^n \cdot n!$ -krát väčší, ako by v skutočnosti mal byť. Príkladom pokusu o elimináciu tejto redundancie je kríženie váh s rovnakým znamienkom u dvojíc neurónov s rovnakým počtom kladných váh a záporných váh. Počet zodpovedajúcich dvojíc sa dá zvýšiť prípadným prehodením všetkých znamienok váh pri neuróne.

Tieto problémy sa väčšinou znižujú zvýšením pravdepodobnosti mutácie a znížením pravdepodobnosti kríženia pri genetickom algoritme. Dá sa tiež použiť evolučné programovanie, čo je v podstate genetický algoritmus bez kríženia, prípadne s gaussovskými mutáciami (viď kap. 8) [56].

10.3 Optimalizácia pomocou neurónových sietí

Algoritmy inšpirované teóriou neurónových sietí môžeme použiť nielen na predikciu a klasifikáciu [1-6], ale aj na *optimalizačné problémy*. Napríklad na neurónovú sieť Hopfieldovho typu alebo na Boltzmannov stroj sa môžeme pozeráť ako na masívne paralelné algoritmy, ktorých cieľom je minimalizácia energie systému. Energia systému tu vlastne zastupuje účelovú (cenovú, objektívnu, ohodnocovaciu, kritériálnu, chybovú) funkciu, ktorej hodnota je optimalizovaná. U týchto sietí vlastne nejde o principiálne nové optimalizačné algoritmy – na optimalizáciu sa používa u Boltzmannovho stroja simulované žihanie, u Hopfieldovej siete gradientová metóda. Podstatná je v týchto príkladoch formulácia problému, ktorá je zakomponovaná do topológie siete a do funkcie energie systému. Na každý typ problému treba sformulovať mierne odlišný predpis na zostrojenie siete a funkcie energie. Vo všeobecnosti je funkcia energie zostavená z dvoch členov:

$$E = \text{"hodnota vlastnej optimalizovanej funkcie"} + \text{"pokuta za prekročenie ohraničujúcich podmienok"} \quad (10.19)$$

Myšlienka Hopfieldových sietí a Boltzmannových strojov je založená na Isingovom modeli spinového skla [57,58], ktorý sa použil na skúmanie kolektívnych vlastností fyzikálnych systémov zložených z veľkého množstva jednoduchých prvkov.

Netypickou možnosťou je využitie Kohonenovej SamoOrganizujúcej sa Mapy (SOM) na riešenie problému obchodného cestujúceho. Táto sieť sa však nevyužíva na optimalizáciu iných problémov a ani jej použitie pre problém obchodného cestujúceho nie je typické. Preto sa jej budeme venovať až na konci tejto kapitoly.

Pomocou Hopfieldových sietí a Boltzmannových strojov sa riešia nielen kombinatorické optimalizačné problémy, ale aj spracovanie obrazu (image processing), čiže rekonštrukcia objektu zo zašumeného alebo rozmazaného obrazu. Treba zdôrazniť, že v prípade spracovania obrazu nejde o vybavenie si podobného obrazu z pamäti (aj také problémy sa typicky riešia Hopfieldovou neurónovou sieťou), ale o interpoláciu povrchov objektov [22], detekciu hrán [1,23], rekonštrukciu trojrozmerného tvaru objektov z ich tieňovania alebo binokulárnej disparity [20,21,59], rekonštrukciu tvaru objektov na základe farby alebo pohybu a pod. Pretože spracovanie obrazu nie je typickým optimalizačným problémom, aj keď sa k nim v súvislosti s neurónovými sieťami niekedy zaraďuje a samotná problematika je dosť špecializovaná, nebudeme sa spracovaním obrazu ďalej zaoberať. Problematike spracovania obrazu je podrobnejšie venovaná kniha [60] (bez použitia neurónových sietí), alebo kniha [61,9] (viacvrstvové siete s dopredným šírením aj iné).

Neurónové siete sa dajú použiť aj na minimalizáciu lineárnych alebo kvadratických funkcií s obmedzeniami premenných vyjadrenými pomocou lineárnych rovníc a nerovníc, alebo aj pre nelineárne programovanie, kedy optimalizované funkcie a ohraničujúce podmienky pre premenné sú nelineárne. V týchto prípadoch sa však väčšinou vyžaduje, aby optimalizované funkcie boli hladké a diferencovateľné. Pokiaľ nie sú, musíme problémy na optimalizáciu hladkých diferencovateľných funkcií pretransformovať. Podrobne je táto tematika uvedená v knihe [62], tu sa týmito prístupmi nebudeme zaoberať, lebo sú pre svoju neefektívnosť zaujímavé (aspoň zatiaľ) iba z teoretického hľadiska. Efektívnymi by sa tieto prístupy mohli stať iba pri hardvérovej implementácii.

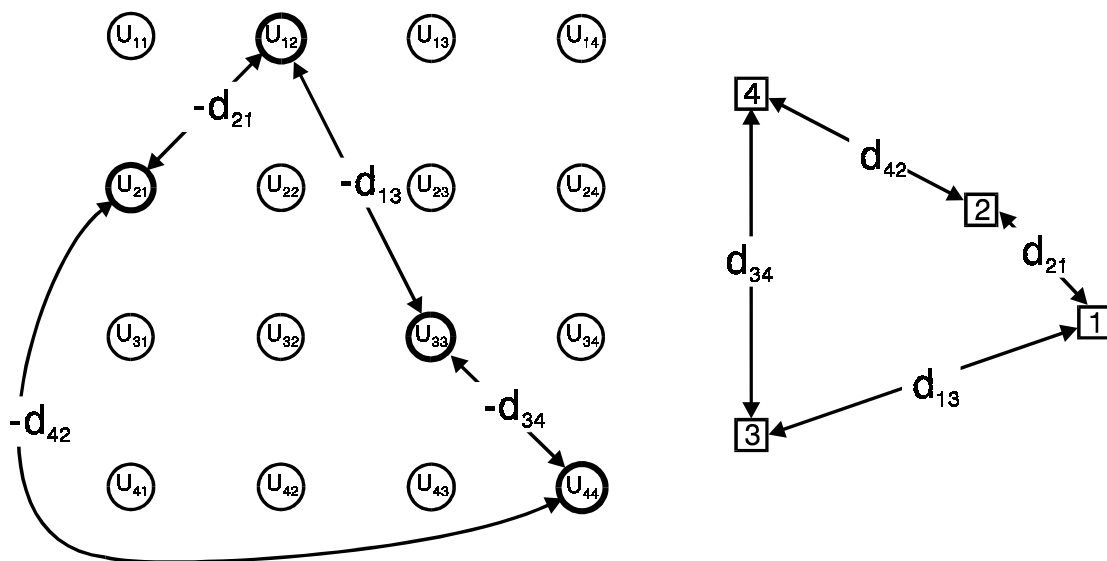
10.3.1 Kombinatorická optimalizácia pomocou Boltzmannovho stroja a Hopfieldovej siete

U optimalizačných problémov je najpodstatnejšie zostaviť sieť a na jej základe definovať optimalizovanú funkciu tak, aby optimálny stav siete zodpovedal optimálnemu riešeniu. Základným problémom je napríklad optimalizácia permutácie. Problémom tu je už dostať samotnú permutáciu, bez ohľadu na jej ďalšie ohodnotenie. Tento problém sa rieši na obrázku 10.11, kde optimalizácia znamená zapínanie a vypínanie uzlov siete - pravouhlej mriežky.

Ide vlastne o generovanie permutačnej matice, kde zapnutý uzol predstavuje jednotku, vypnutý nulu a v každom riadku alebo stĺpci by mala byť práve jedna jednotka. Uzly sú ohodnotené hranami a slučkami, pričom hrany spájajú všetky uzly v rovnakom riadku i všetky uzly v rovnakom stĺpci, iba šikmé hrany sa nevyskytujú. Hrany sú ohodnotené záporným číslom, slučky kladným číslom. Minimalizuje sa tu súčet

$$E = \sum_{x,i} \sum_{y,j} u_{xi} u_{yj} w_{xi,yj} \quad (10.20)$$

kde $u_{xi} = 0$, keď je prvok U_{xi} vypnutý; $u_{xi} = 1$, keď je prvok U_{xi} zapnutý; $w_{xi,yj}$ znamená váhu spojenia medzi dvoma prvkami (alebo slučku, keď $xi=yj$). Počet prvkov v permutácii je n .

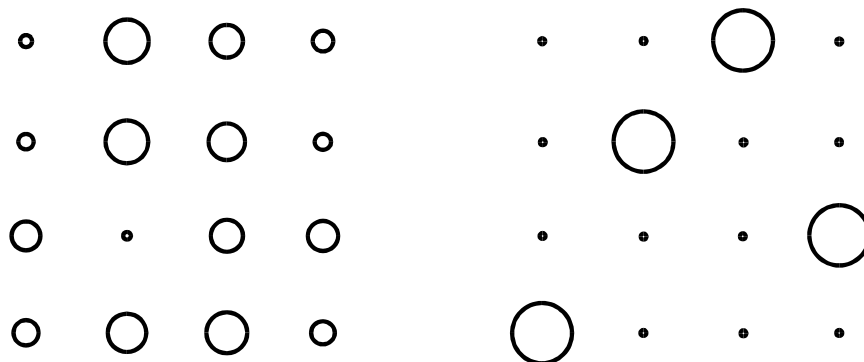


Obrázok 10.11. Riešenie problému obchodného cestujúceho. Na ľavej strane je uvedená sieť so zvýraznenými vybranými uzlami, ktoré sú spojené ďalšími váhami tvorenými zápornou vzdialenosťou medzi zodpovedajúcimi mestami určenými prvými indexami uzlov. Kvôli prehľadnosti nie sú uvedené váhy b a p spôsobujúce uprednostnenie výberu iba jedného uzla na každý riadok a stĺpec, čím definujú permutáciu. Na pravej strane je potom reálne rozmiestnenie týchto miest, ktoré sú označené štvorčekmi. Indexy vo vnútri štvorčekov odpovedajú prvým indexom (teda riadkom) zvýraznených uzlov siete na ľavej strane, ich postupnosť odpovedá hľadanej permutácii.

Váhy sú nastavené tak, že medzi uzlami v riadku alebo stĺpci sú záporné $-p$, aby súčasne neboli "zapnuté" prvky v rovnakom riadku alebo stĺpci, a zároveň slučky majú kladné ohodnotenie b , aby bol zapojený vždy jeden prvok v každom riadku alebo stĺpci. Keďže b a p sú kladné čísla, musí platiť, že $p > b$, aby zapnutie druhého uzla v rovnakom riadku alebo stĺpci nezvýšilo hodnotu maximalizovanej funkcie E . Sčítavania prebiehajú cez všetky uzly. Týmto sa však dostávame iba k permutácii, ktorá môže vyjadrovať riešenie spĺňajúce základné obmedzenia. Ohodnotenie permutácie je ďalším problémom. Keď riešime napríklad problém obchodného cestujúceho hľadajúceho najkratšiu dráhu, ktorá prechádza práve raz cez každé mesto a vracia sa do počiatočného mesta, musíme minimalizovať súčet vzdialeností medzi mestami na dráhe. Pretože však používame maximalizáciu, označíme vzdialenosti medzi mestami opačným znamienkom. Pre mestá x a y označíme spojnice uzlov U_{xi} a U_{yj} váhou $-d_{xy}$. Tieto nové spojnice budú teda existovať medzi všetkými uzlami v susedných stĺpcoch a medzi všetkými uzlami prvého a posledného stĺpca. Konštanta b pritom musí byť väčšia ako dvojnásobok najdlhšej vzdialenosti medzi mestami, $b > 2d_{MAX}$. Funkcia (10.20) sa teda bude počítat cez všetky váhy, b , p aj d .

Týmto spôsobom sme však previedli iba optimalizáciu permutácie na optimalizáciu siete, no nepovedali sme, ako optimalizovať sieť. Pokiaľ použijeme ako mutáciu alebo perturbáciu "zapnutie" alebo "vypnutie" uzla a nový stav siete akceptujeme na základe simulovaného žihania (viď kapitola 6), máme Boltzmannov stroj zavedený pôvodne Hintonom a Sejnowskim [63].

Pokiaľ použijeme rýchle simulované žihanie [64] založené na pridaní šumu k hodnotám funkcie navrhovaných riešení a rýchlejšom poklese teploty a pokiaľ namiesto gaussovskej distribúcie použijeme Cauchyho distribúciu, dostaneme tak Cauchyho stroj [65].



Obrázok 10.12. Schematické znázornenie aktivácií Hopfieldovej-Tankovej siete pre riešenie problému obchodného cestujúceho pre 4 mestá. Podobne ako na obr. 10.11 číslo riadku predstavuje číslo mesta, číslo stĺpca predstavuje poradie mesta v permutácii. Veľkosť krúžku naznačuje veľkosť aktivácie daného uzla. Na ľavej strane sú náhodne vygenerované počiatočné aktivácie, ktorých súčet má byť počet miest, teda 4. Na pravej strane je táto sieť už skonvergovaná, po “zaokrúhlení”, kedy uzly s najväčšou aktiváciou berieme ako “zapnuté” a ostatné ako “vypnuté”, ako výsledok dostávame permutáciu 4,2,1,3. Z tohto výsledku vyplýva, že ako prvé bude mesto č. 4, z neho pôjde cesta cez mestá č. 2, 1, 3, až sa zasa nakoniec z mesta č. 3 vráti do mesta č. 4.

Ďalším obdobným prístupom k optimalizácii je Hopfieldova sieť. Jej zostavenie je podobné ako na obr. 10.11. Nebudeme teraz brať do úvahy váhy p a b , ale budeme ohodnocovať jednotlivé uzly (viď obr. 10.12). Toto ohodnotenie však nebude “zapnuté” alebo “vypnuté”; bude založené na súčte predchádzajúceho ohodnotenia, relaxačného členu spôsobujúceho stále znižovanie pravdepodobnosti zmeny stavu siete, váh zodpovedajúcich vzdialenostiam medzi mestami a ďalších členov závislých od hodnôt ostatných uzlov. Výstupný signál z uzla s aktivitou u_i sa podľa Hopfielda a Tanka [25,26] väčšinou transformuje sigmoidálnou funkciou g s parametrom α určujúcim strmosť a s výstupom medzi nulou a jednotkou,

$$v_i = g(u_i) = 0.5 (1 + \tanh(\alpha u_i)) \quad (10.21)$$

Keď vo všeobecnosti definujeme optimalizovanú funkciu (energiu) ako

$$E = \sum_{x,i=1}^{n,n} \sum_{y,j=1}^{n,n} v_{xi} v_{yj} w_{xi,yj} + \sum_{x,i=1}^{n,n} v_{xi} \theta_{xi} \quad (10.22)$$

potom podľa ďalej uvedeného algoritmu bude sieť konvergovať, dokiaľ derivácia tejto funkcie podľa času bude záporná. Premenná θ_{xi} označuje externý vstup do siete, čo v prípade problému obchodného cestujúceho znamená zahrnutie vzdialeností miest a podmienky, že celkový počet zapnutých uzlov by mal byť rovný počtu miest. Pre časovú konštantu systému τ sa aktivita jednotlivých neurónov v priebehu času bude meniť podľa diferenciálnej rovnice

$$\frac{d}{dt} u_{xi} = -\frac{u_{xi}}{\tau} + \sum_{y,j=1}^{n,n} w_{xi,yj} v_{yj} + \theta_{xi} \quad (10.23)$$

Vo väčšine obdobných rovníc sa vyskytuje iba jeden index, napr. i namiesto xi , tu sme urobili výnimku pre problém obchodného cestujúceho, kedy prvý index označuje mesto a druhý pozíciu v permutácii, teda koľké v poradí bude dané mesto navštívené. Hopfieldova-Tankova funkcia “energie” pre problém obchodného cestujúceho je vyjadrená pomocou štyroch častí. Prvá časť s konštantou A zodpovedá príspevkom dvojíc uzlov v jednom rade, pokiaľ sú obidva uzly “zapnuté”. Druhá časť s konštantou B zodpovedá príspevkom dvojíc

uzlov v jednom stĺpci, pokiaľ sú obidva uzly "zapnuté". Tretia časť s konštantou C závisí od uzlov samotných, malo by byť "zapnutých" nie viac, ani nie menej ako N uzlov. A konečne posledný člen s konštantou D vyjadruje vlastný cieľ problému obchodného cestujúceho, teda že vzdialenosť cesty by mala byť čo najmenšia.

$$E = \frac{A}{2} \sum_x \sum_i \sum_{j \neq i} v_{xi} v_{xj} + \frac{B}{2} \sum_i \sum_x \sum_{y \neq x} v_{xi} v_{yi} + \frac{C}{2} \left[\sum_x \sum_i v_{xi} - N \right]^2 + \frac{D}{2} \sum_x \sum_{y \neq x} \sum_i d_{xy} v_{xi} (v_{y,i+1} + v_{y,i-1}) \quad (10.24)$$

Parameter N v tejto rovnici sa väčšinou zafinuje o niečo väčší ako počet miest n . Polohy miest sa transformujú tak, aby sa vošli do štvorca 1.0×1.0 . Hopfield a Tank použili pre desať miest nasledujúce hodnoty konštánt: $A=B=500$, $C=200$, $D=500$, $N=15$, $\alpha=50$, $\tau=1$. Relaxačný term spôsobujúci konvergenciu hodnôt energie bol vo Wilsonovej a Pawleyovej adaptácii [27] Hopfieldovho algoritmu (viď Algoritmus 10.2) stanovený na $\Delta t=10^{-5}$. Podmienka ukončenia sa splnila pri nájdení platnej cesty, kedy pri aktivácii v väčšej ako 0.9 bol uzol považovaný za zapnutý a pri aktivácii menšej ako 0.1 za vypnutý. Keď bola sieť "zmrznutá", teda žiadna aktivácia sa nezmenila viac ako o 10^{-35} , alebo keď sa nenašla platná cesta v priebehu 1000 epoch, algoritmus bol zastavený.

Inicializuj aktivácie v jednotlivých uzloch náhodnými číslami tak, aby súčet všetkých aktivácií bol rovný počtu miest, $\sum_x \sum_i v_{xi} = n$;

```
while podmienka ukončenia nie je splnená
begin repeat 3-5  $n^2$  krát
  begin náhodne zvoľ uzol;
    zmeň aktivitu na uzle;
     $u_{xi}(new) = u_{xi}(old) + \Delta t [-u_{xi}(old) - A \sum_{j \neq i} v_{xj} - B \sum_{y \neq x} v_{yi} -$ 
       $- C \left\{ \sum_x \sum_j v_{xj} - N \right\} - D \sum_{y \neq x} d_{xy} (v_{y,i+1} + v_{y,i-1})]$ 
     $v_{xi} = 0.5 [1 + \tanh(\alpha u_{xi})]$ ;
  end;
end;
```

Algoritmus 10.2. Hopfieldov-Tankov algoritmus na riešenie problému obchodného cestujúceho.

Úloha 10.2. Zostrojte Hopfieldovu a Boltzmannovu sieť na priradenie práce strojom vo fabrike. Nech je M úloh, N strojov, cena priradenia úlohy k pre stroj i je c_{ik} . Každý stroj i má celkové množstvo zdrojov b_i , z ktorého sa úlohou k vyčerpá množstvo a_{ik} . Cieľom je minimalizovať cenu priradenia každej úlohy práve jednému stroju bez toho, že by sa prečerpával zdroj b_i tohto stroja. Tento problém sa dá formálne popísať ako minimalizácia energie E

$$E = \sum_{i=1}^N \sum_{k=1}^M c_{ik} x_{ik} \text{ s obmedzujúcimi podmienkami } \sum_{k=1}^M a_{ik} x_{ik} \leq b_i \quad (i=1, \dots, N), \quad (10.25)$$

$$\sum_{i=1}^N x_{ik} = 1 \quad (k=1, \dots, M)$$

kde $x_{ik} = 0$ alebo 1. Simulujte na počítači pre dáta v tabuľke 10.1, kde $N=3$, $M=8$, a $a_{ik} = a_k$ pre všetky i . (Každý uzol u_{ik} v sieti bude zodpovedať priradeniu úlohy k pre stroj i , taký uzol môže byť buď zapnutý alebo vypnutý). Ideálne optimum je vyznačené podčiarknutými hodnotami c_{ik} , súčet týchto hodnôt je 47. V ideálnom prípade by sa teda mali aktivovať uzly zodpovedajúce podčiarknutým pozíciám.

Tabuľka 10.1. Vstupné hodnoty pre úlohu 10.2

a_k	7	9	8	4	2	3	10	1
-------	---	---	---	---	---	---	----	---

b_i	
8	
24	
12	

c_{ik}								
	6	12	15	13	4	5	8	7
	18	10	6	4	2	8	14	4
	9	7	10	4	8	1	4	12

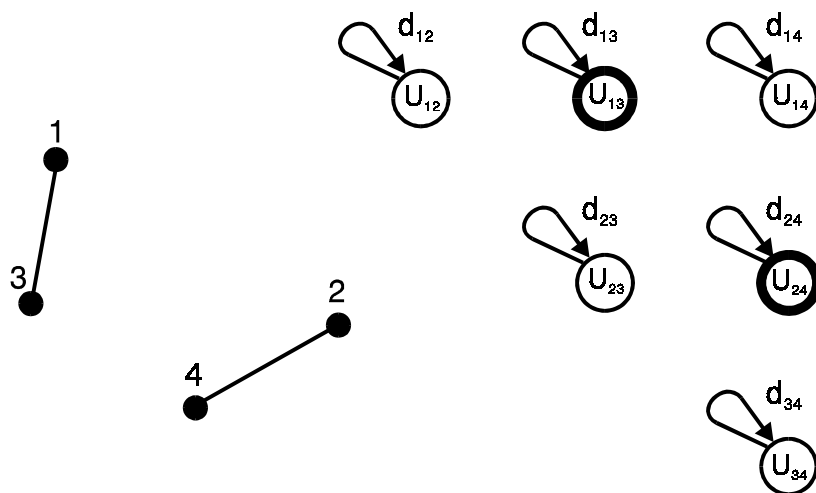
Hopfieldov-Tankov prístup k riešeniu problému využívajúci gradient na riešenie kombinatorického problému je síce originálny, no v praxi zatiaľ nevyužiteľný, pretože pre n miest je potrebných n^2 uzlov. Rovnaký problém má Boltzmannov stroj. Daný algoritmus funguje s problémami pre rádovo desať až dvadsať vrcholov, zatiaľ čo čisto simulované žihanie s mutáciou tvorenou prehodnením pozícií v permutácii alebo ešte špecializovanejšie prístupy Lina [66] fungujú pre stovky miest, pričom poskytujú výsledky blízke optimálnym alebo optimálne. Riešenie kombinatorických optimalizačných problémov pomocou neurónových sietí začne byť zaujímavé až pri ich hardvérovej implementácii.

