9. Evoluční algoritmy a neuronové sítě

9.1 Úvod

Evoluční algoritmy jsou zastřešujícím termínem pro řadu přístupů využívajících modelů evolučních procesů pro účely téměř nikdy nemající nic společné s biologií. Snaží se využít představ o hnacích silách evoluce živé hmoty (případně simulace tvorby krystalů v případě simulovaného žíhání) pro účely optimalizace. Všechny tyto modely pracují s náhodnými změnami navrhovaných řešení. Pokud jsou tato nová řešení výhodnější, nahrazují předcházející řešení.

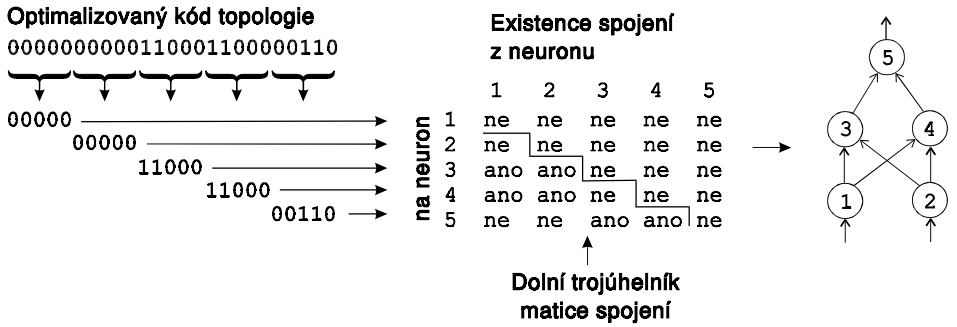
Evoluční algoritmy se používají při stochastické optimalizaci neuronových sítí. Tyto algoritmy využívají informaci z předcházejícího řešení při návrhu nové, lepší sítě. Snažíme se přitom, aby suma čtverců odchylek výstupů z neuronové sítě od předem zadaných hodnot byla co nejmenší. V této kapitole se nadále budeme zabývat typem "*feedforward*", t.j. dopředných vícevrstvých neuronových sítí. Tam, kde se budeme zabývat optimalizací topologie sítě, bude učící strategii tvořit ten nejznámější algoritmus — "*backpropagation*", t.j. metoda adaptace pomocí zpětného šíření chyb. Toto zjednodušení je oprávněné z toho důvodu, že optimalizace pomocí stochastických evolučních algoritmů se v naprosté většině používá právě pro tento základní typ neuronové sítě.

V optimalizaci neuronových sítí se vyskytují dva základní úkoly: optimalizace topologie a optimalizace vah.

■ *Optimalizace topologie neuronové sítě* spočívá v určení počtu skrytých vrstev, počtu neuronů v těchto vrstvách, existence spojení mezi nimi (obr. 9.1), popřípadě i parametrů přechodové funkce neuronu nebo parametrů pro učení sítě pomocí zpětného šíření.

Příklad uvedený na obr. 9.1 ukazuje pouze jednu z možností, co všechno se dá optimalizovat. Daný příklad se dá ještě zjednodušit. Pokud bychom nechtěli pokaždé kontrolovat, jestli je síť opravdu "feed forward", tedy jestli náhodou některá spojení netvoří cyklus, pak je nejjednodušší vyplnit pouze dolní trojúhelník matice spojení.

Tento typ zakódování topologie se nazývá "nízkoúrovňovým", protože přímo kóduje topologii. U jiných typů zakódování, tzv. "vysokoúrovňových", může být architektura sítě specifikována pravidly růstu nebo větami formálního jazyka. Tento typ je více vhodný pro optimalizaci rozsáhlejších sítí (na kterou v našich podmínkách ale stejně nemáme dostatečně rychlý hardware).

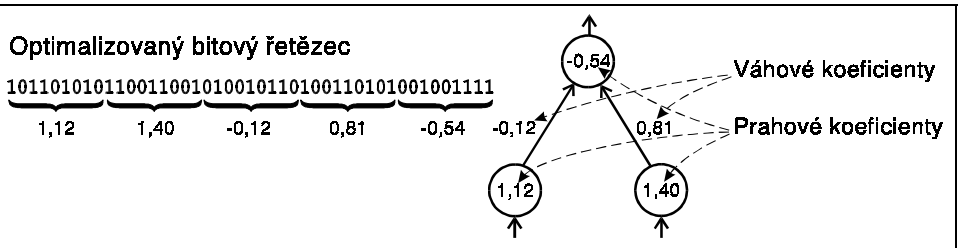


Obrázek 9.1. Příklad zakódování topologie spojení pěti neuronů. Optimalizace topologie neuronové sítě je převedena na optimalizaci úvodního bitového řetězce, přičemž první dva neurony jsou definované jako vstupní a poslední pátý neuron je výstupní.

Stochastické optimalizační algoritmy jsou v podstatě jediným systematickým přístupem k optimalizaci topologie sítě. Bohužel, tyto algoritmy potřebují vygenerovat a ohodnotit řádově tisícovky (nebo více) možných topologií neuronových sítí, aby dospěly k dobrému výsledku. Vzhledem k tomu, že ohodnocení každé topologie představuje vlastně naučení jedné neuronové sítě pomocí tisíců iterací zpětného šíření, jde o výpočetně velmi náročný úkol. Proto se místo systematického přístupu k návrhu topologie stále běžně používá spíše intuice, a příklady optimalizace topologie neuronové sítě existují většinou jen pro jednoduché problémy. Návrh sítě pro složitější problémy je přitom často doprovázen nepřípustným zjednodušováním optimalizačních algoritmů. Jejich hlavní síla, jíž je prohledávání mnoharozměrného prostoru s více minimy a schopnost dostat se z lokálních minim a skončit v globálním minimu, se pak snižuje. Zvyšuje se tím pravděpodobnost, že topologie skončí někde v lokálním minimu, t.j. že nebude ideální. Bohužel výpočetní nároky těchto metod lepší prohledávání možných topologií neumožňují.

Kromě počtu vrstev, počtu neuronů ve vrstvách a existence spojení neuronů může být optimalizováno mnoho dalších věcí. Ani optimalizovaná funkce nemusí být pouze sumou odchylek chyb předpovědí už známých faktů. Síť může být současně optimalizována např. na rychlost učení, nebo na malý počet spojení mezi neurony, nebo na nízký počet vnitřních neuronů. Existuje i speciální zakódování potenciální sítě tak, aby se lehce měnily počty neuronů a vrstev, viz [1], ale tento postup má zase jiné nedostatky a nestojí na nějaké hluboké teorii, proto jej zde nebudeme uvádět.

■ *Optimalizace váhových a prahových koeficientů* spojení v neuronových sítích (obr. 9.2), která nahrazuje metodu zpětného šíření, je další úlohou často řešenou pomocí evolučních metod. Evoluční metody však pro tuto optimalizaci nejsou nejtypičtější, protože klasická metoda zpětného šíření poskytuje dobré výsledky a je výpočetně méně náročná. Klasické algoritmy optimalizující sumu kvadrátu odchylek založené na derivaci "chybové" (účelové) funkce automaticky končí v nejbližším lokálním optimu. Naštěstí hyperplocha optimalizované funkce většinou nemá příliš mnoho lokálních minim a výsledky metody zpětného šíření jsou většinou přijatelné. Pokud se optimalizace nedaří ani po několika "nástřelech" počátečních vah, stačí často přidat několik skrytých neuronů. Chceme-li však vytvořit efektivní síť s minimem vnitřních neuronů a spojů, jsou evoluční algoritmy nenahraditelné. Dokáží relativně rychle vyhledat váhové a prahové koeficienty blízké



Obrázek 9.2. Zobrazení převodu bitového řetězce na váhové a prahové koeficienty, t.j. reálná čísla. Místo vektoru tvořeného reálnými čísly je pak možné optimalizovat bitový řetězec.

optimálním, trvá jim však dlouho přechod od téměř optimálních vah k optimálním vahám [2,3]. Tento nedostatek se často nahrazuje spojením evolučního algoritmu s metodou zpětného šíření — evoluční algoritmus najde přibližné hodnoty optima, a zpětné šíření dokončí optimalizaci do globálního optima.

Některé modifikace stochastických evolučních algoritmů pracují sice přímo s reálnými proměnnými, přesto však je nutné zde uvést **bitovou reprezentaci proměnných** [4].

Nechť $f(\boldsymbol{x})$ je funkce N proměnných (třeba váhových a prahových koeficientů neuronů) definovaná na kompaktní oblasti $D \subseteq R^N$, kde $R=(-\infty,\infty)$ je množina všech reálných čísel a $\boldsymbol{x}=(x_1,x_2,...,x_N)$ je vektor proměnných. Hledejme globální minimum $\boldsymbol{x}^*$ funkce $f(\boldsymbol{x})$ na oblasti D, $f(\boldsymbol{x}^*) = \min f(\boldsymbol{x})$, pro $\boldsymbol{x} \in D$. Zavedeme novou reprezentaci vektoru $\boldsymbol{x}$, která bude realizována pomocí binárního řetězce obsahujícího symboly 0 a 1. Předpokládejme, že každá proměnná x_i (pro $i=1,2,...,N$) je ohraničená zleva a zprava čísly a resp. b, $a \le x_i \le b$. Dále předpokládejme, že proměnné x jsou určeny s přesností na q dekadických míst za desetinnou čárkou. Potom je x přetransformováno nejprve na celé číslo a následovně na binární číslo. Na binární reprezentaci pak potřebujeme k binárních číslic, přičemž délka k binární reprezentace je určena celočíselným řešením rovnice $(b-a)\,10^q = 2^k$,

$$k = \left\lceil \frac{\ln\left[(b-a)10^q\right]}{\ln 2} \right\rceil, \tag{9.1}$$

kde $\lceil \beta \rceil$ je nejbližší větší celočíselná hodnota reálného čísla β, např. $\lceil 1{,}2 \rceil = \lceil 1{,}9 \rceil = 2$. Každému reálnému číslu $x \in \langle a,b \rangle$, vyjádřenému s přesností q, transformací (9.2) přiřadíme celé číslo $y_{(10)}$, jehož binární reprezentace $y_{(2)}$ obsahuje k binárních číslic.

$$x \to y_{(10)} = \left\lceil \frac{x-a}{b-a}\left(2^k - 1\right) \right\rceil \tag{9.2}$$

Inverzním postupem můžeme vytvořit z binárního čísla reálné číslo z intervalu $\langle a,b \rangle$.