Podobne sa prakticky nepoužívajú ani modifikácie sietí pre ďalšie kombinatorické optimalizačné problémy. Pri týchto problémoch hľadáme riešenie v množine veľkého množstva možných kombinácií základných prvkov systému. Celkový počet riešení pre veľkosť problému s počtom prvkov n je obvykle exponenciálnou funkciou n a tomu je úmerný aj čas potrebný na vyriešenie problému. Konkrétne ide napr. o *problém najcennejšieho párenia* (*weighted matching problem*) [25,26]. Pri probléme najcennejšieho párenia máme v priestore množinu n bodov, pričom poznáme vzdialenosti medzi jednotlivými dvojicami týchto bodov d_{ij} . Tieto body sa môžu nachádzať v euklidovskom priestore a d_{ij} môže reprezentovať euklidovskú vzdialenosť, alebo tieto body môžu byť abstraktné entity a hodnoty d_{ij} môžu reprezentovať ich vzťahy. Vo všeobecnosti sa môžu d_{ij} považovať za nezávislé náhodné premenné s pravdepodobnostnou distribúciou $P(d_{ij})$. Našou úlohou je pospájať dvojice bodov tak, aby každý bod bol spojený len s jedným iným bodom a aby celková dĺžka spojení bola minimálna (viď obr. 10.13).

Praktickými príkladmi takéhoto problému môže byť spájanie prvkov v nejakom elektronickom zariadení, optimálne mapovanie procesov na dva ekvivalentné procesory, priraďovanie žiakov do škôl a pod. Každému páru bodov i a j priradíme prvok u_{ij} , $i < j$. Nech $u_{ij} = 1$, keď medzi i -tým a j -tým bodom existuje spojenie, a $u_{ij} = 0$, keď medzi nimi spojenie nie je. Sám so sebou sa bod nemôže spojiť, takže $u_{ii} = 0$, a pre $j < i$ platí $u_{ij} = u_{ji}$. Funkcia, ktorej minimum hľadáme, je celková dĺžka spojení

$$L = \sum_{i < j} d_{ij} u_{ij} . \quad (10.26)$$

Túto funkciu budeme musieť trochu modifikovať, lebo každé riešenie musí spĺňať obmedzenie, že každý jeden bod môže byť spojený iba s jedným ďalším bodom, a teda $\sum_j u_{ij} = 1$ pre $\forall i$. Dodržiavanie tohto obmedzenia sa rieši pokutovaním v optimalizovanej funkcii.



Obrázok 10.13. Problém najcennejšieho párenia. Na ľavej strane je graf zložený zo štyroch vrcholov, kde sú vybrané také dvojice vrcholov, aby súčet vzdialeností vrcholov v dvojiciach bol čo najmenší. Na pravej strane je zodpovedajúca sieť, kde indexy pri uzloch označujú čísla vrcholov hrany, ktorej uzol prislúcha, a zvýraznené sú tu tie dva uzly, ktoré majú ohodnotenie 1 a podľa ktorých sme priradili hrany grafu naľavo.

Ako základ pre účelovú funkciu zoberieme (10.26) a pridáme k nej člen, ktorý bude rásť priamo úmerne porušeniu danej podmienky. Optimalizovaná funkcia E bude rovná:

$$E = \sum_{i < j} d_{ij} u_{ij} + \frac{\gamma}{2} \sum_{i=1}^n \left(1 - \sum_{j=1}^n u_{ij} \right)^2. \quad (10.27)$$

Veľkosť konštanty γ by mala byť asi taká ako priemerná hodnota d_{ij} . Pri použití Boltzmannovho stroja náhodne zapíname alebo vypíname uzly, pričom pravdepodobnosť, že sa zmení hodnota ľubovoľného prvku u_{ij} , je rovná

$$P(u_{ij} \rightarrow u'_{ij}) = \frac{1}{1 + \exp(\beta \Delta E)} \quad \text{kde } \Delta E = E(u'_{ij}) - E(u_{ij}). \quad (10.28)$$

Toto pravidlo hovorí, že zmeny, ktoré minimalizujú optimalizovanú funkciu E , sú pravdepodobnejšie ako tie, ktoré ju zvyšujú. Hľadáme teda takú distribúciu u_{ij} , ktorá zodpovedá jednému z miním (10.27). Problém môžeme riešiť aj pomocou analógového systému, vtedy budeme u_{ij} blízke 1 (0) považovať za rovné 1 (0). Je vhodné použiť i metódu simulovaného žihania a v priebehu evolúcie znižovať teplotu $T = \beta^{-1}$.

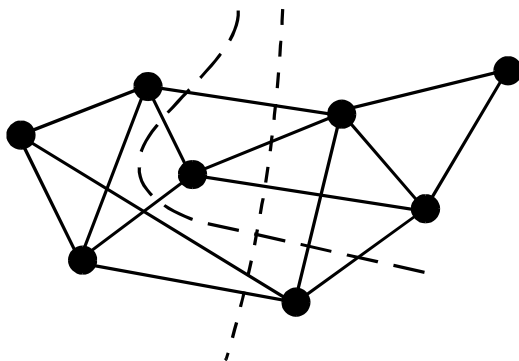
Posledným príkladom kombinatorického optimalizačného problému je *rozklad vrcholovej množiny grafu na dve časti (graph bipartitioning)* [28] (viď. obr. 10.14).

Predstavme si, že navrhujeme čip s N prvkami, ale všetky sa nám naň nezmestia. Potom by sme chceli urobiť dva čipy tak, aby polovica prvkov bola na jednom a polovica prvkov na druhom a aby počet spojení medzi týmito dvoma čipmi bol minimálny. Uvažujme všeobecný graf, t.j. množinu N bodov - vrcholov grafu pospájaných hranami, ktoré spájajú dvojice vrcholov. Nech je N párne. Nech p je fixná pravdepodobnosť existencie hrany medzi ľubovoľnými dvoma vrcholmi. Priemerný počet vrcholov pN , ktoré sú navzájom spojené, sa nazýva valencia grafu. Našou úlohou je rozdeliť vrcholy do dvoch rovnako veľkých množín, medzi ktorými je minimálny počet hrán. Definujme si $c_{ij} = 1$, keď sú vrcholy i a j spojené hranou, a $c_{ij} = 0$, keď nie sú spojené. Ďalej si pre každý vrchol zadefinujeme premennú

$S_i = +1$, keď sa vrchol nachádza v prvej množine, a $S_i = -1$, keď sa vrchol nachádza v druhej množine. Chceme minimalizovať funkciu

$$L = -\sum_{\langle ij \rangle} c_{ij} S_i S_j, \quad (10.29)$$

kde $\langle ij \rangle$ znamená indexovanie cez každý pár vrcholov zvlášť. Premenné S_i podliehajú obmedzeniu $\sum_i S_i = 0$.



Obrázok 10.14. Dve možné rozdelenia vrcholov grafu znázornené prerušovanou čiarou na dve podmnožiny vrcholov tak, aby medzi týmito podmnožinami existovalo čo najmenej hrán. V obidvoch rozdeleniach existuje päť hrán spájajúcich podmnožinu na pravej a na ľavej strane prerušovanej čiary. V danom prípade môžu vrcholy priamo predstavovať uzly siete ohodnotené $S_i=0$, keď sú priradené jednej množine, a $S_i=1$, keď sú priradené druhej množine.

V termínoch teórie magnetických látok tieto rovnice zodpovedajú feromagnetu s nulovou celkovou magnetizáciou. Každú celkovú magnetizáciu, ktorá sa vzdáľuje od 0, budeme v objektívnej funkcii pokutovať, takže objektívna funkcia systému je

$$E = -\sum_{\langle ij \rangle} c_{ij} S_i S_j + \mu \left(\sum_i S_i \right)^2. \quad (10.30)$$

Hodnotu μ vyberáme približne z intervalu (0,0.5). Rovnica (10.30) pripomína energiu systému analogického spinového sklu. Systém naštartujeme z náhodnej nekorelovanej konfigurácie $\mathbf{S}$ a necháme relaxovať, pričom S_i sa menia podľa stochastického pravidla (10.28) s S_i namiesto u_{ij} , kde energia E je vyjadrená ako (10.30). Opäť bude užitočné použiť simulované žihanie.

10.3.2 Riešenie problému obchodného cestujúceho pomocou Kohonenovej siete

Kohonenova SOM (SamoOrganizujúca sa Mapa) [67,68] patrí do kategórie sietí, ktoré sa *učia bez učiteľa* (*unsupervised learning*). To znamená, že algoritmus učenia nemá informáciu, aké váhy pri jednotlivých prvkoch siete sú správne, na rozdiel sietí s dopredným šírením v prvej časti kapitoly, ktoré pre tréningovú množinu mali spolu so vstupným vektorom vždy aj ideálny výstupný vektor. Základnou črtou Kohonenových sietí je schopnosť realizovať zobrazenie zachovávajúce topológiu tréningovej množiny dát. To znamená, že body (vstupné

vektory), ktoré sa podobali svojimi hodnotami, sa premietnu na uzly, ktoré budú "ležať" blízko seba. Termín "ležať blízko seba" bude závislý od topológie siete, ktorá je väčšinou buď lineárna (uzly sú formálne usporiadané do reťazca) alebo mriežková (uzly sú usporiadané v dvojrozmernnej pravouhlej mriežke). Susednosť v takýchto sieťach sa meria iba počtom uzlov, cez ktoré treba prejsť, aby sme sa dostali od jedného uzla k druhému, nejde teda o euklidovskú vzdialenosť. Uzly v reťazci majú dvoch najbližších susedov - naľavo a napravo, uzly v pravouhlej mriežke (uprostred mysleného štvorca) majú ôsmich najbližších susedov - hore, dole, naľavo, napravo (uprostred myslených hrán štvorca) a naprieč na uhlopriečkach (na myslených rohoch štvorca). Každý z týchto bodov je ohodnotený vektorom (spočiatku náhodne vygenerovaným) rovnakého typu, ako sú vstupné vektory. Pri vkladaní vstupných dát sa pre každý vstupný vektor nájde "vítazný" uzol, ktorého vektor má od daného vstupu najmenšiu (väčšinou euklidovskú) vzdialenosť. Hodnoty vektora tohto uzla sa potom zmenia podľa tzv. Hebbovho pravidla učenia tak, aby euklidovská vzdialenosť od daného vstupu bola menšia. Podobne sa danému vstupnému vektoru môžu priblížiť aj hodnoty vektorov uzlov, ktoré sú s "vítazným" uzlom susedné. Pri lineárnej topológii to budú susedné uzly vľavo a vpravo. Môžeme uvažovať aj širšie okolie zahŕňajúce viac uzlov vľavo a vpravo od víťazného uzla, no čím vzdialenejšie sú uzly, tým menej by sa hodnoty vektorov týchto uzlov mali blížiť hodnotám vstupného vektora. To je v algoritme 10.3 zohľadnené funkciou $h(I_{min}, i)$, kde táto funkcia určuje, v akom okolí víťazného uzla a do akej miery budú adaptované ostatné uzly.

```

Inicializuj váhy  $w_{ij}$ , kde vektor  $\mathbf{w}_i = (w_{i1}, w_{i2})$  je vektorom priradeným
uzlu  $i$ . Váhy sú generované náhodne v rozmedzí maximálnych a
minimálnych hodnôt súradníc  $(x, y)$  miest, pre  $i \in \{1, \dots, n\}$ , kde  $n$ 
je počet miest (napr. 10);
Prirad mestám topológiu cyklu, teda mesto  $i$  susedí naľavo s mestom
 $i-1$  a napravo s mestom  $i+1$ . Mesto 1 susedí naľavo s mestom  $n$  a
opačne, mesto  $n$  teda susedí napravo s mestom 1;
Nastav rýchlosť učenia  $\alpha$  (bude nastavená na 0.5 a bude klesať na
0.4);
Nastav veľkosť okolia (zo začiatku 1, teda okrem víťazného uzla
upravujeme vždy váhy aj jeho ľavého a pravého suseda);
while nie je splnená ukončovacia podmienka (napr. počet
cyklov=200)
begin
  begin for  $j=1$  to  $n$  { pre všetky vstupné vektory miest  $(x_j, y_j)$  }
    begin
      for  $i = 1$  to  $n$  do  $d(i) = (w_{i1} - x_j)^2 + (w_{i2} - y_j)^2$ ;
       $I_{min} = \text{index } i$ , kde  $d(i)$  je minimálne;
      pre víťazný uzol  $I_{min}$  a jeho okolie uprav váhy, teda
      pre  $i = \{ I_{min}-1; I_{min}; I_{min}+1 \}$ ;
       $w_{i1}(\text{nová}) := w_{i1}(\text{stará}) + \alpha h(I_{min}, i) [x_j - w_{i1}(\text{stará})]$ ;
       $w_{i2}(\text{nová}) := w_{i2}(\text{stará}) + \alpha h(I_{min}, i) [y_j - w_{i2}(\text{stará})]$ ;
    end;
  end;
  uprav rýchlosť učenia (napr.  $\alpha := \alpha (0.4/0.5)^{0.005}$ );
  zmenši okolie uzla pri splnení zadanej podmienky (napr. pri
  počte cyklov väčšom ako 100 už budeme upravovať iba váhy
  víťazného uzla, bez susedov);
end

```

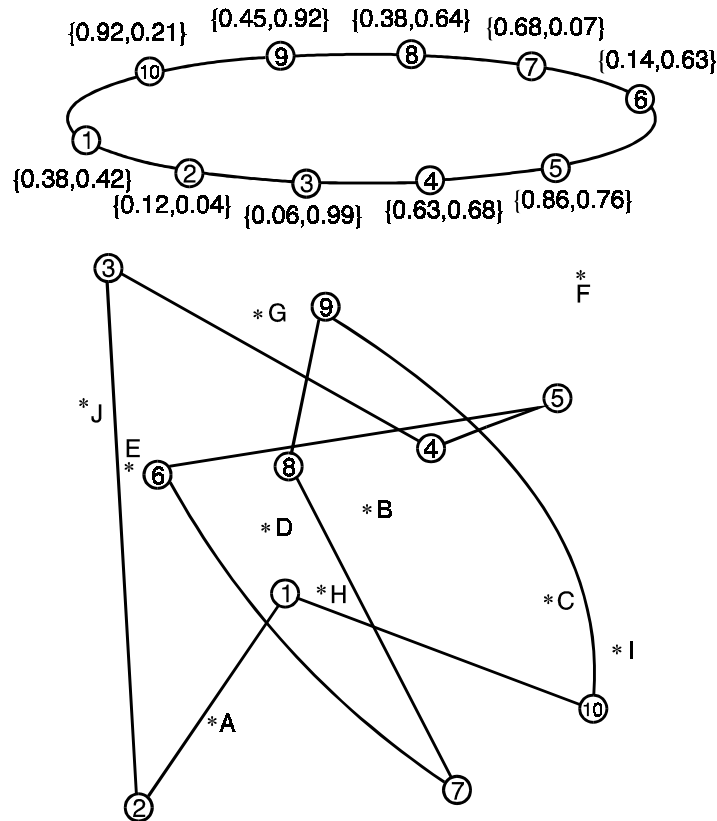
Algoritmus 10.3. Kohonenov algoritmus pre problém obchodného cestujúceho.

Najjednoduchšia používaná funkcia je

$$h(l_{min}, i) = \begin{cases} 1 & \text{ak } d_M(l_{min}, i) \leq 1 \\ 0 & \text{v ostatných prípadoch} \end{cases} \quad (10.31)$$

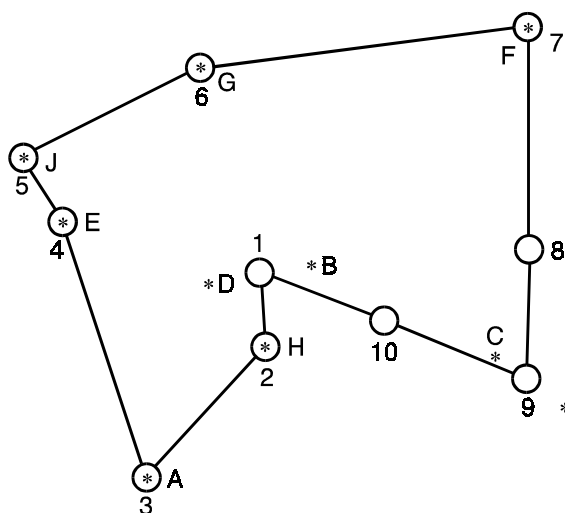
kde $d_M(l_{min}, i)$ je vzdialenosť typu Manhattan medzi neurónmi l_{min} a i , teda počet hrán, po ktorých treba v danej topológii siete prejsť, aby sme sa dostali od neurónu l_{min} k neurónu i . Niekedy sa pre vzdialenejšie okolie volí dokonca “záporné” učenie, teda že vektory uzlov vzdialených cez viac hrán od víťazného uzla sa “vzdďaľujú” od hodnôt vstupného vektora, zatiaľ čo ešte vzdialenejšie uzly už ostávajú bezo zmeny. Tento proces učenia sa opakuje stále dookola. Veľkosť okolia a parameter učenia (teda miera priblíženia vektora víťazného uzla vstupnému vektoru) sa znižujú s časom v priebehu algoritmu.

Kohonenova sieť sa typicky používa na *zhlukovú analýzu (clustering)*, kedy sa v “naučenej” sieti vždy niekoľko vstupov premietne (zobrazí) na jeden uzol. O týchto vstupoch a im zodpovedajúcich objektoch potom hovoríme, že patria do jedného zhluku, a všeobecne sa predpokladá, že objekty v jednom zhluku by mali mať podobné vlastnosti.



Obrázok 10.15. Riešenie problému obchodného cestujúceho pomocou Kohonenovej siete. V hornej časti je zobrazená topológia siete, uzly sú označené krúžkami a očíslované od 1 po 10, pri každom uzle je jemu prislúchajúci náhodne vygenerovaný počiatkový vektor w súradníc $\{x,y\}$. V dolnej časti sú jednotlivé uzly umiestnené v ploche so súradnicami odpovedajúcimi ich vektoru. Rovnako sú v ploche umiestnené aj mestá znázornené hviezdčkami a označené A,B,C,D,E,F,G,H,I,J. K týmto mestám by sa jednotlivé uzly mali v priebehu algoritmu približovať.

Na riešenie problému obchodného cestujúceho sa používa topológia lineárneho reťazca so spojenými koncami (teda formálne kruhu), kde počet uzlov by mal byť rovný počtu miest [31]. Vstupné vektory budú dvojrozmerné, tvorené súradnicami $\{x,y\}$ jednotlivých miest. Počiatočné hodnoty vektorov jednotlivých uzlov sa vygenerujú náhodne v rovnakom rozmedzí hodnôt ako súradnice miest (niekedy sa tiež používa sústredenia týchto počiatočných hodnôt v strede štvorca vymedzujúceho súradnice miest), (viď. obr. 10.15). Vektory jednotlivých uzlov sa postupne približujú k hodnotám súradníc miest a v ideálnom prípade je na konci učenia každému miestu priradený jeden uzol (množina miest je rozdelená na zhľuky obsahujúce jednotlivé mestá) a postupnosť hrán medzi uzlami zodpovedá postupnosti miest s najkratšou vzdialenosťou cesty (viď obr. 10.16). Niekedy sa môže stať, že dve mestá sa zobrazia na jeden uzol a iný uzol nie je "vítězný" ani pre jedno mesto. V takom prípade je poradie miest zobrazených na jeden uzol nejednoznačné; budú síce umiestnené vedľa seba, no nevieme, ktoré je prvé. Ako výsledok môžeme zobrať dve cesty líšiac sa poradím týchto dvoch miest. Ako vidno z obr. 10.16, pokiaľ nastane taký prípad viackrát, situácia sa komplikuje. Z uvedeného príkladu je vidieť, že použitie Kohonenovej siete pre problém obchodného cestujúceho je zaujímavé iba z teoretického hľadiska. No Kohonenova sieť sa v praxi využíva [67,68] napr. pri rozpoznávaní vzorov (hlások reči), v robotike (transformácia súradníc) alebo pri kompresii obrazov.



Obrázok 10.16. Na obrázku je výsledok riešenia problému obchodného cestujúceho, kde počiatočný stav siete bol znázornený na obr. 10.15. Vidíme, že štyrom z desiatich miest nebolo priradené jednoznačné poradie, pretože uzly 8 a 10 nie sú víťazné pre žiadne mesto, zatiaľ čo uzol 9 je víťazný pre dve mestá C a I, a rovnako uzol 1 je víťazný pre dve mestá B a D. Pretože u týchto dvojíc miest sieť neurčila poradie, je prípustné akékoľvek poradie v dvojiciach a výsledkom sú teda štyri cesty: BDHAEJGFCI, DBHAEJGFCI, BDHAEJGFIC a DBHAEJGFIC.

10.4 Záver

Neurónové siete ako predmet optimalizácie zvyčajne nie sú vhodným objektom evolučnej optimalizácie. Pri optimalizácii váh a prahových faktorov viacvrstvovej siete s dopredným šírením sa väčšinou lepšie uplatnia gradientové metódy a evolučné metódy tu výhodne slúžia iba na generáciu počiatočných hodnôt pre gradientové metódy. Budúce uplatnenie pre

evolučné metódy pri optimalizácii neurónových sietí spočíva teda v prípadoch, kedy je gradient ťažko použiteľný - u rekurentných sietí a pri učení bez učiteľa (keď nepoznáme, aké majú byť presné hodnoty na výstupe), čo býva väčšinou pri problémoch riadenia.

Optimalizácia topológie je podmienená nutnosťou ďalej optimalizovať váhy pre každú navrhnutú topológiu. Preto sa na tieto účely používajú rôzne heuristiky a navrhujú úzko špecializované algoritmy, ktoré sa svojím smerovaním k optimu podobajú skôr horolezeckým ako evolučným algoritmom.

Prehľad novších výsledkov týkajúcich sa optimalizácie neurónových sietí, kde sa hovorí aj o optimalizácii pravidiel učenia, je možno nájsť napr. v [69].

Okrem riešenia problému obchodného cestujúceho sa pomocou neurónových sietí dá riešiť aj problém ruksaku (knapsack), rozvrhu (scheduling), rozdelenia vrcholovej množiny na viac častí (graph partitioning), a pod. [70]. Neurónové siete ako optimalizujúce metódy sa okrem špecializovaných problémov pri spracovaní obrazov prakticky budú dať používať iba v hardvérovej implementácii, inak sú v súčasnej podobe na praktické použitie príliš pomalé.

Literatúra

- [1] V. Kvasnička, L. Beňušková, J. Pospíchal, I. Farkaš, P. Tiňo, A. Král: *Úvod do teórie neurónových sietí*. IRIS, Bratislava 1997.
- [2] P. Sinčák, G. Andrejková: *Neurónové siete I, Neurónové siete II*. elfa, Košice 1996.
- [3] M. Šnorek, M. Jiřina: *Neuronové sítě a neuropočítače*. Skripta vydavatelství ČVUT, Praha 1996.
- [4] M. Novák: *Neuronové sítě a neuropočítače*. Edice Výběr, SENZO a.s., Praha 1992.
- [5] M. Novák, J. Fáber, O. Kufudaki: *Neuronové sítě a informační systémy živých organizmů*. Grada, Praha 1992.
- [6] M. Novák, O. Kufudaki, P. Moos, P. Musílek, E. Pelikán, V. Šebesta: *Umělé neuronové sítě. Teorie a aplikace*. C.H. BECK, Praha 1998.
- [7] J. Šíma, R. Neruda: *Teoretické otázky neuronových sítí*. MATFYZPRESS, vydavatelství Mat.-fyz. fakulty Univerzity Karlovy, Praha 1996.
- [8] E. Volná: *Neuronové sítě a genetické algoritmy*. Vydavatelství Ostravské univerzity, Ostrava 1998.
- [9] M. Oravec, J. Polec, S. Marchevský: *Neurónové siete pre číslicové spracovanie signálov*. FABER, Bratislava 1998.
- [10] E. Vonk, L. C. Jain, and R. P. Johnson: *Automatic Generation of Neural Network Architecture Using Evolutionary Computation*. World Scientific Pub. Co., Singapore 1997.
- [11] J.D. Schaffer, D. Whitley, and L. Eshelman: Combination of Genetic Algorithms and Neural Networks: The state of the art. In D. Whitley and J.D. Schaffer (eds.): *COGANN-92: International Workshop on Combination of Genetic Algorithms and Neural Networks*. IEEE Computer Society Press, Los Alamitos, CA 1992.
- [12] D. Whitley: Genetic Algorithms and Neural Networks. In G. Winter, J. Periaux, M. Galan, and P. Cuesta (eds.): *Genetic Algorithms in Engineering and Computer Science*. J. Wiley, Chichester 1996.
- [13] E.J. Chang and R.P. Lippmann: Using genetic algorithms to improve pattern classification performance. In R.J. Lippman, J.E. Moody, and D.S. Touretsky (eds.): *Advances in Neural Information Processing 3*. Morgan Kaufmann, San Mateo, CA 1991, pp. 797-803.
- [14] F.Z. Brill, D.E. Brown, and W.N. Martin: Fast genetic selection of features for neural network classifiers. *IEEE Transactions on Neural Networks* **3** (2) (1992) 324-328.

- [15] R.J. Mitchell, J.M. Bishop, and W. Low: Using a genetic algorithm to find the rules of a neural network. In R.F. Albrecht, C.R. Reeves, and N.C. Steele (eds.): *Artificial Neural Nets and Genetic Algorithms*. Springer Verlag, Wien 1993, pp. 664-669.
- [16] R.C. Eberhart and R.W. Dobbins: Designing neural network explanation facilities using genetic algorithms. In *IEEE international joint conference on neural networks*. IEEE Press, New York, 1991, pp. 1758-1763.
- [17] R.C. Eberhart: The role of genetic algorithms in neural network query-based learning and explanation facilities. In D. Whitley and J.D. Schaffer (eds.): *COGANN-92: International Workshop on Combination of Genetic Algorithms and Neural Networks*. IEEE Press, Los Alamitos, CA 1992.
- [18] J.J. Hopfield: Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the USA* **79** (1982) 2554-2558.
- [19] V.S. Dotsenko: Neural networks: translation-, rotation- and scale-invariant pattern recognition. *Journal of Physics A: Mathematics and General* **21** (1988) L783-L787.
- [20] T. Poggio and C. Koch: Ill-Posed Problems in Early Vision: From Computational Theory to Analogue Networks. *Proc. Royal Soc. Lond. B* **226** (1985) 303-323.
- [21] C. Koch, J. Marroquin, and A. Yuille: Analog "Neuronal" Networks in Early Vision. *Proc. Natl. Acad. Sci. USA* **83** (1986) 4263-4267.
- [22] J.M. Hutchinson and C. Koch: Simple Analog and Hybrid Networks for Surface Interpolation. In J.S. Denker (ed.): *Neural Networks for Computing*. American Inst. of Physics Conf. Proc. Vol. **151**, 1987, pp. 235-240.
- [23] D. Geiger and A. Yuille: A Common Framework for Image Segmentation. *International Journal of Computer Vision* **6**(3) (1991) 227-243.
- [24] J. Hertz, A. Krogh and R.G. Palmer: *Introduction to the Theory of Neural Computation*, Addison-Wesley Publ. Comp., Redwood City 1991.
- [25] J.J. Hopfield and D.W. Tank: "Neural" computation of decisions in optimization problems. *Biological Cybernetics* **52** (1985) 141-152.
- [26] J.J. Hopfield and D.W. Tank: Computing with neural circuits: a model. *Science* **233** (1986) 625-633.
- [27] G.V. Wilson and G.S. Pawley: On the Stability of the Traveling Salesman Problem Algorithm of Hopfield and Tank. *Biological Cybernetics* **58** (1988) 63-70.
- [28] Y. Fu and P.W. Anderson: Application of statistical mechanics to NP-complete problems in combinatorial optimization. *Journal of Physics A: Mathematics and General* **19** (1986) 1605-1620.
- [29] R. Durbin and D. Willshaw: An Analogue Approach to the Traveling Salesman Problem Using an Elastic Net Method. *Nature* **326** (1987) 689-691.
- [30] D. J. Burr: An Improved Elastic Net for the Traveling Salesman Problem. In *IEEE International Conference on Neural Networks (2nd)*, San Diego, July 1988, IEEE Press, pp. 69-76.
- [31] B. Angeniol, G. Vaubois, and J-Y. Le Texier: Self-organizing Feature Maps and the Traveling Salesman Problem. *Neural Networks* **1**(4) (1988) 289-293.
- [32] P.J. Angeline, G.M. Saunders, and J.B. Pollack: An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks* **5**(1) (1994) 54-64.
- [33] D.E. Rumelhart, G.E. Hinton, and R.J. Williams: Learning internal representation by error propagation. In D.E. Rumelhart, J.L. McClelland, and PDP Research Group: *Parallel Distributed Processing. Explorations in the Microstructure of Cognition. Vol 1: Foundation*. The MIT Press, Cambridge, MA 1987, pp. 318-362.
- [34] T. Masters: *Advanced Algorithms for Neural Networks: A C++ Sourcebook*. J. Wiley, New York 1995.

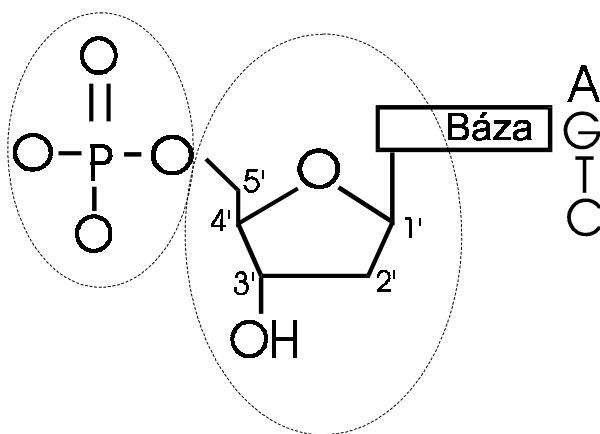
- [35] G.F. Miller, P.M. Todd, and S.U. Hedge: Designing neural networks using genetic algorithms. In J.D. Schaffer (ed.): *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA 1989, pp. 379-384.
- [36] J.D. Schaffer, R.A. Caruana, L.J. Eshelman: Using genetic search to exploit the emergent behavior of neural networks. In S. Forrest (ed.): *Emergent Computation*. North Holland, Amsterdam 1990, pp. 244-248.
- [37] S.A. Harp and T. Samad: Genetic Synthesis of Neural Network Architecture. In L. Davis (ed.): *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York 1991, pp. 202-221.
- [38] H. Kitano: Designing neural networks using genetic algorithms with graph generation systems. *Complex Systems* **4** (1990) 461-476.
- [39] F. Gruau: Genetic synthesis of Boolean neural networks with a cell rewriting developmental process. In D.L. Whitley and J.D. Schaffre (eds.): *COGANN-92: International Workshop on Combination of Genetic Algorithms and Neural Networks*. IEEE Press, Los Alamitos, CA 1992.
- [40] H. Kitano: Neurogenetic learning: An integrated method of designing and training neural networks using genetic algorithms. *Physica D* **75** (1994) 225-238.
- [41] R.K. Belew: Interposing an ontogenetic model between genetic algorithms and neural networks. In S.J. Hanson, J.D. Cowan, and C.L. Giles (eds.): *Advances in Neural Information Processing (NIPS 5)*. Morgan Kaufmann, San Mateo, CA 1993.
- [42] D. Whitley, F. Gruau, and L. Pyeatt: Cellular Encoding Applied to Neurocontrol. In L. Eshelman (ed.): *5th Intern. Conf. on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA 1995. Dostupné na internetovskej adrese <http://www.cs.colostate.edu/~genitor/Pubs.html>
- [43] J. R. Koza: *Genetic Programming (II): Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA 1994.
- [44] J. R. Koza: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA 1992.
- [45] D.J. Chalmers: The evolution of learning: An experiment in genetic connectionism. In D.S. Touretzky, J.L. Elman, T.J. Sejnowski, and G.E. Hinton (eds.): *Proceedings of the 1990 Connectionist Model Summer School*. Morgan Kaufmann, San Mateo, CA 1990.
- [46] D.E. Moriarty and R. Miikkulainen: Hierarchical Evolution of Neural Networks. In *Proceedings of the 1998 IEEE Conference on Evolutionary Computation (ICEC98, Anchorage, AK)*. IEEE Press, Piscataway, NJ 1998, pp. 428-433. Dostupné na internetovskej adrese <http://www.cs.utexas.edu/users/nn/pages/publications/neuro-evolution.html>
- [47] D.J. Montana and L.D. Davis: Training feedforward networks using genetic algorithms. In T. Dean (ed.): *Proceedings of the International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA 1989.
- [48] A.P. Weiland: Evolving controls for unstable systems. In D.S. Touretzky, J.L. Elman, T.J. Sejnowski, and G.E. Hinton (eds.): *Proceedings of the 1990 Connectionist Model Summer School*. Morgan Kaufmann, San Mateo, CA 1990, pp. 91-102.
- [49] A.P. Weiland: Evolving neural network controllers for unstable systems. In *IEEE international joint conference on neural networks*. IEEE Press, New York, 1991, pp. 667-673.
- [50] J.D. Schaffer (ed.): *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, Los Altos, CA 1989, pp. 360-397.
- [51] R.F. Albrecht, C.R. Reeves, and N.C. Steele (eds.): *Artificial Neural Nets and Genetic Algorithms*, Springer Verlag, Wien 1993, pp. 628-730.

- [52] D. W. Pearson, N. C. Steele, and R.F. Albrecht (eds): *Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Ales, France*. Springer Verlag, Wien 1995.
- [53] A. Dobnikar, N.C. Steele, and D. Pearson (eds.): *Artificial Neural Nets and Genetic Algorithms. Proceedings of the International Conference in Portoroz, Slovenia*. Springer Verlag, Berlin 1999.
- [54] A. Skinner and J.Q. Broughton: Neural Networks in Computational Material Science: Training Algorithms. *Modelling and Simulation in Material Science and Engineering* **3** (1995) 371-390.
- [55] M. McInerney and P. Dhawan: Use of genetic algorithms with backpropagation in training of feedforward neural networks. In *Proc. IEEE Int. Conf. Neural Networks*, vol. I. IEEE Press, San Francisco 1993.
- [56] D.B. Fogel, L.J. Fogel, and V. W. Porto: Evolving neural networks. *Biological Cybernetics* **63** (1990) 487-493.
- [57] E. Ising: Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift Fur Physik* **31** (1925) 253-287.
- [58] S. Kirkpatrick and D. Sherrington: Infinite-ranged models of spin-glasses. *Physical Review B* **17** (1978) 4384-4403.
- [59] H.G.E. Hentschel and A. Fine: Statistical Mechanics of Stereoscopic Vision. *Phys. Rev. A* **40** (1989) 3983-3997.
- [60] J. Hlaváč a M. Šonka: *Počítačové vidění*. Grada, Praha 1992.
- [61] T. Masters: *Signal and Image Processing with Neural Networks: A C++ Sourcebook*, J. Wiley, NY 1994.
- [62] A. Cichocki and R. Unbehauen: *Neural Networks for Optimization and Signal Processing*. J. Wiley, Chichester 1994, Chapters 3 and 7.
- [63] G.E. Hinton and T.J. Sejnowski: Optimal Perceptual Inference. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. IEEE Press, Washington DC 1983, pp. 448-453.
- [64] H.H. Szu and R. Hartley: Fast Simulated Annealing. *Physics Letters A* **122**(3,4) (1987) 157-162.
- [65] H.H. Szu: Colored Noise Annealing Benchmark by Exhaustive Solutions of TSP. In *International Joint Conference on Neural Networks*. IEEE Press, Washington DC, I 1990, pp. 317-320.
- [66] S. Lin: Computer Solutions of the Traveling Salesman Problem. *Bell Systems Technical Journal* **44** (1965) 2245-2269.
- [67] T. Kohonen: *Self-Organizing Maps*. Springer Verlag, Berlin 1997.
- [68] E. Oja and S. Kaski (eds.): *Kohonen Maps*. Elsevier Science, Amsterdam 1999.
- [69] X. Yao: Evolving artificial neural networks. *Proceedings of the IEEE* **87**(9) (1999) 1423-1447, dostupné na internetovskej adrese http://www.cs.bham.ac.uk/~xin/journal_papers.html.
- [70] C. Petersen and B. Söderberg: Artificial Neural Networks. In C.R. Reeves (ed.): *Modern Heuristic Techniques for Combinatorial Problems*. McGraw-Hill, London 1995, pp.197-242.

11. Výpočty na báze DNA

Zrod jednej z najrevolučnejších disciplín súčasnej informatiky - výpočtov na báze molekúl deoxyribonukleovej kyseliny (DNA) - sa spája s menom amerického matematika Leonarda M. Adlemana z Kalifornskej univerzity (University of Southern California). L. Adlemana zaujímal mechanizmus šírenia vírusu HIV a činnosť ľudského imunitného systému. Keď čítal článok Jamesa Watsona (mimočodom, spolu s Francisom Crickom, spoluobjaviteľa štruktúry DNA) o enzýme DNA-polymeráze, zarazila ho podobnosť biologických procesov so základnými procesmi súčasných výpočtových systémov. DNA možno použiť ako výpočtové médium, jediné, čo k tomu potrebujeme, je aparát realizujúci základné editovacie operácie ako vymazať, presunúť, či skopírovať podreťazec DNA. Skôr ako sa začneme vážnejšie zaoberať teóriou a praktickými výsledkami výpočtov na báze DNA, uvedieme základné pojmy molekulárnej biológie.

11.1 Štruktúra a replikácia DNA



Obrázok 11.1. Štruktúra nukleotidu - základnej stavebnej jednotky DNA. V menšom kruhu sa nachádza fosfátová skupina, väčší kruh vyznačuje cukor, na ktorý sa viaže jedna zo štyroch molekúl báz - adenín (A), guanín (G), tymín (T) a cytozín (C).

DNA - deoxyribonukleová kyselina (Deoxyribonucleic Acid) sa nachádza v každom živom organizme ako pamäťové médium pre genetickú informáciu. DNA patrí medzi nukleové kyseliny. Nukleové kyseliny sa skladajú z troch jednoduchých látok, ktoré vzájomným pospájaním tvoria dlhú molekulu. Kostru molekuly DNA tvorí päťuhlíkový monosacharid *deoxyribóza*. Na pozícii 1' sa na deoxyribózu viažu dusíkaté *bázy* (base), ktoré "vyčnievajú" z reťazca DNA. Reťazec DNA je tvorený molekulami cukru vzájomne pospájanými na pozícii 5' a 3' *zvyškom kyseliny fosforečnej*. Základnou stavebnou jednotkou molekuly DNA je *nukleotid* (nucleotide). Chemická štruktúra nukleotidu je znázornená na obr. 11.1. Nukleotidy sa odlišujú len bázou. V DNA sa vyskytujú štyri druhy báz - *adenín* (A), *guanín* (G), *tymín* (T) a *cytozín* (C).

Nukleotidy sa reťazia do molekuly DNA v procese nazývanom *polymerizácia*. Polymerizácia spočíva v reakcii medzi fosfátom na pozícii 5' jedného nukleotidu a

hydroxylovou skupinou na pozícii 3' susedného nukleotidu. Každá molekula DNA má dva rôzne konce - jeden s voľnou fosfátovou skupinou na pozícii 5' (5'-koniec) a ďalší s voľnou hydroxylovou skupinou na pozícii 3' (3'-koniec). V abstraktnom zmysle možno molekulu DNA stotožniť s (orientovaným) symbolickým reťazcom nad abecedou $x=(x_1, x_2, \dots, x_n)^T$.

DNA sa v prírode obvyčajne nevyskytuje vo forme lineárnych reťazcov nukleotidov. Za vhodných podmienok sa dva lineárne reťazce DNA spárujú a vytvoria *dvojítu závitnicu*, pričom navzájom si odpovedajúce bázy z rôznych DNA sú spojené vodíkovými mostíkmi tak, že A sa spája s T a G sa spája s C. Dvojice A/T a G/C sa nazývajú *komplementárne bázy*.

Komplementárne vlákna DNA sú k sebe pripojené opačnou orientáciou. Dvojité závitnice sa končí v jednom vlákne na pozícii 3' a v druhom vlákne na pozícii 5'. Pri replikácii DNA sa dvojité závitnice najskôr "rozpleť", takže vzniknú dva voľné jednovláknové reťazce DNA, a potom sa k voľným úsekom DNA "vyrobia" ich komplementárne kópie. Syntézu nového reťazca DNA riadi enzým *DNA-polymeráza*. Je zaujímavé, že polymeráza sa nielen zúčastňuje na syntéze nového komplementárneho vlákna DNA v smere od 5' k 3', ale súčasne aj kontroluje (v opačnom smere) správnosť zoradenia jednotlivých nukleotidov vo vlákne. Ak enzým DNA-polymeráza objaví chybu, je schopný ju (sám, alebo s pomocou ďalších enzýmov) opraviť.

Syntéza bielkovín na základe kódu v DNA prebieha tak, že najskôr sa obsah časti DNA skopíruje na *ribonukleovú kyselinu (RNA)*, ktorá vynesie kód z jadra bunky do cytoplazmy (preto sa tiež nazýva messenger RNA - *mRNA*) a až potom sa pristúpi k samotnej syntéze bielkoviny zo základných jednotiek - aminokyselín. Syntéza prebieha v špecializovaných útvaroch (zložených z RNA a proteínov) plávajúcich v cytoplazme bunky, ktoré sa nazývajú *ribozómy*. Každá aminokyselina je kódovaná trojicou báz (tzv. *kodónom*) na mRNA. Aminokyseliny sú na miesto syntézy prinášané špeciálnymi RNA nazývanými transferové RNA (*tRNA*). Ribonukleová kyselina sa od DNA líši len nepatrne. Je jednovláknová, ako cukor obsahuje ribózu a namiesto tymínu (T) má štruktúrne podobnú bázu *uracil (U)*.

Súčasná genetika venuje veľkú pozornosť mikroorganizmom, predovšetkým baktériám a fágom. Popri nenáročnej kultivácii je ich veľkou prednosťou, že sa rýchlo rozmnožujú. Napríklad z jedného fága môžeme za 20 minút dostať až 150 kópií.

Bakteriálna bunka má organizáciu živej hmoty typickú pre bunky všetkých organizmov. Je obalená bunkovou stenou a v cytoplazme sa nachádzajú dôležité organely (ako napríklad ribozómy uplatňujúce sa pri budovaní bielkovinových buniek na základe kódu DNA). Genetický systém baktérie je tvorený dlhým, kruhovito stočeným vláknom dvojzávitnice DNA, ktorá predstavuje *bakteriálny chromozóm*.

Bakteriofágy (bakteriálne vírusy) sú veľmi jednoduché systémy stojace na hranici medzi živou a neživou hmotou. Medzi najštudovanejšie fágy patria tzv. *T-fágy*, ktoré majú hlavičku s bičíkom a napádajú baktérie *Escherichia coli* (*E. coli*) nachádzajúce sa v črevách človeka. T-fág sa skladá z bielkovinového obalu, v ktorom je uložená DNA fága. Infekcia bakteriálnych buniek prebieha tak, že fág nasadne bičíkom na bunkovú stenu baktérie, naruší ju a do vnútra baktérie vnikne fágová DNA. Fágová DNA sa v bakteriálnej bunke začne rozmnožovať a zvyčajne znefunkční pamäťové médium genetickej informácie baktérie. Z DNA fága sa za výdatnej pomoci enzymatického systému baktérie budujú bielkovinové časti tiel nových fágov.