Pro lepší pochopení uvedeme jednoduchý ilustrační příklad. Nechť x je reálné číslo z intervalu $-1 \le x \le 2$, vyjádřené s přesností na dvě dekadická místa za desetinnou čárkou, $q=2$. Pomocí vztahu (9.1) určíme konstantu k, tedy $k=9$. Nechť $x=1{,}12$, potom pomocí (9.2) určíme celé číslo $y_{(10)}=362$, a jeho binární reprezentace je $y_{(2)}=101101010$. Pro ilustraci inverzního postupu budeme studovat $y_{(2)}=110011001$, přiřazené dekadické celé číslo je $y_{(10)}= 409$, a použitím inverzního vztahu k rovnici (9.2) dostaneme $x=1{,}40$. Minimální

(maximální) binární číslo délky k=9 tvaru 000000000 (111111111) odpovídá dolní (horní) hranici intervalu, x=−1,00 (x=2,00). Vektor proměnných $\boldsymbol{x}=(x_1,x_2,...,x_N)$ lze pak vyjádřit bitovým řetězcem sestaveným ze za sebou jdoucích binárních reprezentací proměnných x_i. Zavedením binární reprezentace proměnných s ohraničenou přesností jsme redukovali původně spojitý optimalizační problém na diskrétní optimalizační problém.

■ *Extrahování pravidel z neuronové sítě* je základním problémem neuronových sítí, který se jen částečně daří řešit. Tato situace je akutní ve finančnictví, kde se investoři nervózně strachují o svoje peníze, jejichž investice jsou řízeny pomocí neuronových sítí, v jaderných elektrárnách, kdy jsou operátoři neochotní předat řízení něčemu, co není schopno ani vysvětlit svá rozhodnutí, nebo ve vědě, kdy výzkumníci tajně doufají, že se jim rozšifrováním předpovědí neuronových sítí podaří nalézt nový přírodní zákon. V podstatě nejúspěšnější je dosud přepis zadání s výsledky do tvaru pravidel jako u expertního systému. V principu by za tímto účelem ani nebylo třeba neuronových sítí, stačila by rozsáhlá databanka vstupů s požadovanými výstupy. Neuronové sítě se zde používají, je-li dat relativně málo, na jejich doplnění svými vypočítanými výstupy [5].

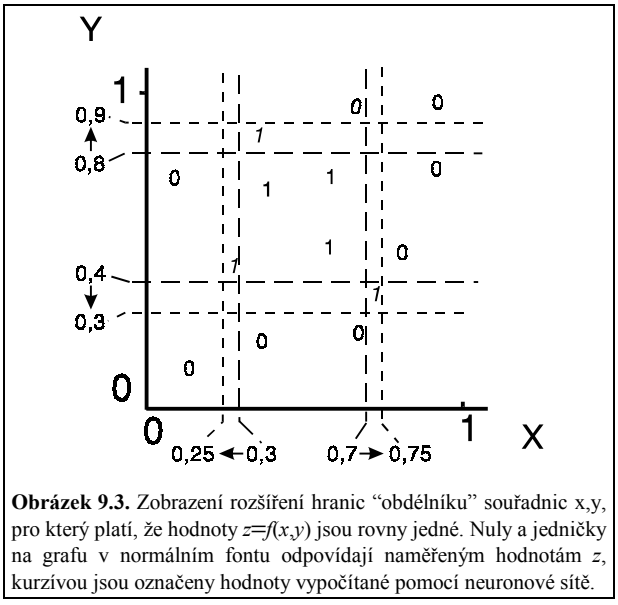
Velmi zjednodušeně si postup získávání pravidel můžeme představit z obr. 9.3, kdy dvě nezávislé proměnné x a y dávají dva možné výsledky pro z, 0 nebo 1. Pravidlo určující maximální možný rozsah výskyt hodnot 1 pak můžeme vytvořit například následovně:

IF((x>=0,3 AND x<=0,7) AND ((y>=0,4 AND y<=0,8)) THEN z=1

Lepším pravidlem je však takové, které má větší hranice, např.

IF((x>=0,25 AND x<=0,75) AND ((y>=0,3 AND y<=0,9)) THEN z=1

Vytváření a posouvání rozsahu hodnot těchto pravidel je otázkou zakódování, všechna pravidla i se svými hodnotami jsou zakódována ve formě binárního řetězce. K optimalizaci těchto pravidel se používá genetických algoritmů. Rozsah hodnot pravidel a jejich platnost pak ohodnocuje daný binární řetězec. Pokud však v daném rozsahu hodnot pravidel neexistují naměřená data se známými výstupy, simulují se výstupy pomocí neuronové sítě, a takto vytvořená data se pak používají k ohodnocení platnosti pravidel.



Obrázek 9.3. Zobrazení rozšíření hranic "obdélníku" souřadnic x,y, pro který platí, že hodnoty $z=f(x,y)$ jsou rovny jedné. Nuly a jedničky na grafu v normálním fontu odpovídají naměřeným hodnotám z, kurzívou jsou označeny hodnoty vypočítané pomocí neuronové sítě.