V minulom storočí sa zistilo, že baktérie sú pôvodcovia mnohých chorôb a v štyridsiatych rokoch tohoto storočia sa začala výroba prvého antibiotika - penicilínu. Odvtedy sa postupne vyvíjali nové a nové antibiotiká. Už v päťdesiatych rokoch sa však ukázalo, že účinnosť antibiotík začala prudko klesať. Odolnosť baktérií súvisí s genetickou adaptáciou mikroorganizmov. V baktériách totiž existujú okrem chromozomálnej DNA ešte aj iné geneticky autonómne jednotky - *plazmidy*. Gény zodpovedné za vlastnosť rezistencie baktérií voči antibiotikám sa nachádzajú v bakteriálnych R-plazmidoch. Plazmidy nie sú nevyhnutné pre životnú funkciu baktérie. Plazmidové DNA sú dôležité pre prežitie baktérií,

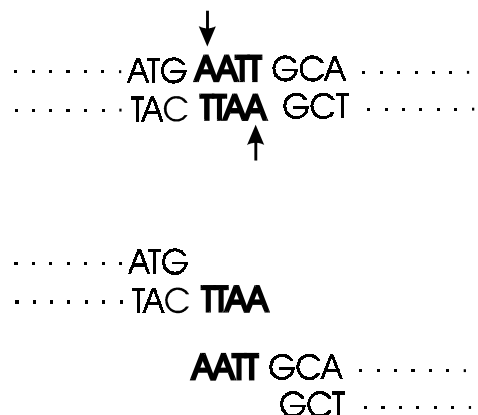
ak sa tieto dostanú do prostredia, proti ktorému plazmidy kódujú výrobu účinných ochranných látok. R-plazmidy môžu vyvolať konjugáciu baktérií a môžu sa v bunke replikovať nezávisle od replikácie bakteriálneho chromozómu. Dvojvláknová kruhová molekula R-plazmidu sa pri konjugácii rozpojí a presúva sa (podobne ako chromozóm) do ďalšej bunky. V pôvodnej hostiteľskej bunke sa vlákno znova dosyntetizuje.

V sedemdesiatych rokoch bola objavená ďalšia forma nezávislého genetického systému v baktériách. Ide o prvky DNA, ktoré sa od bakteriálneho chromozómu odpájajú a opäť pripájajú na rozličných pozíciách. Najskôr sa pre takéto prvky používalo pomenovanie mobilné gény, neskôr sa začali označovať ako *transpozóny*. Transpozóny majú na koncoch palindromické nukleotidové sekvencie, takže konce sú k sebe pripútané viazaním komplementárnych báz a voľná časť transpozónu nesie samotný putujúci funkčný gén. Zároveň tvoria konce transpozónov aj tzv. inzerčné sekvencie, pomocou ktorých sa môžu transpozóny vsunúť do rozličných miest bakteriálneho chromozómu a spolu s ním sa replikovať. Transpozóny sa od bakteriálneho chromozómu môžu aj odtrhnúť a existovať samostatne. Na rozdiel od plazmidov však nie sú schopné samostatne sa replikovať a pre rozmnožovanie potrebujú naviazanie na *replikóny*, čo sú úseky DNA schopné samostatnej replikácie. Transpozóny, okrem iného, môžu niesť aj gény rezistencie proti antibiotikám.

Obrovský význam extrachromozomálnych genetických systémov v baktériách spočíva v možnosti prenosu genetickej informácie nielen v rámci jednotlivých druhov mikroorganizmov, ale aj v rámci taxonomicky nepříbuzných rodov. Tak sa napríklad môže stať, že sa R-plazmid z vcelku nevinnej bunky baktérie *E. coli* dostane do nejakej vysokopatogénnej baktérie. Mimochromozomálny geneticky autonómny systém predstavuje v prírode zásobárňu rezistentných génov, ktoré sú schopné rýchlo sa šíriť do rozličných kmeňov baktérií. Tým sa zároveň zvyšuje genetická variabilita mikroorganizmov.

Zásadný objav, ktorý umožnil človeku aktívne zasahovať do genetického materiálu mikroorganizmov, bolo objavenie špecifických enzýmov - *reštrikčných endonukleáz*. Tieto štiepia molekulu DNA na špecifických reštrikčných miestach. Fungujú teda ako nožnice na strihanie molekúl DNA v špecifických pozíciách. Do roku 1970 boli známe len enzýmy, ktoré buď štiepili nukleové kyseliny na veľké časti, alebo ich rozbíjali na jednotlivé nukleotidy. Reštrikčné endonukleázy boli objavené pri štúdiu interakcie fágovej DNA s hostiteľskou bakteriálnou bunkou. Pozorovalo sa, že ak infikujeme jedným typom fága rovnako rezistentnú populáciu baktérií, v ďalších generáciách sa fág postupne prispôbuje ochrannému systému hostiteľských buniek a po niekoľkých generáciách sa už rozmnožuje na pôvodne rezistentných baktériách v plnej sile. Mechanizmus rezistencie bakteriálnej bunky voči fágovej infekcii spočíva v produkcii špecifického enzýmu-endonukleázy, ktorá štiepi cudzorodú (fágovú) DNA na malé kúsky. Chromozómová DNA vlastnej bunky je na určitých miestach označená metylovou skupinou, ktorá endonukleáze signalizuje, že ide o domácu DNA. Pri silných nákazách však ochranný systém baktérie dokonale nezničí fágovú DNA. Neporušené fágové DNA označí bakteriálna bunka na rovnakých miestach ako vlastnú. Po infekcii ďalšej bunky už endonukleáza nerozozná vlastnú DNA od cudzej.

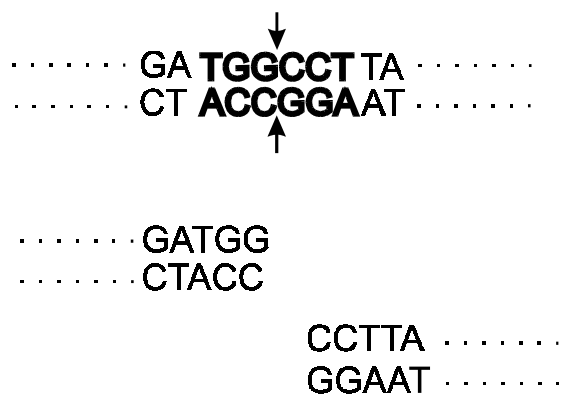
Reštrikčné endonukleázy majú dôležitú vlastnosť - štiepia DNA na rovnakých špecifických miestach. Po pôsobení reštrikčnej endonukleázy jedného typu vznikajú vždy rovnaké konce rozštiepenej dvojzávitnice DNA. Napríklad endonukleáza *EcoR I* štiepi dvojzávit DNA v sekvencii GAATTG (viď obr. 11.2). Rozštiepeným kúskom DNA sa hovorí *lepkavé konce* (sticky ends). Niektoré reštrikčné endonukleázy, ako napr. *Hae III*, rozstrihnú dvojzávit DNA priamo medzi dvojicami báz (obr. 11.3) a vzniknú tzv. tupé konce. Reštrikčné endonukleázy predstavujú prvý krok k získaniu presne definovaných malých úsekov DNA. Krátke molekuly DNA sa nazývajú aj oligonukleotidy.



Obrázok 11.2. Znáznorenie činnosti reštrikčného enzýmu *EcoR I*. Enzým “rozstrihne” dvojzávit DNA na miestach vyznačených šípkami, takže vzniknú dva voľné “lepkavé” konce.

Iný, nemenej dôležitý enzým pre génovú manipuláciu je *DNA-ligáza*. Tento enzým je schopný opäť spojiť jednotlivé konce rozstrihutej DNA. To napríklad umožňuje vložiť do rozstrihutej kruhovej molekuly plazmidu kúsok cudzorodej DNA a kruh uzatvoriť. Niektoré ligázy sú schopné spájať “tupé” konce molekúl DNA, ktoré vznikajú pri strihaní napríklad enzýmom *Hae III* (obr. 11.3). Takto sa na “tupé” konce môžu pripájať krátke umelo syntetizované úseky *DNA-linkery*, ktoré majú na druhom konci nukleotidové sekvencie vhodné na pripojenie k nejakej nosnej DNA. Pripravený úsek sa pomocou linkerov vloží do molekuly DNA, ktorej “lepkavé” konce majú (vzhľadom na “lepkavé” konce linkerov) komplementárne sekvencie nukleotidov.

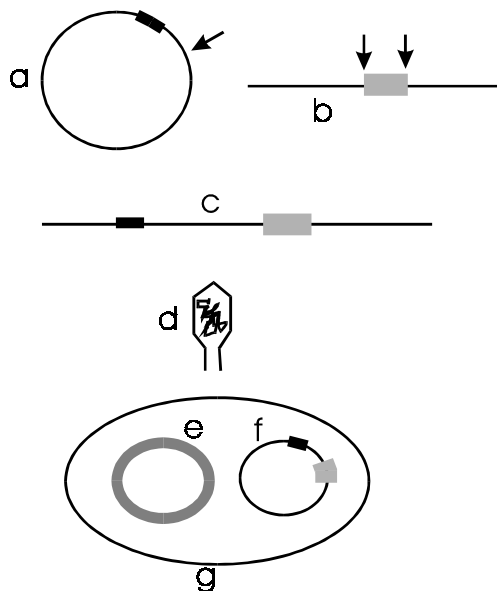
Jedným z najväznejších problémov molekulárnej genetiky je transport a integrácia cudzieho génu do genetického systému hostiteľskej bunky. Aby bolo možné úsek cudzej DNA v hostiteľskej bunke replikovať, je potrebné, aby bol spojený so systémom, ktorý je schopný samostatnej replikácie. Už sme spomenuli, že takéto systémy sa nazývajú replikóny. Zvyčajne sa takýto proces popisuje ako spojenie cudzorodej DNA s vhodným vektorom, spolu s ktorým sa prenesie do hostiteľskej bunky. Vektor sa v hostiteľskej bunke samostatne rozmnožuje a vytvorí viacero kópií. Takýto proces sa nazýva *molekulové klonovanie*. Najvhodnejšími vektormi sú plazmidy, fágy a vírusy.



Obrázok 11.3. Činnosť reštrikčnej endonukleázy *Hae III*. Tento enzým “rozstrihne” dvojzávitnicu DNA priamo medzi dvojicami báz a vzniknú dva “tupé” konce.

Hlavné etapy prípravy vektora sú: rozštiepenie plazmidovej kruhovej molekuly DNA, pripojenie cudzorodého fragmentu a opätovné spojenie do kruhu. Dnes sa okrem prírodných plazmidov používajú aj umelo skonštruované plazmidy. Ich hlavnou výhodou je lepšia manipulovateľnosť. Napríklad umelý vektor *pBR 322* sa skonštruoval na báze plazmidu *pSC 101*. Z pôvodného plazmidu sa okrem signálov pre replikáciu prebrala aj rezistencia voči ampicilínu a tetracyklínu. Nepotrebné nukleotidové sekvencie sa odstránili. Hmotnosť vektora *pBR 322* je v porovnaní s hmotnosťou plazmidu *pSC 101* trikrát menšia. Vektor *pBR 322* obsahuje 4361 párov báz.

Ako vektory sa používajú aj fágy a vírusy. Tieto sa rozmnožujú v hostiteľských bunkách a môžu vytvoriť až niekoľko sto kópií. *Fág lambda* má 48 502 bázových párov a rozmnožuje sa v baktérii *E. coli*. Má lineárnu molekulu DNA, ktorá za určitých okolností môže vytvoriť kruh. Podobne ako v prípade plazmidov sa vytvorili aj tzv. fagy *chárony*. Fágová DNA obsahuje až 40% sekvencií, ktoré nie sú potrebné pre vývoj fága a môžu sa vystrihnúť reštrikčnou endonukleázou. Umelý fág *M 13*, ktorý sa používa na transport génov, má jednovláknovú kruhovú DNA dĺžky 7200 nukleotidov. Bohužiaľ sa však molekuly fágovej DNA s naimplantovanou cudzorodou DNA môžu rozmnožovať, len ak dĺžka takéhoto vektora nepresiahne 105% dĺžky molekuly pôvodného fága.

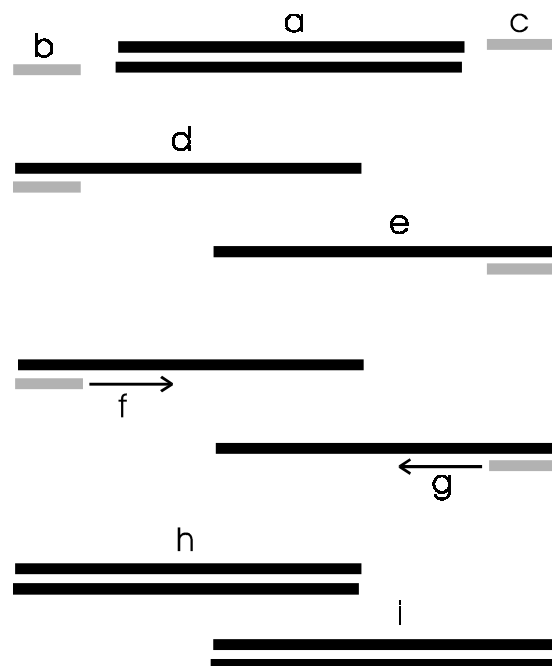


Obrázok 11.4. Prenos cudzorodej DNA do baktérie pomocou kozmidu. (a) Kozmid - kruhová štruktúra sa naruší reštrikčným enzýmom, (b) cudzorodá DNA - úsek určený na prenos - sa vystrihne reštrikčnou endonukleázou, (c) vystrihnutý úsek cudzorodej DNA sa vsunie do lineárneho vlákna pomocou ligázy, obnoví sa kruhová štruktúra kozmidu, (d) zabalenie rekombinantnej DNA do fágovej hlavy a následné prenesenie do baktérie (g), v ktorej sa teraz okrem bakteriálnej chromozomálnej DNA (e) nachádza aj kozmid (f) s "prepašovaným" úsekom cudzorodej DNA.

Na prenos a replikáciu väčších reťazcov DNA sa používajú tzv. *kozmidy*. Sú to umelo skonštruované útvary podobné plazmidom. Obsahujú informáciu pre "vbaľovací systém" fágov. Po pridaní *in vivo* pripravených súčastí fágov (obal, bičík) je celá kozmidová DNA "zabalená" a pomocou fágových mechanizmov vnesená do hostiteľskej bunky. V nej sa už replikuje samostatne, podobne ako plazmid. Mechanizmus prenosu cudzorodej DNA do hostiteľskej bunky pomocou kozmidu je znázornený na obr. 11.4. Pre porovnanie uvádzame,

že "veľké" plazmidy obsahujúce vyše 10 000 bázových párov sa už replikujú pomerne ťažko, kým fágy "unesú" až 30 000 bázových párov. Veľkosť kozmidov (45 000 - 50 000 bázových párov) je limitovaná veľkosťou fágovej hlavy.

Metóda molekulárneho klonovania vychádza z predpokladu, že bunka je schopná za určitých okolností replikovať cudzorodú DNA. Geneticky zmanipulovaný vektor sa musí dostať do hostiteľskej bunky, ktorá túto službu vykoná. Tieto procesy však prebiehajú za hranicami možnosti svetelného mikroskopu, a preto sa na sledovanie ich priebehu používajú nepriame genetické metódy. Ako príklad si uvedieme molekulárne klonovanie pomocou umelého plazmidu pBR 322. Využíva sa skutočnosť, že reštrikčné miesto, kde sa napája cudzorodá DNA, sa nachádza v gène na rezistenciu proti ampicilínu. Plazmid pBR 322 má však aj gén rezistencie proti tetracyklínu, ktorý zostane neporušený. Prakticky nie je možné zabezpečiť, aby všetky plazmidy obsahovali po génovej manipulácii cudzorodú rekombinantnú DNA. Ak vysejeme hostiteľské bakteriálne bunky na platňu s tetracyklínom, rozmnožia sa všetky baktérie, do ktorých prenikol vektor pBR 322, či už s rekombinantnou DNA, alebo bez nej. Ak neskôr takto vyselektované bunky preniesieme do prostredia s ampicilínom, rozmnožovať sa budú len bunky neobsahujúce rekombinantnú DNA, pretože plazmidy, ktoré do nich prenikli, majú neporušený gén rezistencie proti ampicilínu. Dnes sa na selekciu baktérií obsahujúcich rekombinantnú DNA používajú skôr signálne gény, ktoré sa dajú ľahko (napr. farebne) zistiť. Kolónie buniek sa líšia farebne, svietia, alebo rastú na vybranom substráte.



Obrázok 11.5. Technika polymeráznej reťazovej reakcie (PCR). Dvojzávitnicu vzorovej DNA (a) zahriatím rozdelíme na dve komplementárne jednolávkové molekuly DNA (d-e). Po následnom ochladení sa iniciátory (b-c) viažu na vlákna (d) a (e). Enzým Taq polymeráza dosyntetizuje príslušné komplementárne vlákna (f-g), takže z pôvodnej dvojzávitnice DNA (a) dostávame dve identické kópie (h-i).

Význam objavu *polymeráznej reťazovej reakcie* (Polymerase Chain Reaction - *PCR*) sa v genetike porovnáva s významom objavu tlačiarenskeho lisu v renesancii. Technika PCR môže v priebehu niekoľkých minút až hodín znásobiť požadovanú sekvenciu DNA niekoľko stotíľiónov-krát. PCR je založená na termostabilnom enzýme *Taq polymeráze*, ktorý je schopný syntetizovať komplementárny reťazec k danému reťazcu DNA v zmesi obsahujúcej jednak samostatné DNA bázy, jednak oligonukleotidové DNA molekuly dlhé asi 20 báz. Oligonukleotidy hrajú úlohu *iniciátorov* a anglicky sa nazývajú *primers*. Iniciátory tvoria krátke reťazce DNA komplementárne k opačným koncom protilahlých reťazcov dvojzávitnice pôvodnej DNA. Celá zmes sa najprv zohreje, aby sa navzájom separovali lineárne reťazce v dvojzávitnici vzorovej DNA. Neskôr sa zmes ochladí a iniciátory sa naviažu na konce oddelených jednovláknových molekúl DNA. Potom sa pomocou *Taq-polymerázy* syntetizujú komplementárne molekuly k oddeleným reťazcom vzorovej DNA, čím sme dostali dva identické dvojzávitné zdroje DNA. Tento proces sa opakuje, pričom počet syntetizovaných dvojzávitových kópií vzorovej DNA rastie exponenciálne. Za jednu hodinu prebehne asi 20 PCR cyklov, pričom sa počet kópií zväčší asi $2^{20} \approx 10^6$ krát. Na obr. 11.5 je názorné zobrazenie techniky PCR.

Prv ako prikrôčime k príkladom riešenia konkrétnych problémov výpočtami na báze DNA, stručne zhrneme fakty o jednom z najrozšírenejších teoretických modelov symbolických výpočtových systémov (vrátane najmodernejších superpočítačov) - o Turingovom stroji.

11.2 Teoretické modely výpočtového procesu

Konečnostavový automat pozostáva z konečnej množiny stavov a konečného zoznamu *prechodových pravidiel*. Každé prechodové pravidlo má tvar: ak si v stave i a na vstupe máš symbol A , choď do stavu j a na výstup daj symbol B . *Turingov stroj* je konečnostavový automat, ktorý je navyše vybavený nekonečnou *1-dimenzionálnou* zapisovacou páskou, na ktorej môže uchovávať informáciu zapisovaním symbolov do jednotlivých políčok. Samotný Turingov stroj je potom definovaný konečnou množinou pravidiel tvaru: ak si v stave i a na políčku, ktoré práve čítaš, je zapísaný symbol A , potom zmeň svoj stav na j , prepíš symbol A na symbol B a pohni sa na páske o jedno políčko doprava. Pohyb na páske môže byť aj smerom doľava, prípadne aktuálne políčko na páske vôbec nemusíme opustiť. Turingov stroj má špeciálny *koncový stav*, z ktorého už nie je možný prechod do iného stavu. Na začiatku výpočtu je zaplnená len koncová časť pásky (na ostatných políčkach sú tzv. prázdne symboly). Počiatočný obsah pásky reprezentuje "program" pre Turingov stroj.

Hovoríme, že stroj M má *univerzálne výpočtové schopnosti* na nejakej triede výpočtových systémov C , ak ľubovoľný stroj N z triedy C môže byť emulovaný na M jeho vhodným naprogramovaním. Turingove stroje sú univerzálne na triede všetkých konvenčných počítačov - existujú špeciálne *univerzálne Turingove stroje*, ktoré môžu emulovať ľubovoľný iný výpočtový systém, ktorý by sme v súčasnosti nazvali "počítačom".

Nedeterministický Turingov stroj má množinu pravidiel, ktoré nemusia za každých okolností špecifikovať jedinú možnú akciu, ale v určitých súvislostiach ponúkajú viacero možností na výber (napr. po zmene stavu a prepísaní symbolu na páske pohni hlavou doprava alebo doľava).

Trieda problémov, ktoré sú riešiteľné univerzálnym Turingovým strojom za maximálne K krokov, kde K je *polynomiálnou* funkciou počtu bitov na vstupe, sa označuje P . Ak v predošlej vete vymeníme univerzálny Turingov stroj za nedeterministický univerzálny Turingov stroj, dostávame triedu úloh, ktorá býva označená NP . Znamená to, že ak by sme sa vedeli pri nedeterministickom výbere pravidiel "správne" rozhodovať, trval by výpočet len polynomiálny počet krokov vzhľadom na veľkosť vstupu. Problém je však v tom, že väčšinou

nevieme, ktoré pravidlo je v danom momente najvhodnejšie použiť, a tak sa výpočet môže neúmerne predĺžiť (až na *exponenciálny* počet krokov vzhľadom na veľkosť vstupu). Trieda problémov X taká, že každú úlohu z NP možno previesť v polynomiálnom čase na úlohu v X , sa nazýva *NP-úplná*. Vo všeobecnosti sa predpokladá, že trieda P je vlastnou podmnožinou triedy NP a že *NP-úplné* problémy možno vo všeobecnosti riešiť na deterministických Turingových strojoch len za exponenciálny počet krokov. Toto pomerne pesimistické tvrdenie ostáva stále otvorenou otázkou, aj keď väčšina teoretických informatikov verí v jeho pravdivosť.

11.3 Formálna notácia operácií nad DNA

V odbornej literatúre je možné nájsť mnoho rozličných formálnych zápisov štruktúry DNA a operácií nad reťazcami DNA. Keďže počítanie na báze DNA je mladá vedná disciplína, neexistuje v súčasnosti všeobecne akceptovaná notácia pre kompaktný zápis experimentálnych postupov, či formuláciu teoretických výsledkov. V tejto kapitole budeme používať notáciu, ktorú zaviedli traja informatici Dan Boneh, Christopher Dunworth a Richard Lipton z Princetonskej univerzity v USA.

Už sme spomenuli, že molekulu DNA možno považovať za reprezentáciu orientovaného reťazca nad abecedou $x = (x_1, x_2, \dots, x_n)^T$. Formálne je reťazec, alebo slovo x nad abecedou $\Xi = \{A, T, G, C\}$ konečná postupnosť symbolov z Ξ , napr. $x = ACCTGAC$. Množinu všetkých neprázdnych konečných slov nad abecedou Ξ označujeme Ξ^+ . Komplement reťazca x je slovo $\hat{x}$, v ktorom je každý symbol nahradený jeho Watsonovým-Crickovým komplementom. Inými slovami, $\hat{x}$ vznikne tak, že na každý symbol reťazca x aplikujeme zobrazenie $A \rightarrow T$, $C \rightarrow G$, $G \rightarrow C$, $T \rightarrow A$. Reťazec, v ktorom symboly nasledujú v opačnom poradí, ako v slove x , sa označuje x^R .

Pre každý reťazec $x \in \Xi^+$ nad Ξ existuje molekula DNA, v ktorej za sebou nasledujúce bázy v orientácii 5'-3' zodpovedajú symbolom slova x . Takúto molekulu budeme označovať $\uparrow x$. $\downarrow x$ označuje molekulu DNA komplementárnu k molekule $\uparrow x$. Keď sa molekuly $\uparrow x$ a $\downarrow x$ spoja väzbami medzi komplementárnymi bázami, vznikne dvojzávit $\uparrow x$. Táto situácia nastane, ak sú v roztoku molekuly $\uparrow x$ aj $\downarrow x$ a roztok ochladíme. Formálne $\uparrow x + \downarrow x \rightarrow \uparrow x$. Podobne pre $x, y, z \in \Xi^+$ sa molekuly $\uparrow xy$ a $\downarrow yz$ za vhodných podmienok spoja, aby vytvorili čiastočný dvojzávit $\uparrow x \uparrow y \downarrow z$.

Nasledujúci príklad sumarizuje zavedené operátory:

$$\begin{aligned} x &= ACCTGAC \\ \hat{x} &= TGGACTG \\ x^R &= CAGTCCA \\ \uparrow x &= 5' - ACCTGAC - 3' \\ \downarrow x &= 3' - TGGACTG - 5' = \uparrow \left(\hat{x}^R \right) \\ \uparrow \downarrow x &= \begin{matrix} 5' - ACCTGAC - 3' \\ 3' - TGGACTG - 5' \end{matrix} \end{aligned}$$

Nech

$$\begin{aligned} y &= ATCGA \\ z &= TTGAGCT, \end{aligned}$$

potom

$$\uparrow xy = 5' - ACCTGACATCGA - 3'$$

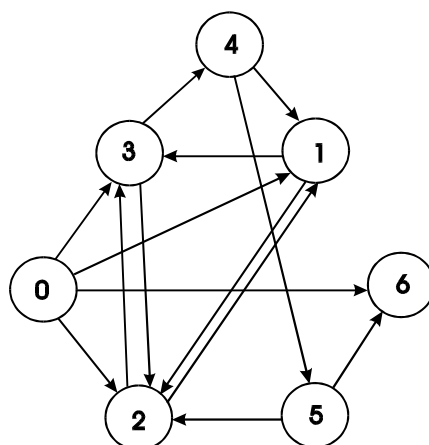
$$\uparrow yz = 3' - TAGCTAACTCGA - 5'$$

a

$$\uparrow xy + \downarrow yz \rightarrow \uparrow x \downarrow y \downarrow z = \begin{array}{c} 5' - ACCTGACATCGA - 3' - - - - - \\ - - - - - 3' - TAGCTAACTCGA - 5' \end{array}$$

11.4 Experiment L. Adlemana - hľadanie hamiltonovskej cesty v orientovanom grafe na báze DNA

Matematika ako vedná disciplína prenikla do mnohých vedných odborov a v ich rámci prispela k jasnej formulácii problémov a mnohokrát aj k ich elegantnému riešeniu. Výnimkou nie je ani biológia. Čitateľa, ktorý by sa chcel dozvedieť viac o matematickom modelovaní biologických procesov, odkazujeme napr. na publikáciu [1]. L. Adleman však túto situáciu obrátil naruby. Využil biologické procesy na simuláciu matematických výpočtov [2]. Podstata jeho prístupu spočíva v kódovaní informácie na reťazcoch DNA a vo využití enzýmov na simuláciu jednoduchých výpočtov.



Obrázok 11.6. Orientovaný graf použitý v pokuse L. Adlemana nájsť výpočtami na báze DNA hamiltonovskú cestu z uzla 0 do uzla 6.

L. Adleman sa podujal prakticky ukázať použiteľnosť svojej teórie na testovaní existencie hamiltonovskej cesty v orientovanom grafe. Problém existencie hamiltonovskej cesty v orientovaných grafoch je NP-úplný. Znamená to, že dosiaľ sa nenašiel (a ako sme uviedli, mnoho informatikov verí, že sa ani nikdy nenájde) algoritmus, ktorý by pre ľubovoľný orientovaný graf G a pre jeho ľubovoľné dva vrcholy A, B , rozhodol po N krokoch, či v G existuje cesta z A do B , pričom každý vrchol grafu G by bol navštívený práve raz. Počet N krokov algoritmu pritom musí byť polynomiálnou funkciou počtu vrcholov v G . Voľne povedané - overovanie existencie hamiltonovskej cesty v orientovanom grafe je vo všeobecnosti výpočtovo náročná úloha. Počet krokov potrebných na vyriešenie problému rastie exponenciálne s veľkosťou grafu. Treba však povedať, že NP-úplné problémy robia starosti informatikom len pri úlohách väčšieho rozsahu a L. Adleman si pre svoj experiment

zvolil pomerne jednoduchý graf na obr. 11.6 so štartovacím uzlom 0 a koncovým uzlom 6. Hamiltonovská cesta je $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$.

Uvedme si najprv hrubý náčrt algoritmu overujúceho existenciu hamiltonovskej cesty z uzla A do uzla B v orientovanom grafe G s n vrcholmi:

1. generuj náhodné cesty v grafe G ,
2. ponechaj si len cesty vychádzajúce z uzla A a končiace v uzle B ,
3. vylúč všetky cesty, ktorých dĺžka je rôzna od n ,
4. ponechaj si len tie cesty, ktoré navštívia každý vrchol najviac raz,
5. ak je množina zostávajúcich ciest neprázdna, daj kladnú odpoveď, ináč odpovedz záporne.

Teraz si opíšeme, ako sa dajú jednotlivé kroky takéhoto algoritmu realizovať pomocou molekúl DNA a operácií nad nimi.

1. Každý uzol v grafe G stotožníme s náhodne zvoleným reťazcom $O(v)=L(v)R(v)$ dĺžky 20, ktorý sa skladá z dvoch za sebou nasledujúcich reťazcov $L(v)$, $R(v)$ dĺžky 10. Oligonukleotid kódujúci vrchol v je potom $\uparrow O(v)$. Každú orientovanú hranu z vrcholu p do vrcholu q zakódujeme úsekom DNA $\uparrow R(p)L(q)$. Orientované hrany z počiatočného vrcholu v_{INI} kódujeme oligonukleotidmi $O(v_{INI})L(q)$ dĺžky 30. Podobne kódujeme hrany do koncového vrcholu v_{END} oligonukleotidmi $R(p)O(v_{END})$. Za sebou nasledujúce hrany tvoriace cestu v grafe G spájame pomocou komplementárnych kódov $\downarrow O(v)$ vrcholov, cez ktoré cesta prechádza. Napríklad ak v grafe G existujú hrany z vrcholu p do vrcholu q a z vrcholu q do vrcholu r , potom reprezentácia cesty $p \rightarrow q \rightarrow r$ vznikne zviazaním kódov $\uparrow R(p)L(q)$ a $\uparrow R(q)L(r)$ hrán $p \rightarrow q$ a $q \rightarrow r$ s komplementárnymi kódmi, $\downarrow O(p)=\downarrow L(p)\downarrow R(p)$, $\downarrow O(q)=\downarrow L(q)\downarrow R(q)$ a $\downarrow O(r)=\downarrow L(r)\downarrow R(r)$ vrcholov p , q a r :

$$\begin{aligned} &\downarrow L(p)\downarrow R(p)+\uparrow R(p)\uparrow L(q)+\downarrow L(q)\downarrow R(q)+\uparrow R(q)\uparrow L(r)+\downarrow L(r)\downarrow R(r)= \\ &\downarrow L(p)\uparrow R(p)O(q)L(r)\downarrow R(r) \end{aligned}$$

2. Použitím DNA-ligázy možno pospájať za sebou nasledujúce hrany a vygenerovať v roztoku molekuly DNA, ktoré kódujú cesty v grafe G . Ak dáme do roztoku dostatočné množstvo molekúl, ktoré kódujú hrany a molekúl, ktoré komplementárne kódujú vrcholy grafu G (L. Adleman pridal do roztoku 50 pmol z každého typu molekúl DNA), existuje možnosť, že medzi vygenerovanými cestami bude aj hamiltonovská cesta.
3. Molekuly kódujúce cesty v grafe G , ktoré boli vygenerované v predošlom kroku, znásobíme technikou PCR použitím iniciátorov $\uparrow O(v_{INI})$ a $\downarrow O(v_{END})$. Veľký počet kópií vznikne len z molekúl, ktoré kódujú cesty začínajúce v v_{INI} a končiace v v_{END} .
4. Produkty kroku 3. zoradíme podľa veľkosti metódou gélovej elektroforézy (molekulového sita). Molekuly DNA naniesieme na štartovacie miesto agarózneho gélu, na ktorý pôsobí elektrické pole. Vplyvom poľa sú molekuly DNA priťahované smerom k čelu. Dlhšie molekuly putujú gélom pomalšie ako kratšie a po určitom čase sa molekuly distribuujú v prostredí v jasne detekovateľných pásoch. Pásma jednoznačne odpovedajú dĺžkam molekúl. Napríklad pre Adlemanov graf na obr. 11.6 bolo treba detekovať pásmo molekúl dĺžky 140 bázových párov kódujúcich cesty v grafe G dĺžky 7. Molekuly z príslušného pásma extrahujeme a celý postup niekoľkokrát opakujeme, aby sme dosiahli čo najpresnejšiu selekciu molekúl požadovanej dĺžky.

5. Iteratívne aplikujeme proces afinitného prečisťovania (affinity purification). Najprv zahriatím roztoku rozpojíme dvojzavíty kódujúce cesty v grafe G na jednovláknové molekuly DNA. Potom pre každý vrchol v v grafe G opakujeme tento postup: pre vrchol v v grafe G vygenerujeme kópie jeho komplementárnej reprezentácie $\downarrow O(v)$ a pripojíme ich na magnetické častice (presnejšie, $\downarrow O(v)$ označujeme molekulami biotínu; biotínové molekuly sa zas viažu na magnetické častice s avidínom). Molekuly vyselektované v predošlom kroku necháme reasociovať so sekvenciami na magnetických časticiach. Na magnetické častice sa naviažu len molekuly kódujúce cesty prechádzajúce cez vrchol v . Ostatné molekuly z roztoku odplávime.
6. Prítomnosť molekuly kódujúcej hamiltonovskú cestu v grafe G overíme znásobením produktov doterajších krokov pomocou procedúry PCR a následným použitím techniky gélovej elektroforézy.

V čom spočíva hlavný prínos myšlienky L. Adlemana využiť molekuly DNA ako výpočtové médium [3]? Teoreticky môžu molekulárne počítače prekonať existujúce superpočítače. Operácie, ktoré je možné vykonať nad molekulami DNA pomocou metód molekulárnej biológie, sa môžu využiť pri masívnom paralelnom prehľadávaní obrovských priestorov riešení. Odhaduje sa, že molekulárne počítače by prekonal existujúce počítače nielen svojou rýchlosťou, ale aj ekonomikou uchovávaní informácie a malou energetickou spotrebou. Adlemanov model je schopný vykonať asi 1.2×10^{18} operácií za sekundu. To je asi 1200000-krát viac ako najrýchlejší superpočítač. Dnešné superpočítače spotrebujú jeden joule na vykonanie 10^9 operácií. Energetická spotreba molekulárneho počítača je asi 2×10^{19} operácií na jeden joule. Znamená to, že molekulárny počítač by mohol byť asi 10^{10} -krát efektívnejší ako moderné superpočítače. Pritom molekulárne počítače potrebujú na uchovanie jedného bitu informácie asi jeden kubický nanometer, zatiaľ čo existujúce pamäťové médiá uchovávajú informáciu s hustotou asi 1 bit na 10^{12} kubických nanometrov.

Popri vlne nadšenia, ktorú Adlemanov pokus vyvolal, sa ozvali aj kritické hlasy spochybňujúce jeho optimistické vízie éry molekulárnych výpočtových systémov. V čom je problém? Praktická realizácia operácií nad DNA naráža na mnohé úskalía, ktoré sú profesionálnym genetikom úplne zrejmé, no pre informatikov ostávajú skryté. Konkrétny problém, ktorý L. Adleman riešil, je totiž naozaj jednoduchý a človek je schopný nájsť jeho riešenie po chvíli uvažovania. Adlemanovi stačilo zakódovať 7 vrcholov orientovaného grafu G náhodne vygenerovanými oligonukleotidmi dĺžky 20. Intuitívne je zrejmé, že takéto zakódovanie problému bude v poriadku okrem situácií, ktoré sú málo pravdepodobné. Ak by sme však mali nájsť hamiltonovskú cestu v podstatne väčšom grafe, budeme potrebovať dlhšie kódy pre vrcholy, a tým vzrastie pravdepodobnosť chýb prvého a druhého druhu (teda zistenia falošných hamiltonovských ciest alebo nezistenia existujúcich hamiltonovských ciest). Dlhšie kódy pre vrcholy prinášajú riziko nedokonalých alebo zle usmernených spájaní komplementárnych úsekov DNA. Môže sa dokonca stať, že vlákno DNA sa ohne a vytvorí na určitom úseku dvojzávit samo so sebou. Takáto situácia nastane, ak molekula DNA obsahuje dve dostatočne dlhé podpostupnosti báz U a V také, že $\uparrow U = \downarrow V$. Takýmto sekvenciám sa hovorí *palindrómy*. Deformované vlákno DNA pripomína sponku do vlasov (hairpin match).

Problém vhodných kódov pre model L. Adlemana analyzovali R. Deaton a spolupracovníci v [4]. Hlavnou myšlienkou ich prístupu je použiť také kódy $O(v) = L(v)R(v)$ pre vrcholy $v \in V(G)$ grafu G , aby sa molekuly z množiny

$$\{\uparrow L(v) \mid v \in V(G)\} \cup \{\downarrow L(v) \mid v \in V(G)\} \cup \{\uparrow R(v) \mid v \in V(G)\} \cup \{\downarrow R(v) \mid v \in V(G)\},$$

kde $V(G)$ je množina vrcholov grafu G , odlišovali v najmenej t bázach. Parameter t závisí od dĺžky oligonukleotidov N a od podmienok, za ktorých sa výpočet realizuje.

Iný prístup k vhodnému kódovaniu informácie, ktorý by obišiel problém palindrómov a zabezpečil dostatočnú rôznosť kódujúcich sekvencií, možno nájsť v [5]. Postupnosti báz v kódujúcich molekulách DNA zodpovedajú modifikovaným DeBruijnovým sekvenciám.

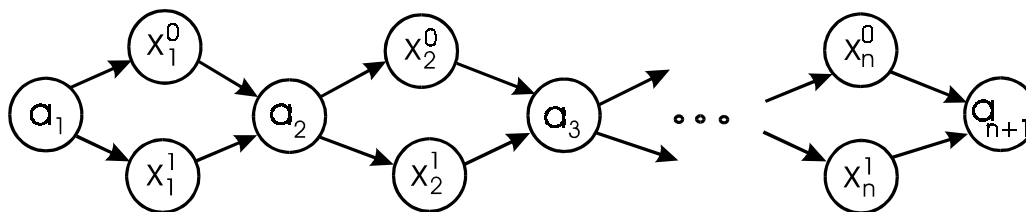
11.5 R. Lipton - Určenie splniteľnosti logických formúl na báze DNA

Americký matematik R. J. Lipton z Princetonskej univerzity rozšíril prácu L. Adlemana a postavil ju na pevnejší formálny základ [6,7]. Sústredil sa na riešenie ďalšieho NP-úplného problému - určenia splniteľnosti logickej formuly (Formula Satisfaction Problem). Formálne možno tento problém vyjadriť takto: máme danú (potenciálne ľubovoľnú) logickú formulu $F(x_1, \dots, x_n)$ dĺžky s . Rozhodnite, či existujú také pravdivostné hodnoty boolovských premenných $x_1, \dots, x_n$, že formula $F(x_1, \dots, x_n)$ je pravdivá. Takémuto problému sa hovorí aj (n,s) -problém splniteľnosti logickej formuly.

R. Lipton sa rozhodol vybudovať všeobecnejšiu stratégiu výpočtov na báze DNA, pretože cítil, že postup L. Adlemana je "šitý na mieru" problému hamiltonovskej cesty a nie je priamo použiteľný pre riešenie akejkoľvek NP-úplnej úlohy. Samozrejme, jednou z možností by bolo daný NP-úplný problém transformovať na konvenčnom počítači (v polynomiálnom čase) na problém existencie hamiltonovskej cesty v orientovanom grafe a tento vyriešiť na báze DNA. Takýto prístup by však nebol elegantný a neposunul by výskum v oblasti výpočtov na báze DNA na systematickejšiu platformu. Jedným z prínosov R. Liptona bolo, že ukázal ako postupovať pri implementácii hierarchicky organizovaného DNA počítača, ktorý volá podprogramy realizované DNA počítačmi na nižšej úrovni.

Formálne možno skúmavku s molekulami DNA považovať za konečnú multimnožinu orientovaných reťazcov nad abecedou $\{A, C, G, T\}$. Multimnožina (multiset) je súbor prvkov, kde spolu s prvkom patriacim do množiny udržíme aj informáciu o počte kópií daného prvku v množine. Ak sa obrátíme na multimnožinu a použijeme z nej pre nejakú operáciu prvok x , potom početnosť prvku x znížime o jednotku. Klasické množiny sú teda multimnožiny, ktorých prvky sú k dispozícii v neobmedzene veľkom počte kópií.

Liptonov počítač na báze DNA pracuje so skúmavkami obsahujúcimi DNA, nad ktorými sú povolené určité operácie. Výpočet sa inicializuje vždy s jednou skúmavkou, navyše tou istou pre všetky výpočty. Obsah počiatočnej skúmavky zodpovedá grafu G na obr. 11.7.



Obrázok 11.7. Orientovaný graf, ktorého cesty z uzla a_1 do uzla a_{n+1} určuje obsah počiatočnej skúmavky. Sekvencie a_j predstavujú oddelovače umiestnené medzi sekvencie x_j^0, x_j^1 kódujúce hodnoty jednotlivých bitov n -bitového binárneho čísla $b_1b_2\dots b_n$.

Molekuly DNA nachádzajúce sa v počiatočnej skúmavke zodpovedajú cestám v grafe G s počiatočným uzlom a_1 a koncovým uzlom a_{n+1} . Počiatočné molekuly sa generujú technikou použitou L. Adlemanom na hľadanie hamiltonovských ciest v orientovanom grafe. Tentoraz však negenerujeme cesty začínajúce v uzle a_1 , končiace v uzle a_{n+1} a prechádzajúce *každým* vrcholom práve raz, ale požadujeme len cesty v grafe G z a_1 do a_{n+1} . Každá takáto cesta γ reprezentuje n -bitové binárne číslo $b_1b_2\dots b_n$, kde hodnota bitu $b_i \in \{0,1\}$ je $b_i = 0$, ak

cesta γ prechádza cez uzol x_i^0 , a $b_i = 1$, ak γ prechádza cez x_i^1 . Molekuly DNA reprezentujúce uzly a_i slúžia ako oddelovače jednotlivých bitov binárneho čísla.

Nad skúmvkami sú povolené tieto operácie:

1. *Extrahuj* - zo skúmvky t extrahuj všetky molekuly DNA, ktoré obsahujú danú podpostupnosť báz.
2. *Zisťuj* - zisti, či sa v skúmvke t nachádza nejaká molekula DNA alebo nie.
3. *Znásob* - znásob počet kópií molekúl DNA nachádzajúcich sa v skúmvke t .

S takýmito operáciami nad skúmvkami obsahujúcimi molekuly DNA sme sa už stretli pri opise pokusu L. Adlemana: operácia *extrahuj* vybrala iba tie molekuly DNA, ktoré prechádzali daným uzlom grafu, operáciou *znásob*, realizovanou technikou PCR, sme získali mnoho kópií molekúl kódujúcich cesty zo začiatočného do koncového vrcholu grafu, operácia *zisťuj* dala konečnú odpoveď na otázku existencie hamiltonovskej cesty v danom orientovanom grafe.