9.2 Přehled a základní vlastnosti stochastických optimalizačních algoritmů

Stochastické optimalizační algoritmy se používají pro optimalizaci mnohoparametrových funkcí s "divokým" průběhem, t.j. s mnoha extrémy, nebo neznámým gradientem. Standardní gradientové metody (např. nejprudšího spádu, sdružených gradientů, proměnné metriky [6,7]) nebo negradientové metody (např. simplexová [7]) nejsou vhodnými přístupy tehdy, když požadujeme nalezení globálního extrému funkce s mnoha extrémy. Tyto metody obvykle konvergují jen k extrému blízko od startovního bodu, a tento extrém už nejsou schopné opustit. Tento nedostatek se obvykle odstraňuje jejich randomizací tak, že se opakovaně náhodně zvolí počáteční řešení optimalizační úlohy a za výsledné řešení se vezme nejlepší výsledek. Stochastičnost tohoto postupu spočívá jen v náhodném výběru počátečního řešení, následovně použitý optimalizační algoritmus je striktně deterministický.

Bohužel se tento přístup u optimalizace neuronových sítí často používá ze striktně statistického pohledu nesprávně. Síť se optimalizuje gradientovou metodou pro tréninkovou množinu, ohodnotí se pomocí testovací množiny, a pokud výsledek nevyhovuje, začne se znovu s jinými náhodně zvolenými startovními vahami. Tento přístup ve svém důsledku zahrnuje vlastně testovací množinu do trénovací. Správný přístup by měl ohodnocovat řešení pouze na základě chyby dosažené pro trénovací množinu a až vybraná natrénovaná síť by měla být testována pomocí testovací množiny. Pokud se výsledek nepodařil, pak

máme smůlu a správně bychom měli nadále použít jinou trénovací a testovací množinu (k čemu nám většinou chybí data).

Stochastické optimalizační algoritmy si zachovávají svoji "stochastičnost" v celém průběhu optimalizačního procesu a ne jen ve výběru počátečního řešení. Navíc pro tyto metody byly dokázány existenční teorémy, které za jistých předpokladů zabezpečují jejich schopnost nalezení globálního extrému (ovšem v nekonečném čase). Programová implementace těchto metod je poměrně jednoduchá. Jednou z hlavních podmínek jejich úspěšného použití je vhodná reprezentace proměnných pomocí řetězce znaků (např. bitového řetězce obsahujícího symboly 0-1) a rychlost výpočtu hodnot účelové funkce v daném bodě. Zvláště tato poslední podmínka podstatně limituje úspěšné použití stochastických optimalizačních metod pro optimalizaci neuronových sítí, jednoduchost je "kompenzována" náročností na výpočetní čas.

U optimalizace neuronových sítí optimalizujeme jak diskrétní, tak i reálné proměnné. Stochastické optimalizační metody nutně fungují pomaleji než jakékoli heuristické přístupy. Pokud nemáme dopředu zadané podmínky pro globální extrém, nikdy nevíme, jestli jsme ho dosáhli a máme-li optimalizaci zastavit. Stochastické optimalizační metody však mají i zásadní výhody: jsou velmi obecně formulované a tedy aplikovatelné téměř na jakýkoli problém a dokáží se dostat z lokálního extrému. Evoluční proces prohledávání prostoru potenciálních řešení vyžaduje rovnováhu dvou cílů:

- *co nejrychleji* najít nejbližší (většinou lokální) optimum v malém okolí výchozího bodu
- *co nejlépe* prohledat prostor všech možných řešení.

Metody se liší svým zaměřením k těmto dvěma cílům, a je zhruba možné je seřadit podle posloupnosti od metod jdoucích k lokálnímu optimu až k metodám prohledávajícím prostor řešení. Tato posloupnost je následovná:

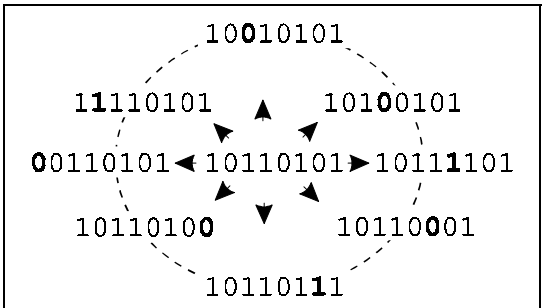
1. stochastický “horolezecký” algoritmus
2. tabu search
3. simulované žíhání
4. evoluční strategie
5. genetické algoritmy

Pro optimalizaci topologie neuronových sítí se zatím prakticky výhradně používají genetické algoritmy, pro optimalizaci váhových a prahových koeficientů se používají genetické algoritmy a simulované žíhání.

Při používání genetických algoritmů jde však spíše o zvyk, ostatní algoritmy nejsou zásadně horší, záleží vždy spíše na nastavení různých parametrů v algoritmech. Ostatně, v poslední době se stále více používá “směsi” algoritmů, t.j. metod, které přebírají a kombinují několik základních přístupů do jednoho. Zde však uvedeme základní algoritmy odděleně.

9.3 Stochastický "horolezecký" algoritmus

Název metody "horolezecký" algoritmus je volným překladem anglického termínu "hill climbing" [4]. Jde vlastně o variantu gradientové metody "bez gradientu", kdy se směr nejprudšího spádu určí kompletním prohledáním okolí. Tento algoritmus také trpí základní nectností gradientových metod, t.j. nejspíše skončí v lokálním optimu, a nedosáhne globálního optima. Vychází se zde z náhodně navrženého řešení. Pro momentálně navržené řešení se generuje pomocí konečného souboru transformací určité okolí (viz obr. 9.4) a funkce se minimalizuje jen v tomto okolí. Získané lokální řešení se použije jako "střed" nového okolí, ve kterém se lokální minimalizace opakuje.