Označme symbolom $E(t, i, b)$ všetky molekuly v skúmvke t kódujúce čísla, ktorých i -ty bit má hodnotu $b \in \{0, 1\}$. Takéto molekuly možno získať operáciou *extrahuj* nad t , pričom kľúčová podpostupnosť bude postupnosť báz v molekule DNA kódujúcej vrchol x_i^0 , ak $b_i = 0$, alebo postupnosť báz v DNA kóde pre x_i^1 , ak $b_i = 1$.

Dokážme si najprv jednoduchšie tvrdenie o riešení problému určenia splniteľnosti formúl v konjunktívnom normálnom tvare. Logická formula je v konjunktívnom normálnom tvare, ak je tvorená konečným počtom klauzúl spojených logickým súčinom (konjunkciou), pričom každá klauzula predstavuje logický súčet konečného počtu literálov.

Veta 1: Nech F je logická formula n premenných, ktorá je v konjunktívnom normálnom tvare. Nech počet klauzúl formuly F je c . Potom problém splniteľnosti formuly F je výpočtom na báze DNA riešiteľný za maximálne $O(c)$ krokov *extrahuj* (teda počet krokov *extrahuj* je lineárnou funkciou počtu klauzúl c) a jeden krok *zisťuj*.

Dôkaz. Nech $C_1, \dots, C_c$ sú klauzuly formuly F . Skonstruujeme takú postupnosť skúmviek $t_0, \dots, t_c$, že v skúmvke t_k sa nachádzajú molekuly DNA kódujúce n -bitové čísla x , pre ktoré $C_1 = C_2 = \dots = C_k = 1$.

V počiatočnej skúmvke t_0 sa nachádzajú molekuly reprezentujúce každé n -bitové číslo. Konštrukciu urobíme indukciou. Predpokladajme, že skúmvku t_k sme už skonštruovali, ukážeme si konštrukciu skúmvky t_{k+1} . Nech klauzula C_{k+1} má tvar $l_1 \vee l_2 \vee \dots \vee l_m$, kde každé l_i je buď literál, alebo jeho komplement. Pre každé l_i vytvor skúmvku S_i obsahujúcu molekuly z $E(t_k, j, 1)$, ak $l_i = x_j$, alebo $E(t_k, j, 0)$, ak $l_i = \bar{x}_j$. Skúmvku t_{k+1} získame zliatím obsahov skúmviek $S_1, \dots, S_m$.

Posledný krok algoritmu predstavuje operácia *zisťuj* na skúmvke t_c .

Veta 2: Každý (n, s) -problém splniteľnosti logickej formuly je výpočtom na báze DNA riešiteľný za maximálne $O(s)$ krokov *extrahuj* a jeden krok *zisťuj*.

Dôkaz. Pre boolovskú formulu f s n premennými a skúmvku t obsahujúcu molekuly kódujúce n -bitové čísla definujme (multi)množinu $TT(f, t) = \{x \in t \mid f(x) = 1\}$. Naším cieľom je skonštruovať $TT(F, t_0)$, kde t_0 je počiatočná skúmvka obsahujúca kópie všetkých n -bitových čísel. Ukážeme si, ako "vypočítať" TT . Môžu nastať štyri možnosti:

1. $f = x_i$. V tomto prípade $TT(x_i, t) = E(t, i, 1)$.

2. $f = \bar{x}_i$. Potom $TT(x_i, t) = E(t, i, 0)$.
3. $f = F_1 \vee F_2$. Najprv vytvor $t_1 = TT(F_1, t)$ a $t_2 = t - TT(F_1, t) = t - t_1$. V skúmvke t_2 sa nachádzajú molekuly kódujúce n -bitové čísla, ktoré nespĺňajú formulu F_1 . Ďalej skonštruuj $t_3 = TT(F_2, t_2)$ a $t_4 = t_2 - t_3$. Výsledok dostaneme zliatím skúmviek t_1 a t_3 ($t_1 \cup t_3$), pričom sa molekuly nevyhovujúcu formule $F_1 \vee F_2$ nachádzajú v skúmvke t_4 .
4. $f = F_1 \wedge F_2$. Utvor $t_1 = TT(F_1, t)$ a $t_2 = t - t_1$. Potom skonštruuj $t_3 = TT(F_2, t_1)$ a $t_4 = t_1 - t_3$. Výsledok je v skúmvke t_3 a nevyhovujúce vstupy sú v $t_2 \cup t_4$.

Nie je ťažké vidieť, že počet krokov potrebných na zodpovedanie otázky splniteľnosti ľubovoľnej boolovskej formuly je lineárnou funkciou jej veľkosti.

Dôkaz by sa ešte zjednodušil, ak by sme povolili aj operáciu *znásob*. Prípady 1. a 2. by ostali nezmenené, ale v prípade 3. by stačilo vykonať logické *alebo* $TT(F_1, t) \cup TT(F_2, t)$, keďže máme možnosť vyrobiť si kópie skúmvky t . Prípad 4. by sme vyriešili ako $TT(F_1, TT(F_2, t))$.

Už sme spomínali, že R. Lipton navrhol aj mechanizmus volania podprogramov na báze DNA z hierarchicky vyšších výpočtov na báze DNA. Načrtnime základnú ideu tohoto mechanizmu.

Predpokladajme, že máme za úlohu prehľadať exponenciálne veľký prehľadávací priestor (napríklad množinu všetkých m -bitových čísel pre dostatočne veľké m) a máme nájsť prvok x spĺňajúci daný predikát P . Prehľadávací algoritmus založený na hrubej sile by mohol mať tvar:

pre každé $x = 1, 2, \dots, 2^m$
 ak $P(x)$ potom
 vráť x

Predpokladajme tiež, že výpočet predikátu P nie je "drahý" a je vykonateľný v polynomiálnom čase. Môže nám nejaký pomôcť fakt, že výpočtom na báze DNA sme schopní vyriešiť problém splniteľnosti logickej formuly v lineárnom čase? Ak by sme pre každé x generovali len prvých $m - n$ bitov ($m > n$) a skonštruovali formulu F , ktorá je pravdivá vtedy a len vtedy, ak nejaké x má začiatočných $m - n$ bitov zhodných s danou $(m - n)$ -ticou a súčasne vyhovuje predikátu P , stačilo by hrubou silou prehľadávať len potenciálne omnoho menší priestor veľkosti 2^{m-n} . Zvyšných n bitov každého prvku x by sme dokázali otestovať v lineárnom čase v procese testovania splniteľnosti formuly F . Ak berieme do úvahy polynomiálnu časovú zložitosť výpočtu predikátu P , prehľadávací proces by sme boli schopní ukončiť za $O(n^c 2^{m-n})$ krokov.

11.6 Počítače na báze DNA sú univerzálne výpočtové systémy

Ďalším krokom v systematickom štúdiu výpočtov na báze DNA bolo určenie typu úloh, ktoré sú schopné riešiť. Otázka znela: Sú počítače na báze DNA v princípe rovnako schopné ako najmodernejšie superpočítače, alebo existujú úlohy, ktoré je teoreticky možné vyriešiť na súčasných počítačoch, no výpočet na báze DNA by nikdy k výsledku nedospel? Poznamenávame, že pri otázkach tohoto typu nás viac ako konkrétne pamäťové a časové zložitosti algoritmov na jednotlivých výpočtových médiách zaujíma, či architektúra tohto ktorého výpočtového systému v *princípe* umožňuje daný typ úlohy riešiť alebo nie.

V tejto kapitole si ukážeme, že v *princípe* neexistuje žiadny problém riešiteľný na konvenčných počítačoch, ktorý by nebol riešiteľný aj výpočtami na báze DNA. Z hľadiska teoretickej informatiky možno túto skutočnosť formulovať aj takto [5]:

Veta 3: *Výpočty na báze DNA sú schopné simulovať akýkoľvek nedeterministický Turingov stroj.*

Dôkaz. Existuje niekoľko verzií dôkazu vety 3. My si ukážeme dôkaz, ktorý prezentovali W.D. Smith a A. Schweitzer z výskumného ústavu firmy NEC v Princetone, USA, na prvom oficiálnom pracovnom stretnutí pracovníkov zaoberajúcich sa výpočtami na báze DNA [5].

Dôkaz spočíva v priamej konštrukcii Turingovho stroja s využitím molekúl DNA ako pamäťového média a enzymatických operácií umožňujúcich transformácie medzi jednotlivými konfiguráciami Turingovho stroja. Turingov stroj bude reprezentovaný kruhovou molekulou DNA, kde sa úseky jednovláknovej DNA striedajú s úsekmi dvojzávitú. Väčšina takejto "páskovej" molekuly DNA je rozdelená na krátke úseky (dĺžky napr. 20 báz), ktoré reprezentujú jednotlivé symboly vyskytujúce sa na páske simulovaného Turingovho stroja. Pochopiteľne, v praxi bude dĺžka reťazcov reprezentujúcich páskové symboly závisieť od ich počtu a voľba samotných sekvencií musí obísť problémy spomínané v kapitole 11.4.

Sekvencie kódujúce páskové symboly sú oddelené "čiarkami" •, ktoré sú kódované špeciálnou postupnosťou báz dĺžky 20.

Páska nie je nekonečná, ale má konečnú dĺžku a je kruhová. Model Turingovho stroja na báze DNA však skonštruujeme tak, aby si mohol vyrobiť viac páskového priestoru, ak to bude potrebné. Na pozícii protiľahlej k počiatočnému políčku pásky je umiestnená špeciálna postupnosť báz, ktorá signalizuje "koniec" pásky. Touto sekvenciou je zároveň pevne pripojená na špecifické miesto v skúmvke kruhová molekula DNA biotín-avidínovou technikou, ktorú použil L. Adleman v kroku 4. algoritmu hľadania hamiltonovskej cesty v orientovanom grafe (kapitola 11.4).

Sekvenciami na kruhovej molekule DNA kódujeme aj polohu hlavy Turingovho stroja. Konkrétne, podpostupnosti báz na DNA reprezentujú stav H hlavy a symbol L nachádzajúci sa bezprostredne naľavo od čítacej hlavy stroja. Za sekvenciou H nasleduje špeciálna postupnosť $>$ rozpoznávaná reštrikčným enzýmom, ktorý rozstrihne kruhovú molekulu DNA práve za sekvenciou kódujúcou stav hlavy Turingovho stroja.

Krok 1. Použij reštrikčný enzým na rozstrihnutie DNA napravo od špeciálnej postupnosti báz označenej symbolom $>$. Tento krok transformuje kruhovú DNA na lineárnu molekulu s "lepkavými" koncami. Vypláchnutím skúmvky odstránime enzým a molekula DNA ostane pripevnená k stene biotín-avidínovou väzbou. Zahriatím roztoku roztrhneme väzby v dvojzávitoch a utvoríme izolované vlákno DNA. Zvyšky DNA opäť odstránime jednoduchým vypláchnutím skúmvky.

Reštrikčný enzým *EcoR II* je jedným z vhodných kandidátov pre realizáciu tohto kroku. *EcoR II* strihá molekulu na konci špecifickej postupnosti $\uparrow \geq \uparrow GGWCC$ (resp. $\downarrow >$), kde $W \in \{A, T\}$. Teda rozstrihnutím dvojzávitú $\uparrow S > Q$, ($S, Q \in \{A, T, G, C\}^+$) vzniknú dve jednovláknové molekuly $\uparrow S >$ a $\downarrow > Q$. Čas potrebný na realizáciu kroku je asi 10-100 minút.

Krok 2. Do pracovnej skúmvky pridáme zmes jednovláknových molekúl DNA, ktoré kódujú prechody medzi konfiguráciami nachádzajúce sa v prechodovej tabuľke Turingovho stroja. Molekuly kódujúce pravidlo - ak je čítacia hlava v stave H a číta páskový symbol L , potom zmeň stav hlavy na H^* a prepíš symbol L symbolom L^* - majú tvar $\downarrow \bullet L \bullet H > \bullet L^* \bullet H^* > \bullet$. Ak by sme chceli posunúť hlavu o jedno políčko doľava, mali by kódujúce molekuly tvar $\downarrow \bullet L \bullet H > \bullet H^* > \bullet L^* \bullet$.

Molekuly kódujúce prechody medzi konfiguráciami Turingovho stroja sa viažu na oba rozstrihnuté konce "páskovej DNA" sekvenciami $\downarrow \bullet L \bullet H >$ a $\downarrow \bullet$. Takýmto spôsobom bude pásková DNA opäť kruhová. Na túto operáciu budeme potrebovať enzým *ligázu* a molekuly *adenozíntrifosfátu* (ATP), ktoré v bunkách fungujú ako donor energie.

Molekula DNA veľkosti fágu lambda (48514 básových párov) môže byť ligázou uvedená do kruhového tvaru za 20 minút. Viazanie molekúl kódujúcich prechodové pravidlá na páskové molekuly DNA (ktorých môže byť v jednej skúmavke niekoľko, najmä ak implementujeme nedeterministický Turingov stroj) by podľa odhadov autorov dôkazu nemalo trvať dlhšie ako hodinu.

Krok 3. Pomocou enzýmu DNA-polymerázy doplníme chýbajúce články na páskovej molekule DNA, takže dostaneme úplný kruhový dvojzávit DNA. *E. Coli* DNA-polymerázy I, II a III pracujú rýchlosťami asi 10, 0.5 a 150 báz za sekundu. *Taq*-polymeráza bežne používaná v technike PCR je rýchlejšia, ale za cenu väčšej chybovosti pri syntetizovaní komplementárnych úsekov DNA.

Krok 4. Zvýšením teploty v reakčnej zmesi denaturujeme páskovú DNA a z kruhového dvojzávit opäť dostaneme jednovláknovú kruhovú molekulu DNA.

Krok 5. Do pracovnej skúmavky pridáme molekuly DNA kódujúce povolené transformácie medzi konfiguráciami Turingovho stroja ako v kroku 2. Tentoraz však nebudú "transformačné" molekuly končiť $\bullet$ čiarkou, ale budú mať skrátený tvar $\downarrow \bullet L \bullet H > \bullet L^*$, $\downarrow \bullet L \bullet H > \bullet H^*$. Po vzniku príslušných väzieb bude pásková DNA stále jednovláknová kruhová DNA, ale na pozícii čítacej hlavy bude mať úsek tvorený dvojzávitom.

Krok 6. Pomocou restriktívneho enzýmu, ktorý strihá molekulu DNA po rozpoznaní špecifickej postupnosti báz, ktorá sa nachádza jedine v reťazci kódujúcom "čiarku" $\bullet$, odstráni úsek DNA kódujúci pôvodný stav čítacej hlavy H a pôvodne čítaný symbol L . "Lepkavé" konce na rozstrihnutých koncoch páskovej DNA je možné opäť spojiť do kruhovej molekuly použitím enzýmu ligázy.

Poznamenávame, že v kroku 1 bolo naším cieľom dostať jednovláknovú verziu rozstrihutej páskovej DNA. Preto sme bezprostredne po použití restriktívneho enzýmu rozstrihnutú molekulu zahriali a vlákno nepripojené k stene pracovnej skúmavky odplavili z roztoku preč.

Krok 7. Použitím DNA-polymerázy možno páskovú DNA dosyntetizovať na kompletný kruhový dvojzávit.

Pre názornosť si teraz stručne zhrnieme 7-krokovú procedúru realizujúcu jeden prechod Turingovho stroja:

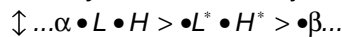
Krok 0. Vlož do skúmavky páskovú DNA $\uparrow \dots \alpha \bullet L \bullet H > \bullet \beta \dots$, kde $\alpha, \beta \in \{A, C, T, G\}^+$.

Krok 1. Rozstrihni páskovú DNA na špecifickom mieste $>$, zohrej skúmavku a odplav voľné vlákna DNA. Vzniknú dva voľné "lepkavé" konce $\uparrow \dots \alpha \bullet L \bullet H >$ a $\uparrow \bullet \beta \dots$ jednovláknovej DNA.

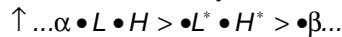
Krok 2. Pridaj molekuly kódujúce konfiguračné transformácie Turingovho stroja.

$\uparrow \dots \alpha \bullet L \bullet H > + \downarrow \bullet L \bullet H > \bullet L^* \bullet H^* > \bullet + \uparrow \bullet \beta \dots \rightarrow \uparrow \dots \alpha \uparrow \bullet L \bullet H > \downarrow \bullet L^* \bullet H^* > \uparrow \bullet \beta \dots$

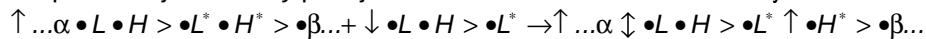
Krok 3. Použi DNA-polymerázu na dosyntetizovanie dvojzávitú páskovej DNA



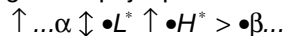
Krok 4. Zohrej páskovú DNA a v skúmavke nechaj iba izolované vlákno



Krok 5. Do pracovnej skúmavky pridaj skrátene "transformačné" molekuly DNA



Krok 6. Rozstrihni páskovú DNA v dvojzávitovom úseku na špecifickom mieste uprostred sekvencie kódujúcej "čiarku" • a ligázou spoj lepkavé konce do kruhovej DNA



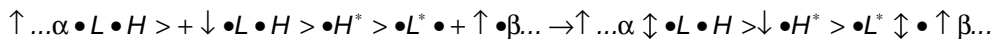
Krok 7. DNA polymerázou dosyntetizuj páskovú DNA na dvojzávit $\updownarrow \dots \alpha \bullet L^* \bullet H^* > \bullet \beta \dots$

Teraz si uveďme procedúru pre prípad, ak Turingov stroj nielen prepisuje páskový stroj a mení vnútorný stav čítacej hlavy, ale po vykonaní týchto úkonov aj posúva čítaciu hlavu o jedno políčko doľava.

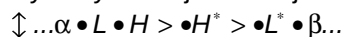
Krok 0. Pásková DNA: $\updownarrow \dots \alpha \bullet L \bullet H > \bullet \beta \dots$, kde $\alpha, \beta \in \{A, C, T, G\}^+$.

Krok 1. Po rozstrihnutí páskovej DNA na špecifickom mieste > a zohriatí skúmavky odplav voľné vlákna DNA. Vzniknú tak dva "lepkavé" konce $\uparrow \dots \alpha \bullet L \bullet H >$ a $\uparrow \bullet \beta \dots$ jednovláknovej DNA.

Krok 2. Pridaním molekúl kódujúcich konfiguračné transformácie Turingovho stroja vznikne molekula DNA



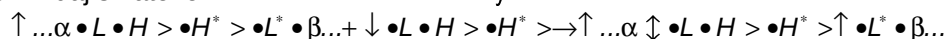
Krok 3. Pomocou DNA-polymerázy dosyntetizujeme dvojzávit páskovej DNA



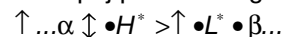
Krok 4. Zahriatím skúmavky a následným odplavením voľných molekúl ostane v skúmavke vlákno



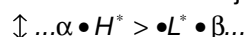
Krok 5. Pridaj skrátene "transformačné" molekuly DNA



Krok 6. Rozstrihni páskovú DNA v dvojzávitovom úseku na špecifickom mieste uprostred sekvencie kódujúcej "čiarku" •. Potom spoj pomocou ligázy lepkavé konce do kruhovej DNA



Krok 7. Polymerázou dosyntetizuj páskovú DNA na dvojzávit



Poznamenávame, že dosiaľ sme popísali len jednosmerný Turingov stroj, ktorý posúva hlavu len doľava. Predpokladáme, že čitateľ si už teraz dokáže predstaviť mechanizmus posunu hlavy doprava. Stačí rozlíšiť špeciálne stavy, ktoré po prepísaní páskového symbolu a zmene vnútorného stavu hlavy (7-krokovou procedúrou uvedenou vyššie) deklarujú, že chceme posun hlavy o jedno políčko doprava. Implementácia samotného posuvu hlavy doprava je veľmi podobná implementácii posuvu hlavy doľava.

Autori konštrukčného dôkazu, ktorý sme si práve uviedli, W. Smith a A. Schweitzer, upozorňujú na praktické obmedzenia, ktoré je potrebné spomenúť pri každej teoretickej práci o počítačoch na báze DNA. Napríklad v súčasnosti je možné vyrobiť umelé vlákna DNA dĺžky len asi 100 báзовých párov s chybovosťou asi 1%. Cena výroby oligonukleotidov je asi 2\$ za bázu. Treba poznamenať, že metódy syntézy vlákien DNA sú v centre pozornosti výskumu molekulárnej biológie a je možné očakávať podstatné zlepšenie dĺžky, kvality a cenových relácií syntetizovaných molekúl DNA. Napríklad M. Kálmán navrhuje nasledujúci postup syntézy DNA [8]:

1. konvenčným postupom syntetizujú kratšie úseky DNA
2. syntetizované úseky spoj ligázou do dlhej jednovláknovej molekuly DNA
3. jednovláknovú molekulu enzymaticky dosyntetizujú na dvojzávit DNA
4. dvojzávit DNA znásob in vivo v baktériách postupom uvedeným v kapitole 11.1.

Kálmánovmu kolektívu sa takto podarilo umelo vytvoriť molekulu DNA dlhú 1761 báзовých párov s chybovosťou asi 0.5% na báзовý pár.

11.7 Počítače na báze DNA in vivo

V tejto kapitole si uvedieme dva príklady rýchlo sa rozvíjajúcej symbiózy dvoch vedných disciplín - informatiky a molekulárnej biológie. Pôjde o nové, potenciálne stimulujúce pohľady informatikov (majúcich tendenciu interpretovať dynamické scenáre ako výpočtové procesy) na niektoré javy skúmané molekulárnou biológiou.

11.7.1 Editovanie RNA

Parazitný prvok *Trypanosoma brucei* spôsobujúci africkú spavú nemoc je bunka dlhá asi 4 μm . Jej vonkajšia stena je pokrytá glykoproteínmi, ktoré náhodne menia svoju štruktúru raz za niekoľko dní. To dáva bunke *T. brucei* určitý náskok pred imunitným systémom hostiteľskej bunky. Ako každá bunka, má aj *T. brucei* v jadre chromozómovú DNA. Navyše však obsahuje aj mitochondrie s extrachromozomálnou DNA. V mitochondrii sa nachádzajú "veľké" (asi 30 000 báзовých párov) a "malé" (400-2500 báзовých párov) cyklické molekuly DNA. Malé kruhy sú vzájomne previazané do reťazce. DNA vo veľkých kruhoch kóduje najmenej 13 génov, ale niektoré gény zjavne chýbajú, a DNA v malých kruhoch priamo nekóduje žiadny gén. Záhadnú funkciu takýchto extrachromozomálnych útvarov objasnil R. Benne v roku 1986 [9].

Mnoho génov vo veľkých kruhoch DNA je zakódovaných v špeciálnej forme. Pri transkripcii génov sa zdanlivo nekódujúce úseky DNA obvyklým spôsobom skopírujú na molekuly messenger-RNA (mRNA). Potom nasleduje proces editovania mRNA. Špeciálne "editovacie" proteínové komplexy nachádzajúce sa v *T. brucei* s pomocou špeciálnych "editovacích" molekúl RNA (eRNA) realizujú proces, v ktorom sa niektoré bázy z reťazca

mRNA vyhodí a naopak, určité bázy sa vložia na špecifické miesta v mRNA. Takto "upravené" molekuly mRNA sa potom použijú ako "program" pri konštrukcii bielkovín. Molekuly eRNA obsahujú postupnosť báz špecifikujúcu kontext v mRNA, v ktorom by malo dôjsť k editovaniu, ako aj postupnosť báz definujúcu rozsah vloženia (prípadne zmazania) báz na príslušnom editovacom mieste. Obsahujú tiež časti, ktoré majú tvar chvosta škorpióna a sú naplnené bázami, ktoré sa budú na editovacie miesto vkladať. Ukázalo sa, že eRNA sú kódované v malých kruhových DNA a v postupnostiach báz medzi gény kódujúcimi časťami veľkých kruhových DNA.

Z informatického hľadiska sme práve opísali proces, v ktorom dochádza k editovacím operáciám na molekulách mRNA, pričom editovacie operácie implementované molekulami eRNA sú kontextovo špecifické. Teoretický informatik v takejto situácii vidí možnosť abstraktných výpočtov porovnateľných s výpočtami univerzálneho Turingovho stroja. Skutočne, procesy v bunke *T. brucei* sa nápadne podobajú na molekulárnu implementáciu Turingovho stroja v dôkaze vety 3 z predošlej kapitoly.

11.7.2 Intróny a transpozóny

Transpozóny sú podpostupnosti DNA, ktoré sa môžu premiestňovať z jednej pozície na druhú. Replikujúce sa transpozóny (skrátene im budeme hovoriť replikóny) dokážu viac: sú schopné samostatne sa replikovať, pričom ich kópie sa môžu implantovať na iných miestach v genóme bunky. *Intróny* sú sekvencie DNA v oblastiach kódujúcich gény na DNA, ktoré hoci sú prepísané na mRNA, sa nezúčastňujú na tvorbe bielkovín, pretože sú vylúčené z mRNA ešte pred samotnou syntézou bielkoviny. Sekvenciám, ktoré na mRNA ostávajú a teda priamo kódujú postupnosť aminokyselín v syntetizovanom proteíne, sa hovorí *exóny*. Odhaduje sa, že asi 97% ľudského genómu nemá zatiaľ žiadnu nám známu funkciu. Niekedy sa táto časť označuje aj ako nepotrebná, odpadová DNA (junk DNA).

Zdá sa, že replikóny ovládajú genómy vyšších živočíchov. Takmer 10% ľudského genómu je obsadených dvoma dominantnými replikónmi *Alu* a *LINE1*. Milión kópií replikónu *Alu* v ľudskom genóme plní jedinú úlohu: usmerňovanie syntézy proteínov potrebných na jeho vlastné kopírovanie. Pritom napríklad genóm baktérie *E. Coli* neobsahuje skoro žiadny nepotrebný "odpad". DNA *E. Coli* je podrobne preštudovaná a obsahuje len málo intrónov a jediné replikóny, ktoré môžu byť v *E. Coli* prítomné, sú fágy. Fágy sú pre baktériu *E. Coli* smrteľné a v zdravých bunkách sa nevyskytujú.

Namiesto je otázka: Prečo je z hľadiska evolúcie výhodné mať v genóme "nepotrebné" časti a transpozóny? Transpozóny môžu urýchľovať mutačný proces. Gény nachádzajúce sa v transpozónoch sa spolu s transpozónmi môžu pohybovať po genóme a kopírovať. Najmä kombinácia transpozónov a intrónov je dôležitá pre generovanie mutácií. Prítomnosť len bodových mutácií by mnohokrát nestačila na rýchle generovanie variácií génov v obrovskom priestore možných konfigurácií. Podobne ako v procese kríženia pri sexuálnej reprodukcii môže prehadzovanie a kopírovanie blokov DNA kódujúcich užitočné gény viesť priamočiarejšie a rýchlejšie k vyvinutiu nového užitočného génu ako v prípade náhodných bodových mutácií. To ale neznamená, že transpozóny a intróny sú nevinné vhodné nástroje evolúcie. Tento mechanizmus môže byť aj nebezpečný. Napríklad rakovinové bunky cicavcov obsahujú znásobené gény a ľudské leukocyty sú náchylnejšie na rakovinu ako iné ľudské bunky. Je zaujímavé, že práve leukocyty sa vyznačujú preusporiadaním svojho genetického materiálu.

Replikóny môžu predstavovať hrozbu pre vyššie živočíchy. Pozorovalo sa, že najúspešnejšie replikóny (ako napríklad *LINE1*) majú utlmujúce mechanizmy, ktoré im bránia replikovať sa príliš rýchlo. Transpozón, ktorý by sa kopíroval príliš rýchlo, by totiž ovládol celý genóm a biologický druh by vyhynul.

Matematici W. Smith a A. Schweitzer, ktorých už poznáme ako autorov dôkazu tvrdenia o univerzálnej výpočtovej schopnosti počítačov na báze DNA, prichádzajú s nasledovným zaujímavým (aj keď dosť špekulatívnym) vysvetlením problému intrónov a replikónov.

Vyššie životné formy museli mať nejakú formu imunitného systému, aby sa ubránili jednoduchším parazitom. Ak sa na problém pozeráme z hľadiska informatiky, imunitný systém predstavoval prehadzovanie génov a iné výpočtové schopnosti, ktoré boli vyspelejšie, ako výpočtové procesy v parazitoch. V takomto prípade sa nedalo vyhnúť vyvinutiu výpočtovej Turingovej univerzality v bunkách. Avšak Turingova univerzálnosť urobila život omnoho nebezpečnejší.

Podobne ako je možné malou modifikáciou počítačového programu získať program, ktorý vykonáva omnoho zložitejší výpočet a je ťažšie predikovateľný ako pôvodný program, možno drobnou zmenou štruktúry replikónov vyvolať ťažko predikovateľné následky. W. Smith a A. Schweitzer sa nazdávajú, že boj medzi host'ujúcimi a parazitnými organizmami sa dá interpretovať ako boj výpočtových procesov o nadvládu. Je zaujímavé, že mechanizmy presúvania génov (pomocou replikónov) používané stavovcami na boj proti infekciám sa pozorovali v eukaryotických parazitoch (bunkách s jadrami) pri syntéze obalových glykoproteínov, ktoré sa neustále menia. Turingova univerzalita znamená, že vo všeobecnosti nemožno rozhodnúť, či nejaká časť genómu je nebezpečná alebo nie.

Replikóny ponechané samy na seba majú tendenciu zničiť vyššie životné formy. Keďže vyššie živočíchy existujú, mohlo by to znamenať, že existuje nejaká forma "replikónovej polície". Ak by sa nejaký replikón začal príliš presadzovať zabieraním neúmerne veľkej časti genómu, mohol by sa vyvinúť "smrtiaci gén", ktorý by kódoval enzýmy schopné rozpoznať replikón, vystrihnúť ho z genómu a následne ho zničiť. Nikto však nevie povedať, čo by sa dialo, keby sa takýmto spôsobom podarilo daný replikón zredukovať na únosnú mieru. Smrtiaci gén schopný ničiť časti DNA by totiž ostal naďalej v genóme, a tak by predstavoval pre bunky potenciálnu hrozbu. Situácia je podobná závodom v zbrojení v ľudskej spoločnosti.

V súčasnosti nie je známy žiadny smrtiaci gén pre replikóny. Akýsi nepriamy ukazovateľ možnej existencie replikónovej polície je fakt, že žaby a hmyz nemajú v genóme replikóny *Alu* (v genóme sa však nachádza gén *7SL* veľmi podobný replikónu *Alu*), ktorých varianty sú zahrnuté v genóme cicavcov. Z evolučného hľadiska predstavuje doba existencie replikónov v genóme určitých živočíchov pomerne krátky úsek a replikóny sa zdajú byť veľmi dravé pri ich obsadzovaní. Pri veľkom raste koncentrácie replikónov by však vyššie formy života prestali existovať, a teda zrejme musí existovať mechanizmus regulácie ich nárastu. Aj pozorovanie laboratórnych kmeňov vínnych mušiek *Drosophila* ukázalo, že P-replikón sa v ich genóme objavil len pred 50 rokmi a odvtedy sa jeho koncentrácia mnohonásobne zvýšila.

Informatický boj v genóme živočíchov však pokračuje. Replikóny sú schopné maskovať svoju formu, takže pripomínajú dôležité gény. Ako sme už spomínali, replikón *Alu* veľmi pripomína gén *7SL*. Tento fakt sa zvyčajne interpretuje ako podpora hypotézy o vzniku *Alu* z *7SL*. Môže však ísť aj o spôsob ochrany replikónu *Alu* pred potenciálnymi smrtiacimi génmi a možno práve vďaka tomu je *Alu* ešte stále prítomný v našom genóme.

W. Smith a A. Schweitzer išli vo svojich úvahách ešte ďalej a vyslovili hypotézu, že "odpadová" DNA je z veľkej časti tvorená zbytkami bývalých replikónov. To by vysvetlilo, prečo je značná časť odpadovej DNA repetitívna. Napríklad 25% genómu mušky *Drosophila* tvoria repetície molekuly $\uparrow$ ATAAACT. Najprv sa myslelo, že tieto repetitívne sekvencie môžu plniť nejakú štruktúrálnu funkciu. Ich úloha však nemôže byť dôležitá, lebo aj veľmi príbuzné druhy majú v odpadovej DNA úplne odlišné repetitívne sekvencie.

11.8 Nová vedná disciplína

Cieľom tejto kapitoly bolo načrtnúť základné idey a pracovné postupy prudko sa rozvíjajúcej vednej disciplíny - výpočtov na báze DNA. Z informatického hľadiska môžu počítače na báze DNA podstatne posunúť hranice praktickej riešiteľnosti mnohých kombinatorických problémov, v ktorých priestor potenciálnych riešení rastie exponenciálne s rozsahom problému. Na druhej strane môže spôsob nazerania informatikov na problémy skúmané molekulárnou genetikou priniesť nové zaujímavé hypotézy a riešenia.

Práce venované výpočtom na báze DNA sledujú dva hlavné trendy:

1. skúmanie vhodných techník kódovania dát sekvenciami DNA, spracovania dát manipuláciou s DNA a interpretácie výsledkov výpočtov
2. určenie výpočtovej sily rôznych modifikácií výpočtov na báze DNA.

Zo zaujímavých aplikácií výpočtov na báze DNA spomenieme pokus D. Boneha, Ch. Dunwortha a R. Liptona navrhnuť systém na dešifrovanie štandardu šifrovania dát (Data Encryption Standard - DES) [10]. Tento problém študoval aj "otec" počítačov na báze DNA L. Adleman [11]. Výpočty na báze DNA sa realizujú zväčša *in vitro*, teda v skúmavkách mimo živých organizmov. Alternatívne je možné skonštruovať počítače na báze DNA *in vivo*. Ako vhodné médium boli v [12] použité baktérie. V baktériách *E. coli* možno znásobiť želané úseky DNA technikou molekulárneho klonovania využívajúcou plazmidy ako nosiče cudzorodej DNA (viď kapitolu 11.1.).

Zaujímavý prístup k určovaniu výpočtovej sily rôznych variant počítačov na báze DNA sa objavil v [13]. R. Freund, L. Kari a G. Paun formalizujú tento počítač ako formálny systém s konečným počtom prepisovacích pravidiel špeciálneho typu. Ich dôkaz turingovskej ekvivalencie výpočtov na báze DNA spočíva v ukázaní, že každá gramatika typu 0 v Chomského hierarchii (na rozpoznanie jazykov generovaných týmto typom gramatík je potrebné použiť Turingov stroj) sa dá previesť na formálny prepisovací systém zodpovedajúci určitému výpočtovému systému na báze DNA. Alternatívna implementácia univerzálneho nedeterministického Turingovho stroja založená na interakcii malých molekúl DNA bola publikovaná v [14].

Rozsah tejto kapitoly neumožnil spomenúť mnoho mien a prác, ktoré tvoria dôležité míľniky na ceste skúmania výpočtov na báze DNA. Čitateľa, ktorého táto problematika hlbšie zaujala, odkazujeme na publikácie [15]-[22].

Na záver treba poznamenať, že riešenie realistických úloh veľkého rozsahu výpočtami na báze DNA naďalej zostáva hudbou budúcnosti. Je však možné, že technika výpočtov na báze DNA nájde uplatnenie v medicíne pri konštrukcii mikroskopických organických výpočtových prostriedkov, ktoré bude možné implantovať do žijúceho organizmu [22].

Literatúra

- [1] F. Hoppensteadt: Getting started in mathematical biology. *Notices of the AMS* **42**(9) (1995) 969-975. Dostupný na internetovskej adrese <http://www.ams.org/notices/199509/199509-toc.html>
- [2] L. Adleman: Molecular computation of solutions to combinatorial problems. *Science* **266** (1994) 1021-1024.
- [3] L. Adleman: On constructing a molecular computer. In R. Lipton and E. Baum (eds.): *DNA based computers*. American Mathematical Society, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Vol. 27, 1996, pp. 1-21.

- [4] R. Deaton, R.C. Murphy, M. Garzon, D.R. Franceschetti, and S.E. Stevens Jr.: Good encodings for DNA-based solutions to combinatorial problems. In R. Lipton and L. Landweber (eds.): *DIMACS proceedings of the second annual Princeton conference on DNA based computers*. American Mathematical Society, 1996, pp. 159-171.
- [5] W.D. Smith and A. Schweitzer: DNA computers in vitro and vivo. In R. Lipton and E. Baum (eds.): *DNA based computers*. American Mathematical Society, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Vol. 27, 1996, pp. 121-186.
- [6] R.J. Lipton: DNA Solution of Hard Combinatorial Problems. *Science* **268** (1995) 542-545.
- [7] R.J. Lipton: Speeding up Computations via Molecular Biology (nepublikovaný rukopis (1995)). Dostupný na internetovskej adrese <ftp://ftp.cs.princeton.edu/pub/people/rjl/bio.ps>
- [8] M. Kálmán et al.: Synthesis of a gene for human serum albumin and its expression in *Saccharomyces Cerevisiae*. *Nucleic Acid Research* **18** (1990) 6075-6081.
- [9] R. Benne et al.: Major transcript of the frameshifted *coxII* gene from trypanosome mitochondria contains four nucleotides that are not encoded in the DNA. *Cell* **46** (1986) 819-826.
- [10] D. Boneh, Ch. Dunworth, and R. J. Lipton: Breaking DES Using a Molecular Computer. In R. Lipton and E. Baum (eds.): *DNA based computers*. American Mathematical Society, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Vol. 27, 1996, pp. 37-66. Dostupný na internetovskej adrese <ftp://ftp.cs.princeton.edu/techreports/1995/489.ps.Z>
- [11] L.M. Adleman, P.W.K. Rothmund, S. Roweis and E. Winfree: On Applying Molecular Computation to the Data Encryption Standard. *Journal of Computational Biology* **6**(1) (1999) 53-64.
- [12] P.N. Bertone: *Combinatorial Solution of Search Problems via Biomolecular Computation in vivo* (manuscript) (1996). Dostupný na internetovskej adrese http://www.umassd.edu/Public/People/PBertone/research/Biocomp_Rev_2-27.pdf.
- [13] R. Freund, L. Kari, and G. Paun: DNA computing based on splicing: the existence of universal computers. *Theory of Computing Systems* **32** (1999) 69-112.
- [14] D. Beaver: A Universal Molecular Computer. In E.B. Baum and R.J. Lipton (eds.): *DNA Based Computers*. American Mathematical Society, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Vol. 27, 1996, pp. 29-36.
- [15] L. Adleman: Computing with DNA. *Scientific American* **279** (1998) 54-61.
- [16] F. Guarnieri, M. Fliss, and C. Bancroft: Making DNA. *Science* **273** (1996) 220-223.
- [17] Q. Ouyang, P. Kaplan, S. Liu and A. Libchaber: DNA solution of the maximal clique problem. *Science* **278** (1997) 446-449.
- [18] A.G. Frutos, L.M. Smith, and R.M. Corn: Enzymatic ligation reactions of DNA "words" on surfaces for DNA computing. *Journal of American Chem. Society* **120** (1998) 10277-10282.
- [19] L.F. Landweber and E.B. Baum (eds.): *DNA Based Computers*. American Mathematical Society, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 44, 1999.
- [20] R.M. Kapr, C. Kenyon, and O. Waarts: Error-resilient DNA computation. *Random Struct. Algor.* **15** (1999) 450-466.
- [21] Q. Liu, L. Wang, A.G. Frutos, A.E. Condon, R.M. Corn, and L.M. Smith: DNA computing on surfaces. *Nature* **403** (2000) 175-179.
- [22] M. Ogihara and A. Ray: DNA computing on a chip. *Nature* **403** (2000) 143-144.

Index

- acyklický orientovaný graf 158
- Adami 143
- adenín 183
- Adleman 183, 191
- afinitné prečisťovanie (affinity purification) 193
- agenti, autonómni 123-124
- aktivácia 159, 172
- aktivačná (prechodová) funkcia 159-161
- aktivita 159-162
- algoritmická, umelá chémia 146-147
- algoritmizácia neurónovej siete 161-162, 172, 176
- algoritmus 189
- algoritmus analýzy Readovho kódu 61
- algoritmus backtracking 8, 61, 62
- algoritmus evolúcie 34
- algoritmus evolučného programovania 30
- algoritmus evolučnej stratégie 103
- algoritmus evolučnej stratégie, mnohočlennej 114
- algoritmus generácie Readovho kódu 63
- algoritmus generátora náhodnej premennej 106
- algoritmus generovania súťažiacich populácií 132
- algoritmus generovania veľkosti populácie 130
- algoritmus genetický 31, 44, 55, 168
- algoritmus genetický paralelný 55-56
- algoritmus genetický pre m-chromozómy 54
- algoritmus Hopfieldov-Tankov 172
- algoritmus horolezecký (hill climbing) 19, 102
- algoritmus horolezecký s učením (hill climbing with learning) 20-23
- algoritmus Kohonenov 176
- algoritmus konštrukcie prvočísel čísel-
deliacou chémiou 147
- algoritmus kríženia (crossover) 38
- algoritmus kríženia m-chromozómu (m-
crossover) 54
- algoritmus kríženia Readovho kódu 65
- algoritmus Metropolisov 79
- algoritmus Metropolisov s elitizmom 87
- algoritmus mutácie binárneho reťazca 18
- algoritmus mutácie koreňového stromu 65, 70
- algoritmus mutácie m-chromozómu 53
- algoritmus mutácie Readovho kódu 65
- algoritmus mutácie reálneho vektora
Perturbation 103
- algoritmus mutácie zobrazenia 96
- algoritmus neurónovej siete 162, 172, 176
- algoritmus operátora inverzie 49
- algoritmus paralelného simulovaného
žihania (Parallel Simulated Annealing)
90
- algoritmus pre distribúciu pravdepodob-
nosti binárnych vektorov získanú
mutáciou 17
- algoritmus reprodukcie 40
- algoritmus rulety 37
- algoritmus simulovaného žihania 80
- algoritmus simulovaného žihania s
elitizmom 87
- algoritmus slepý (blind) 14
- algoritmus trapézový 105
- algoritmus zakázaného hľadania (tabu
search) 27
- algoritmy stochastické 100
- aminokyseliny 184
- analýza (parsing) kódu 61
- analýza regresná 155, 157
- analýza zhluková (clustering) 155, 177
- animácia 126, 134
- aproximátor univerzálny 155
- aspiračné kritérium 26
- autokatalytická sieť 145-146
- automat 117, 189
- automat celulárny (bunkový) 124, 134-139
- Automaticky Definované Funkcie (ADF) 66, 165
- autonómni agenti 123-124
- autoreplikácia, sebareplikácia 122, 143
- Avida 143
- avidín 193, 197
- Axelrod 141
- backpropagation - metóda spätného šírenia
161, 164
- backtracking algoritmus 8, 61, 62
- baktéria 184-185, 187-188, 200-201, 203

bakteriofág 184
 Baldwinov efekt 144
 bariéra Hammingova 48-51, 104
 bázy komplementárne 184-185, 190
 Bellew 165
 Benne 200
 binárna reprezentácia reálnej premennej 9, 16
 biomolekuly 122, 143
 biomorph 128
 biorytmy 128
 biotín 193
 biotín-avidínová technika 197
 boid 134
 Boltzmannov stroj 169-170, 173
 Boltzmannovské rozdelenie pravdepodobnosti 77, 80, 87
 Boneh 190, 203
 Boolovské funkcie 72-74, 194
 Boxova-Mullerova metóda 105
 Brook 139
 bublinkové triedenie (bubble sort) 142
 building blocks (stavebné kamene) 47
 bunka 184, 185, 188, 200
 bunkový (celulárny) automat 124, 134-139
 Buss 149