Obrázek 9.4. Bitový řetězec vprostřed slouží jako centrum, jeho okolí je vytvořeno vždy překlopením jednoho bitu. Bitové řetězce okolí se ohodnotí, a nejvýhodnější z nich postupuje jako nový střed do další iterace.

Tento proces se iteračně opakuje předepsaný počet-krát (viz obr. 9.5). V průběhu celé historie algoritmu se zaznamenává nejlepší řešení, které slouží jako výsledné optimální řešení. Základní nevýhodou tohoto algoritmu je, že se po určitém počtu iteračních kroků vrací k lokálně optimálnímu řešení, které se již vyskytlo v předcházejícím kroku (problém zacyklení, viz obr. 9.6). Tento problém se obchází tak, že se algoritmus spustí několikrát s různými náhodně vygenerovanými počátečními řešeními, a poté se vybere nejlepší výsledek.

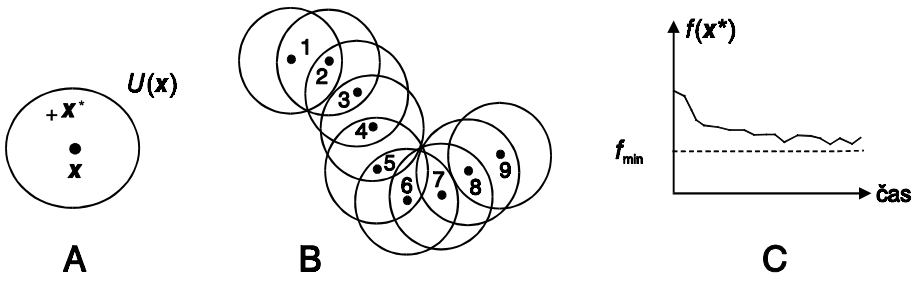
Minimalizace funkce $f(\boldsymbol{x})$ na oblasti D (například oblasti všech možných osmibitových řetězců) spočívá v hledání řešení $\overline{\boldsymbol{x}}$ (např. konkrétního osmibitového řetězce), které poskytuje minimální funkční hodnotu na oblasti D,

$$f(\overline{\boldsymbol{x}}) = \min_{\boldsymbol{x} \in D} f(\boldsymbol{x}). \tag{9.3}$$

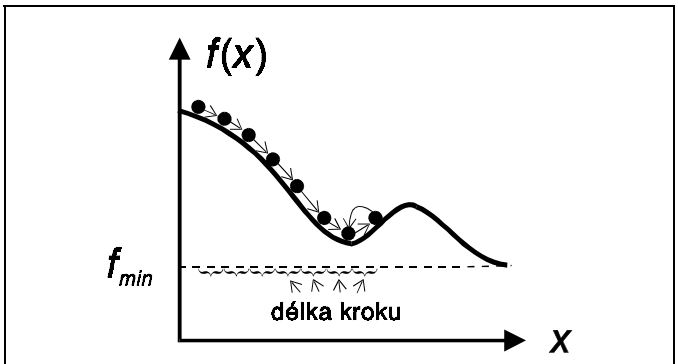
Definujme množinu přípustných transformací S, kde transformace $t \in S$ (například překlopení všech možných bitů řetězce) zobrazuje vektor $\boldsymbol{x} \in D$ na jiný vektor $\boldsymbol{x}' \in D$, kde $\boldsymbol{x}' \neq \boldsymbol{x}$, t: $D \rightarrow D$ pro $\forall\ t \in S$. Postulujme, že pro každou transformaci existuje transformace k ní inverzní. Okolí $U(\boldsymbol{x})$ obsahuje obrazy $\boldsymbol{x}$ vytvořené transformacemi $t \in S$,

$$U(\boldsymbol{x}) = \{t\boldsymbol{x}; \forall t \in S\}. \tag{9.4}$$

Nyní už máme aparát k formulaci horolezeckého algoritmu. Pro náhodně vygenerovaný vektor $\boldsymbol{x}$ hledáme minimum funkce $f(\boldsymbol{x})$ v okolí $U(\boldsymbol{x})$,



Obrázek 9.5. (A) Znázornění okolí $U(\boldsymbol{x})$ momentálního řešení $\boldsymbol{x}$ a nejlepšího řešení z tohoto okolí $\boldsymbol{x}^*$. (B) Znázornění iteračního postupu horolezeckého algoritmu. (C) Hodnoty momentálního nejlepšího řešení v závislosti na čase (iteraci). Ke konci graf začne pravidelně oscilovat.



Obrázek 9.6. Znázornění zacyklení u horolezeckého algoritmu hledajícího minimum funkce. Protože bariéra je větší než délka kroku, algoritmus ji není schopen překonat a řešení znázorněné kuličkou se vrací do již předtím nalezeného "dolíku". Tyto poslední dva kroky se pak do nekonečna opakují, poněvadž algoritmus není schopen zakázat "zpětnou transformaci" do předchozího řešení.

$$f(\boldsymbol{x}^*) = \min_{\boldsymbol{x}' \in U(\boldsymbol{x})} f(\boldsymbol{x}'). \tag{9.5}$$

Získané řešení $\boldsymbol{x}^*$ je použito v následující iteraci algoritmu jako "střed" nového okolí $U(\boldsymbol{x}')$, a tento proces se opakuje předepsaný počet-krát. Pseudo-pascalovský algoritmus vypadá následovně:

“Horolezecký” algoritmus:

1 $\boldsymbol{x}$:=náhodně vygenerovaný vektor;
2 time:=0; f_{min}:=∞;
3 **WHILE** time<$time_{max}$ **DO**
4 **BEGIN** time:=time+1;
5 $f(\boldsymbol{x}^*) := \min_{\boldsymbol{x}' \in U(\boldsymbol{x})} f(\boldsymbol{x}')$
6 **IF** $f(\boldsymbol{x}^*) < f_{min}$ **THEN**
7 **BEGIN** $f_{min} := f(\boldsymbol{x}^*)$;
8 $\boldsymbol{x}_{min} := \boldsymbol{x}^*$;
9 **END**;
10 $\boldsymbol{x} := \boldsymbol{x}^*$;
11 **END**;

Proměnné f_{min} a $\boldsymbol{x}_{min}$ zaznamenávají nejlepší řešení získané v průběhu celého algoritmu opakovaného $time_{max}$-krát. Hlavní omezení tohoto jednoduchého heuristického algoritmu je problém cyklických řešení. Po konečném počtu iteračních kroků se algoritmus vrátí k řešení, které se již vyskytovalo jako lokální řešení v předcházejícím iteračním kroku, přičemž nejlepší zaznamenané řešení je obvykle vzdálené od globálního minima funkce v oblasti *D*.

Kromě “okolí” bitového řetězce získaného překlopením jednoho bitu se také někdy používá jiné operace, tzv. inverze. Tato operace je užitečná především u binární reprezentace reálných čísel, kdy např. čísla 00001111 a 00010000 sousedí v reálné reprezentaci (alespoň při daném rozlišení), ale na přeměnu jednoho v druhé by bylo potřeba pěti překlopení bitu. Proto se někdy zavádí operace, která od daného bitu překlopí všechny následující bity. Tak by se z prvního řetězce dal vyrobit druhý překlopením posledních pěti bitů: Inverze_od_čtvrtého_bitu(00001111) = 00010000.

Slovo “stochastický” v názvu stochastický “horolezecký” algoritmus znamená modifikaci “horolezeckého” algoritmu, při které se začíná několikrát z rozdílného výchozího řešení, a za výsledek se bere nejlepší řešení z několika průběhů. Je možná i modifikace algoritmu, při které se neprohledává celé okolí momentálního řešení, ale pouze jeho náhodně vybraná část.

9.4 Tabu search neboli "zakázané prohledávání"

Koncem osmdesátých let navrhl Prof. Fred Glover z University of Colorado, Boulder [8] nový přístup k řešení problému globálního minima, který nazval tabu search. V současnosti patří tato metoda mezi hlavní hity v oblasti operačního výzkumu a algoritmů na řešení kombinatorických úloh a hledání globálního optima. Její základní myšlenka je velmi jednoduchá. Vychází z horolezeckého algoritmu, kde se snaží odstranit problém zacyklení. Do horolezeckého algoritmu je zavedená tzv. krátkodobá paměť, která si pro určitý krátký interval předcházející historie algoritmu pamatuje inverzní transformace k lokálně optimálním transformacím řešení použitým k získání nových "středů" pro jednotlivé iterace. Tyto inverzní transformace jsou zakázány (tabu) při tvorbě nového okolí pro dané aktuální řešení. Tímto jednoduchým způsobem je možné podstatně omezit výskyt zacyklení při pádu do lokálního minima. Takto modifikovaný horolezecký algoritmus systematicky prohledává celou oblast, ve které hledáme globální minimum funkce.

Glover [8,9] navrhl další metody intenzifikace a diverzifikace zakázaného prohledávání, konkrétně přístup tzv. dlouhodobé paměti, ve kterém se pokutují (znevýhodňují) při ohodnocení funkce f ty transformace, které sice nepatří do krátkodobé paměti, ale často se vyskytovaly v předcházející historii algoritmu. Metoda je aktivně rozvíjena, její teoretické základy nejsou však zatím dost solidní, aby daly odpověď např. na otázku, za jakých podmínek metoda zkonverguje ke globálnímu optimu. V současné době jde spíše o soubor algoritmických triků a heuristik, které jsou však vysoce numericky efektivní.

Hlavní myšlenka heuristiky je algoritmicky realizována v zakázaném seznamu T (tabu list). Ten zastupuje krátkodobou paměť, která dočasně obsahuje inverzní transformace k transformacím použitým v předcházejících iteracích. Zakázaný seznam transformací $T \subseteq S$, maximální velikosti k, je sestrojen a obnovován v průběhu chodu celého algoritmu. Jestliže transformace t patří do zakázaného seznamu, $t \in T$, pak se nemůže používat v lokální minimalizaci v rámci okolí aktuálního řešení $\boldsymbol{x}$. Při inicializaci algoritmu je zakázaný seznam prázdný, po každé iteraci se do zakázaného seznamu přidá transformace inverzní k transformaci, která poskytla lokálně optimální současné řešení přeměnou řešení z předcházející iterace (např. se do zakázaného seznamu zapíše pořadové číslo bitu, který by se nadále neměl měnit — tabu list musí být kratší než počet možných transformací). Po k iteracích zakázaný seznam už obsahuje k transformací, a každé další dodání nové transformace je doprovázené vyloučením momentálně "nejstarší" transformace (dodané právě před k iteracemi). Říkáme, že zakázaný seznam se cyklicky obnovuje,