 Cairns 148
 Cauchyho distribúcia 170
 Cauchyho stroj 170
 celulárne kódovanie 165-166
 celulárny (bunkový) automat 124, 134-139
 cesta hamiltonovská 93, 191-194
 clustering (zhluková analýza) 155, 177
 Conway 124, 135
 CPU (Central Processor Unit) čas 8
 Crick 183
 crossover (kríženie) 33, 38, 54, 65, 108-110
 Csontó 121
 cytoplazma 184
 cytozín 183

 Černý 77
 čiarka stratégia 108
 čiastočné priradenie (partial matching) 94

 dámy, problém rozmiestnenia na šachovnici 98-99
 Darwinova evolučná teória 2, 31
 Data Encryption Standard - DES 203
 Dawkins 128, 149

 Deaton 193
 DeBruijnove sekvencie 194
 deceptívne účelové funkcie 51, 83
 dekompresia Readovho kódu 66-67
 deoxyribonukleová kyselina (DNA, Deoxyribonucleic Acid) 128, 144, 147, 183-204
 deoxyribóza 183
 deskriptor 155, 160-161
 diferenciálne rovnice 129-130
 disperzia 81
 distribúcia Cauchyho 170
 distribúcia gaussovská 16, 100-101, 104, 168, 170
 distribúcia logistická 105
 distribúcia Boltzmannovská 77, 80, 87
 distribúciu pravdepodobnosti, algoritmus pre binárnych vektorov 17
 Dittrich 146
 dlhodobá pamäť 27
 DNA 128, 144, 147, 183-204
 DNA odpadová (junk DNA) 201-202
 DNA výpočty 183-204
 DNA výpočty in vivo 200
 doména 80
 dopredné šírenie (feed-forward) 155, 161-162
 dravci 128-129, 142
 drift genetický 2-3
 Drosophila, muška 202
 Dunworth 190, 203
 dvojbodové kríženie 39
 dvojité závitnice 184-186, 188, 189
 dvojzávit DNA 184-186, 188-190, 193, 197-200
 dynamika evolučná 128-134

 efekt Baldwinov 144
 Eigen 144
 elektroforéza 193
 emergent behavior 121
 endonukleázy restriktčné 185
 energia 80
 entropia 81-82
 epigenéza 147
 ES 100
 Escherichia coli 148, 184, 185, 187, 198, 201, 203
 euklidovský priestor 173
 evolúcia, algoritmus 34
 evolúcia, cesty 31, 144

evolúcia lamarkovská 144
 evolúcia, trendy 149
 evolučná dynamika 128-134
 evolučná stratégia (evolution strategy),
 algoritmus 103
 evolučná stratégia mnohočlenná,
 algoritmus 114
 evolučné programovanie (evolutionary
 programming) 28-30, 117, 168
 evolware 139
 exchange (výmena) 56
 exón 201
 extrachromozomálny genetický systém 185
 extrakcia pravidiel z neurónovej siete 156
 extrém globálny 14, 18, 48, 107, 109

fág 184-188, 198, 201
 faktor prahový 159
 fázový prechod 85, 137
 feed-forward neural networks (neurónové
 siete s dopredným šírením) 156-168
 FEM 118
 fenotyp 3, 35, 109, 121, 164
 fitness 3, 31
 fitness renormalizovaná 35-36
 Fogel 117
 Formula Satisfaction Problem (splniteľnosť
 logickej formuly) 194-196
 fraktál 6, 127-128, 139
 Freund 203
 funkcia aktivačná (prechodová) 159-161
 funkcia hustoty rozdelenia náhodnej
 premennej 100, 105-107, 110
 funkcia partičná 77
 funkcia Rosenbrockova 114
 funkcia sigmoidálna 155, 159-161, 171
 funkcia účelová 157
 Funkcie Automaticky Definované (ADF) 66,
 165
 funkcie Boolovské 72-74, 194
 funkcie účelové deceptívne 51, 83
 funkcionál 70
 fylogénéza 147

Gardner 135
 gaussovská distribúcia 16, 100-101, 104,
 168, 170
 gaussovske mutácie 100-101, 168
 Gelatt 77
 generátor náhodnej premennej 106

genetická informácia, horizontálny prenos
 148, 185
 genetické programovanie (genetic
 programming) 58-76, 165
 genetický algoritmus paralelný 55-56
 genetický algoritmus 31, 44, 55, 168
 genetický algoritmus pre m-chromozómy 54
 genetický algoritmus, teória 45
 genetický drift 2-3
 genotyp 3, 32, 35, 109, 121, 164
 glider (klzák) 135
 globálne minimum, veta 14, 18, 48
 globálny extrém 48, 107, 109
 Glover 24
 glykoproteín 200, 202
 Gold 148
 Gosper 136
 gradientová metóda 102, 161
 graf acyklický orientovaný 158
 grafová postupnosť 62
 gramatické kódovanie 163-166
 graph bipartitioning (*rozklad vrcholovej
 množiny grafu na dve podmnožiny*) 156,
 174
 Grayov kód 9-11, 16, 17
 Grunau 165-166
 grupa symetrická 92
 guanín 183

hamiltonovská cesta 93, 191-194
 Hammingova bariéra (útes) 48-51, 104
 Hammingova vzdialenosť 7, 48, 164
 Hebbovo pravidlo 21, 176
 Herdy 100
 hill climbing (horolezecký algoritmus) 19,
 102
 hill climbing with learning (horolezecký
 algoritmus s učením) 20-23
 Hillis 141
 Hinton 170
 Hofstadter 147
 Holland 35, 47, 135, 141
 Hopfieldova neurónová sieť 156, 168-175
 Hopfieldov-Tankov algoritmus 172
 Hopfieldova-Tankova sieť 171
 horizontálny prenos genetickej informácie
 148, 185
 horolezecký algoritmus (hill climbing) 19,
 102
 horolezecký algoritmus s učením (hill
 climbing with learning) 20-23

hra Life (život) 124, 135-136
 hrana 158
 hustota rozdelenia náhodnej premennej,
 funkcia 100, 105-107, 110
 hypercykly 144
 hyperkocka 6-8, 12

 Chalmers 166
 chaos 137-138
 chárón 187
 chémia algoritmická, umelá 146-147
 chémia kombinatorická 143
 Chomského hierarchia 203
 chromozóm "messy" 51-55
 chromozóm 3, 32, 35-57, 108-109, 111,
 113-115, 128, 141, 148, 164-165, 184-
 185, 187, 200

 in vitro 203
 in vivo 187, 200, 203
 iniciátor (primer) 188, 189, 192
 intrón 201
 inverzný operátor 49
 Isingov model spinového skla 169
 iterácie 160-161

 jednobodové kríženie 38-42
 junk DNA (odpadová DNA) 201-202

 Kálmán 200
 Kari 203
 karteziánsky produkt 6, 92
 Kaufmann 144, 146, 148
 Kelemen 121
 Kirkpatrick 77
 Kitano 165
 klasifikačný problém 156
 klasifikátor 155-156
 klastrovanie (zhluková analýza) 155, 177
 klonovanie molekulové 186, 188, 203
 klzák (glider) 135
 KNN klasifikátor 155
 kód Grayov 9-11, 16, 17
 kodón 184
 kódovanie celulárne 165-166
 kódovanie gramatické 163-166
 koeficient učenia 21 (112, 166, 172)
 koeficient váhový 159
 koevolúcia 140-143
 Kohonenov algoritmus 176

 Kohonenova neurónová sieť 156, 169, 175-
 178
 Kochov ostrov 127
 kolónie mravcov 133
 kombinatorická chémia 143
 kombinatorické optimalizačné problémy 92-
 99, 118, 156, 169-178, 203
 komplementárne bázy 184-185, 190
 komplexné javy 137-140
 kompresia Readovho kódu 66-67
 konvergenčný teorém 48, 107
 kooperácia 140-143
 koreň stromu 58-59
 koreňový strom, mutácia 65, 70
 kovariancia 107-114
 kovariančná matica 110
 Koza 58, 64, 66, 118, 165
 kozmid 187-188
 krátkodobá pamäť 24
 krídel 133
 kritérium aspiračné 26
 kríženie (crossover) 33, 38, 54, 65, 108-110
 kríženie (crossover) algoritmus 38
 kríženie dvojbodové 39
 kríženie jednobodové 38-42
 kríženie lineárnou kombináciou 110
 kríženie m-chromozómu (m-crossover)
 algoritmus 54
 kríženie permutácie 94
 kríženie priemerom 108-109
 kríženie Readovho kódu, algoritmus 65
 kríženie uniformné 109-110
 Kroneckerovo delta 82, 98
 kvázináhodný výber 33-38

 Laguna 24
 lamarkovská evolúcia 144
 lambda 137-138
 Langton 121, 135, 137
 Layzer 142
 lepkavé konce (sticky ends) 185-186, 197-
 199
 Levenbergova-Marquardtova metóda 161
 Levy 122
 liečivá 143
 Life (život), hra 124, 135-136
 ligáza 186, 187, 192, 198-200
 Lin 173
 Lindenmayerove systémy 126-127
 Lindgren 142
 lineárna kombinácia, kríženie 110

linker 186
 Lipton 190, 194, 196, 203
 LISP 59
 literál 195
 logistická distribúcia 105
 logistická Verhulstova-Pearlova rovnica 130
 Lotkova-Volterrova rovnica 128
 L-systémy 126-127

 magnetizácia 175
 Manhattan vzdialenosť 97-99, 177
 markovovské procesy 117
 matica kovariančná 110
 memetika 144
 messenger RNA 184, 200
 messy chromozóm 51-55
 meta-evolúcia 108
 metóda Boxova-Mullerova 105
 metóda gradientová 102, 161
 metóda konečných prvkov (FEM) 118
 metóda Levenbergova-Marquardtova 161
 metóda Monte Carlo 78
 metóda spätného šírenia - backpropagation 161, 164
 Metropolis 78
 Metropolisov algoritmus 79
 Metropolisov algoritmus s elitizmom 87
 Miller 144
 minimalizácia, účelová funkcia 157
 mitochondrie 147, 200
 mnohočlenná evolučná stratégia 107
 mnohočlenná evolučná stratégia, algoritmus 114
 množina testovacia 155-156
 množina tréningová 155-156
 molekulárny počítač 193
 Monte Carlo metóda 78
 morfogénéza 128
 Morris 128
 mravce 133
 multimnožina (multiset) 194
 muška Drosophila 202
 mutácia 15, 33, 80, 83, 100
 mutácia binárneho reťazca, algoritmus 18
 mutácia koreňového stromu 65, 70
 mutácia m-chromozómu, algoritmus 53
 mutácia permutácie, algoritmus 93
 mutácia reálneho vektora, algoritmus Perturbation 103
 mutácia zobrazenia, algoritmus 96
 mutácie gaussovské 100-101, 168

 mutácie smerované 148

 náhodná premenná, generátor 106
 Navierove a Stokesove rovnice 139
 navigácia robotov 58, 139-140, 146, 165, 167, 178
 neurón 155
 neurón, skrytý (hidden) 158
 neurón, vstupný (input) 158
 neurón, výstupný (output) 158
 neurónová sieť Hopfieldova 156, 168-175
 neurónová sieť Kohonenova 156, 169, 175-178
 neurónové siete 118, 125-126, 147, 155-182
 neurónové siete rekurentné 160
 neurónové siete s dopredným šírením (feed-forward neural networks) 156-168
 nika 144, 149
 normalizovaná fitness 35-36
 NP (Nondeterministic Polynomial) obtiažne (úplné) problémy 92, 118, 189-190
 NPP, problém rozkladu čísla (Number Partitioning Problem) 95
 nukleotid 183

 obchodný cestujúci, problém 92, 118, 170-172, 175-178
 obraz, spracovanie (image processing) 169, 179
 odchýlka štandardná 100, 105-106
 odpadová DNA (junk DNA) 201-202
 oligonukleotidy 185, 189, 192, 193
 Olsson 141
 ontogenéza 147
 operátor inverzie 49-50
 optimalizačný problém 6
 ortogonálnosť 113
 oscilácie 128, 136

 palindróm 193-194
 palindromatická nukleotidová sekvencia 185
 pamäť dlhodobá 27
 pamäť krátkodobá 24
 paralelné simulované žihanie (Parallel Simulated Annealing) 88-90
 paralelný genetický algoritmus 55-56
 Parallel Simulated Annealing (paralelné simulované žihanie) 88-90
 parameter usporiadania 22, 44

párenie, problém najcennejšieho (weighted matching problem) 156, 173-174
 Pargellis 143
 parita 72-73
 parse tree (syntaktický strom) 58
 parsing (analýza) kódu 61
 partičná funkcia 77
 Paun 203
 Pawley 172
 Pearl 130
 penalizácia 27, 171-175
 permutácia 92, 169
 permutácia, algoritmus mutácie 93
 permutácia, kríženie 94
 perturbation (porucha) 78, 103
 pixel 159
 plazmid 184, 188-203
 plus stratégia 102, 108
 počítač molekulárny 193
 počítač Turingov 136, 189-190, 197-203
 počítač virtuálny 143
 počítačový vírus 124, 143
 pohlavná reprodukcia 40
 polymeráza 183-184, 188-189, 198-199
 polymerázna reťazová reakcia (Polymerase Chain Reaction - PCR) 188-189
 polymerizácia 183
 porucha (perturbation) 78, 103
 postupnosť grafová 62
 potenciál neurónu 159
 povrch fitness (fitness' surface/landscape) 31-32
 prahový faktor 159
 pravdepodobnosť, Boltzmannovské rozdelenie 77, 80, 87
 pravdepodobnostný vektor 20-23, 43-44
 pravidlo 1/5 102, 107
 prebiotická polievka 144-146
 prediktor 155-156
 prechodová (aktivačná) funkcia 159-161
 prerušovaná rovnováha (punctuated equilibrium) 148
 preučenie 161
 priemerom kríženie 108-109
 priestor euklidovský 173
 primer (iniciátor) 188, 189, 192
 prirodzený výber 2
 prisoner's dilemma (problém väzňov) 140-141
 problém klasifikačný 156
 problém obchodného cestujúceho (Traveling Salesperson Problem) 92, 118, 170-172, 175-178
 problém rozkladu čísla (NPP, Number Partitioning Problem) 95
 problém najcennejšieho párenia (weighted matching problem) 156, 173-174
 problém rozmiestenia dám na šachovnici 98-99
 problém väzňov (prisoner's dilemma) 140-141
 problémy kombinatorické 92-99, 118, 166-178, 203
 problémy NP (nonpolynomialy) obtiažne, úplné 92, 118, 189-190
 procesy Markovovské 117
 produkt karteziánsky 6, 92
 programovacie jazyky 59, 105
 programovanie evolučné (evolutionary programming) 28-30, 117, 168
 programovanie genetické (genetic programming) 58-76, 165
 proteíny 128, 184, 200-202
 prvok Trypanosoma brucei 200
 punctuated equilibrium (prerušovaná rovnováha) 148
 Purkyneho vlákna 128
 Ray 143
 Readov kód, algoritmus analýzy 61
 Readov kód, algoritmus generácie 63
 Readov kód, algoritmus kríženia 65
 Readov kód, algoritmus mutácie 65
 Readov kód, dekompresia 66-67
 Readov kód, kompresia 66-67
 Readov lineárny kód 59-71, 76
 reálna reprezentácia premenných 100
 reálny vektor, algoritmus mutácie Perturbation 103
 redukcia vstupných dát 155
 Reed 130
 regresia symbolická 58, 67-76
 regresná analýza 155, 157
 Rechenberg 100
 rekurentné neurónové siete 160
 relaxačný term 171-172
 renormalizovaná fitness 35-36
 replikón 185-186, 201-202
 reprezentácia premenných reálna 100
 reprezentácia, binárna reálnej premennej 9, 16

reprodukcia, algoritmus 40
 reprodukcia, pohlavná 40
 reprodukčný proces 2, 40
 Resnick 126
 restričné endonukleázy 185
 Reynolds 133
 riadenie 165, 167, 179
 ribonukleová kyselina (RNA) 144, 184, 200-201
 ribóza 184
 ribozóm 184
 RNA, ribonukleová kyselina 144, 184, 200-201
 robot 58, 121, 123, 139-140, 146, 165, 167, 178
 Rosenbrockova funkcia 114
 rovnica logistická Verhulstova-Pearlova 130
 rovnica Lotkova-Volterrova 128
 rovnice diferenciálne 129-130
 rovnice Navierove a Stokesove 139
 rovnováha, prerušovaná (punctuated equilibrium) 148
 rozklad čísla, problém (NPP, Number Partitioning Problem) 95
rozklad vrcholovej množiny grafu na dve podmnožiny (graph bipartitioning) 156, 174
 Rudolph 113
 ruleta, algoritmus 37
 Rumelhart 161
 rýchlosť učenia 21, 112, 166, 172

 saltationism (skokový vývoj) 143, 148
 samo-adaptácia 108
 samo-organizácia 148
 SamoOrganizujúca sa Mapa, Kohonenova (SOM) 156, 169, 175-178
 sebareplikácia, autoreplikácia 122, 143
 Sejnowski 170
 selekcia 35-37, 108-109, 118
 schéma 45-48
 schéma, veta (schema theorem) 47
 Schuster 144
 Schwefel 100, 148
 Schweitzer 197, 200, 202
 sieť autokatalytická 145-146
 sieť Hopfieldova-Tankova 171
 sieť, topológia 163-166, 176
 sieť, váhy 159-162, 167
 siete neurónové 118, 125-126, 147, 155-182

 siete neurónové rekurentné 160
 sigmoidálna funkcia 155, 159-161, 171
 SimLife 126
 Sims 128
 simulované žihanie (Simulated Annealing) 77-91, 168, 173-174
 simulované žihanie paralelné (Parallel Simulated Annealing) 88-90
 simulované žihanie s elitizmom, algoritmus 87
 simulované žihanie, algoritmus 80
 sklo spinové, Isingov model 169
 skrytý (hidden) neurón 158
 sled idúci cez všetky vrcholy stromu (spanning walk) 61
 slepý (blind) algoritmus 14
 smerované mutácie 148
 Smith 197, 200, 202
 spanning walk (sled idúci cez všetky vrcholy stromu) 61
 spätného šírenie - backpropagation 161, 164
 spinové sklo, Isingov model 169
 splniteľnosť logickej formuly (Formula Satisfaction Problem) 194-196
 spracovanie obrazu (image processing) 169, 179
 stavebné kamene (building blocks) 47
 sticky ends (lepkavé konce) 185-186, 197-199
 stochastické algoritmy 100
 stratégia čiarka 108
 stratégia evolučná (evolutionary strategy) 100
 stratégia plus 102, 108
 stratégia mnohočlenná 107
 stredná hodnota 81, 100
 stroj Cauchyho 170
 strom koreňový 58, 59
 strom syntaktický (parse tree) 58
 superpočítač 189, 193
 surface/landscape of fitness' (povrch fitness) 31-32
 swarm (roj) 133, 139
 symbióza 147
 symbolická regresia 58, 67-76
 syntaktický strom (parse tree) 58

 šachovnica, problém rozmiestnenia dám 98-99
 špecifické teplo 81, 85-86, 88

štandardná odchýlka 100, 105-106
 štatistická fyzika 81

tabu list (zakázaný zoznam) 25
 tabu search (zakázané hľadanie) 24-28, 98
 Taq polymeráza 188-189, 198
 templát 52, 54
 teorém konvergenčný 48, 107
 teória genetického algoritmu 45
 tepelná kapacita 83
 teplota 77-90, 170, 174
 term relaxačný 172
 testovacia množina 155-156
 Tierra 143
 tit-for-tat stratégia 140
 topológia siete 163-166, 176
 toroid 55, 124, 136
 transferová RNA 184
 transformácia spojitého optimalizačného problému na binárny 9
 transpozón 185, 201
 trapézový algoritmus 105
 Traveling Salesperson Problem (problém obchodného cestujúceho) 92, 118, 170-172, 175-178
 trendy evolúcie 149
 tréningová množina 155-156
 triedenie bublinkové (bubble sort) 142
 TSP problém 92, 118, 170-172, 175-178
 Turingov počítač 136, 189-190, 197-203
 turnaj 28, 118
 tymín 183
 typogenetika 147

účelová funkcia 157
 účelové funkcie deceptívne 51, 83
 učenie neurónovej siete 155
 učenie, koeficient 21
 učenie s učiteľom 161
 učenie, rýchlosť 21, 112, 166, 172
 Ulam 134
 umelá, algoritmická chémia 146-147
 umelý život (Artificial Life) 121-154
 uniformné kríženie 109-110
 univerzálny aproximátor 155
 uracil 184
 útes Hammingov 48, 104

váha spojenia neurónov 159-162, 167
 váhový koeficient 159
 valencia (stupeň) vrcholu 59

valencia grafu 174
 Vecchi 77
 vektor (v genetike) 186-188
 vektor pravdepodobnostný 20-23, 43-44
 Verhulstova-Pearlova rovnica 130
 veta o globálnom minime 14, 18, 48, (107, 109)
 veta o schéme (schema theorem) 47
 virtuálna realita 121, 125
 virtuálny počítač 143
 virtuálny svet 143
 vírus, počítačový 124, 143
 von Neumann 134-135
 vrcholy binárne 68
 vrcholy funkcionálne 68
 vrcholy koncové, listy (terminal) 68
 vrcholy ternárne 68
 vrcholy unárne 68
 vrstva neurónov 158
 vstupný neurón (input) 158
 výmena (exchange) 56
 výstupný (output) neurón 158
 vzdialenosť Hammingova 7, 48, 164
 vzdialenosť Manhattan 97-99, 177
 vznik života 143-149

Watson 183
 Watson-Crickov komplement 190
 weighted matching problem (problém najcennejšieho párenia) 156, 173-174
 Whitley 165
 Widrow-Hoffovo "delta" pravidlo 166
 Wilson 172
 Wolframove triedy 137
 worm 125

zakázané hľadanie (tabu search) 24-28, 98
 zakázaný zoznam (tabu list) 25
 závitnica dvojité 184-186, 188, 189
 zhluková analýza (clustering) 155, 177
 zobrazenie, algoritmus mutácie 96
 zrkadlenie hranice 103

život (Life), hra 124, 135-136
 život umelý (Artificial Life) 121-154
 život, definícia 122-123
 život, vznik 143-149