$$T := \begin{cases} T \cup \left\{ t^{*-1} \right\} & (\text{pro } |T| < k) \\ \left(T \cup \left\{ t^{*-1} \right\} \right) \setminus \hat{t} & (\text{pro } |T| = k) \end{cases} \tag{9.6}$$

kde t^* je transformace, která vytváří lokálně optimální řešení $\boldsymbol{x}^* = t^* \boldsymbol{x}$, a $\hat{t}$ je "nejstarší" transformace zavedená do zakázaného seznamu právě před k iteracemi. Numerické zkušenosti s algoritmem zakázaného prohledávání ukazují, že velikost k zakázaného seznamu je velmi důležitým parametrem ovlivňujícím možnost vymanit se při prohledávání oblasti D z lokálních minim. Jestliže je parametr k malý, pak se může vyskytnout zacyklení algoritmu, stejně jako u klasického horolezeckého algoritmu.

Zacyklení se sice neopakuje v sousedních dvou krocích, ale řešení se může opakovat po více krocích. V případě, že je parametr k velký, potom při prohledávání oblasti D s velkou pravděpodobností "přeskočíme" hluboká údolí funkce $f(\boldsymbol{x})$, t.j. vynecháváme nadějná lokální minima, která mohou být globálními minimy. Jednou z populárních úprav algoritmu je adaptace délky zakázaného seznamu podle dosud dosažených výsledků.

Zakázaný seznam se používá ke konstrukci modifikovaného okolí aktuálního řešení $\boldsymbol{x}$,

$$U_T(\boldsymbol{x}) = \{ \boldsymbol{x}'; \forall t \in S \backslash T: \boldsymbol{x}' = t\boldsymbol{x} \}. \tag{9.7}$$

Toto okolí obsahuje vektory $\boldsymbol{x}' \in D$, které jsou vytvořeny pomocí transformací z množiny S nepatřících do zakázaného seznamu T.

Lokální minimalizace se vykonává v modifikovaném okolí $U_T(\boldsymbol{x})$ s výjimkou tzv. aspiračního kritéria. Toto kritérium porušuje restrikci zakázaného seznamu tehdy, existuje-li taková transformace $t \in S$, že vektor $\boldsymbol{x}' = t\boldsymbol{x}$ poskytuje nižší funkční hodnotu, než má dočasně nejlepší řešení.

Pascalovský pseudokód metody zakázaného prohledávání je prezentován ve formě algoritmu na následující straně.

Algoritmus zakázaného prohledávání je podobný předchozímu horolezeckému algoritmu. Hlavní rozdíl spočívá v lokálním prohledávání kombinova-ném s aspiračním kritériem uvedeným na řádcích 5-9. V řádku 12 je obnovo-ván zakázaný seznam, viz (9.6).

Algoritmus zakázaného prohledávání byl zatím diskutován jen ve své základní podobě. Možnosti jeho dalších úprav jsou široce diskutovány v literatuře [9,10]. Přístup založený na koncepci dlouhodobé paměti patří mezi základní prostředky intenzifikace a diverzifikace algoritmu směrem k získání globálního minima. Využívá možnost odmítnutí (pokutování) transformací, které se v předcházejícím průběhu algoritmu vyskytly nejčastěji (dlouhodobá paměť). Hledání lokálního minima v modifikovaném okolí $U_T(\boldsymbol{x})$ je v tomto přístupu založeno nejen na změnách funkce $f(\boldsymbol{x})$, ale i na předcházející historii algoritmu. Jednoduchá realizace této všeobecné myšlenky je použití frekvencí transformací $\omega(t)$. Při inicializaci algoritmu jsou tyto frekvence nulové, potom v

Tabu search:

```
1    x:=náhodně vygenerovaný vektor;
2    time:=0; f_min:=∞; T:=∅;
3    WHILE time<time_max DO
4    BEGIN time:=time+1; f_loc-min:=∞;
5            FOR t∈S DO
6            BEGIN x′:=tx;
7                    IF f(x′) < f_loc-min AND (t∉T OR f(x′) < f_min) THEN
8                    BEGIN x*:= x′; t*:=t; f_loc-min:= f(x′); END;
9            END;
10           IF f_loc-min< f_min THEN BEGIN f_min:= f_loc-min ; x_min := x*; END;
11           x := x*;
12           IF |T|<k THEN T:=T∪{t*^-1} ELSE T:=(T∪{t*^-1})\{ t̂ };
13   END;
```

každém iteračním kroku s výslednou transformací t^* je odpovídající frekvence zvýšena o jednotku, $\omega(t^*) \leftarrow \omega(t^*)+1$. Po předepsaném počtu kroků (obvykle řádově větším než je velikost k zakázaného seznamu T) tyto frekvence určují, jak často byly jednotlivé transformace z S použité v lokální minimalizaci. Frekvence se používají jako pokutové funkce při hledání minima v okolí $U_T(\boldsymbol{x})$. Vektor $\boldsymbol{x}' = t\boldsymbol{x} \in U_T(\boldsymbol{x})$, kde $t \in S \backslash T$, je akceptován jako dočasně nejlepší řešení, je-li splněna následující podmínka

$$f(\boldsymbol{x}')+\alpha\,\omega(t) < f(\boldsymbol{x}^*) \tag{9.8}$$

kde α je empiricky určená malá konstanta. To znamená, že minimalizace popsaná v řádcích 5-9 ve výše uvedeném algoritmu je realizována pro funkci $f(\boldsymbol{x}')+\alpha\,\omega(t)$, ovšem jako lokálně nejlepší řešení v okolí $U_T(\boldsymbol{x})$ se zaznamenává jen funkční hodnota $f(\boldsymbol{x}')$. Nejčastěji používané transformace jsou penalizovány jako důsledek vysokých hodnot frekvencí. Přístup dlouhodobé paměti dává šanci i jiným transformacím než těm, které i když poskytují lokálně nižší funkční hodnotu $f(\boldsymbol{x}')$, jsou penalizovány v důsledku jejich frekventovaného výskytu v předcházející dlouhodobé historii algoritmu.

Technika zakázaného prohledávání může být použita jak v klasickém spojení s horolezeckým algoritmem, tak i v kombinaci s jinými algoritmy, např. se simulovaným žíháním nebo genetickými algoritmy. U těchto metod však není natolik efektivní, tyto metody neprohledávají celé okolí momentálního řešení a pravděpodobnost zapůsobení zakázaného seznamu je tedy dost malá.

9.5 Simulované žíhání (simulated annealing)

Počátkem 80-tých let Kirkpatrick, Gelatt a Vecchi [11] (Watson Research Center of the IBM, USA) a nezávisle Černý [12] (MFF UK v Bratislavě) dostali geniální nápad, že problém hledání globálního minima může být realizovaný podobným způsobem jako žíhání tuhého tělesa. Přístup simulovaného žíhání [13,14] je založen na "simulování" fyzikálních procesů probíhajících při odstraňování defektů krystalové mřížky. Krystal se zahřeje na určitou (vysokou) teplotu a potom se pomalu ochlazuje (žíhá). Defekty krystalové mřížky mají vysokou pravděpodobnost zániku. Ochlazování systému zabez-pečí, že pravděpodobnost vzniku nových defektů klesá na malou hodnotu.

V simulovaném žíhání je "krystal" reprezentován binárním řetězcem $\boldsymbol{x}$, tomuto řetězci můžeme přiřadit "energii krystalu" — funkční hodnotu $f(\boldsymbol{x})$. Z výše uvedených fyzikálních úvah vyplývá, že v procesu žíhání se minimalizuje energie krystalu. Tato skutečnost naznačuje, že v metodě simulovaného žíhání minimalizujeme funkci $f(\boldsymbol{x})$. Aktuální řetězec $\boldsymbol{x}$ je náhodně přeměněný na nový řetězec $\boldsymbol{x}'$. Tento proces by měl najít nový řetězec z okolí původního řetězce, ale vzhledem k tomu, že nyní nepotřebujeme prohledávat celé okolí, okolí může být zadefinováno mnohem šířeji a volněji, než tomu bylo třeba u horolezeckého algoritmu. Nový řetězec $\boldsymbol{x}'$ nahradí původní řetězec v následném procesu simulovaného žíhání s pravděpodobností (Metropolisův vzorec [15])

$$\Pr(\boldsymbol{x}' \to \boldsymbol{x}) = \left\{1, \exp\left(-(f(\boldsymbol{x}') - f(\boldsymbol{x}))\,/\,T\right)\right\} \tag{9.9}$$

kde parametr T je formální analogií teploty. Jestliže $f(\boldsymbol{x}') \le f(\boldsymbol{x})$, pravděpodob-nost akceptace je jednotková. V tomto případě je nový řetězec $\boldsymbol{x}'$ automaticky akceptován do dalšího procesu simulovaného žíhání. V případě, že $f(\boldsymbol{x}') > f(\boldsymbol{x})$, pravděpodobnost akceptování $\boldsymbol{x}'$ je menší než jednotková, ale i v tomto případě má nový řetězec šanci pokračovat v simulovaném žíhání. V pseudopascalov-ském kódu má algoritmus simulovaného žíhání následující jednoduchou formu uvedenou na následující straně.

Teplota T je ohraničená maximální a minimální hodnotou, $T_{min} \le T \le T_{max}$, snižování teploty je realizováno ve 14. řádku, kde α je kladné číslo menší než jedna, obvykle α=0,9. Celočíselné proměnné t a k jsou počitadla pro vnější resp. vnitřní WHILE-cyklus. Proměnná t zaznamenává celkový počet "pokusů" simulovaného žíhání pro danou teplotu T, zatímco proměnná k zaznamenává počet úspěšných "pokusů", které byly akceptovány Metropolisovým vzorcem (9.9). Pro volbu konstant t_{max} a k_{max} neexistuje všeobecný předpis. Obvykle je hodnota k_{max} od několika set do několika tisíc a $t_{max} = 10\ k_{max}$. Reálná proměnná *random* (9. řádek) je náhodně generované číslo z intervalu $\langle 0,1)$. Řetězec $\boldsymbol{x}^*$ zaznamenává nejlepší řešení v průběhu celého simulovaného žíhání. Ve všeobecnosti binární řetězec $\boldsymbol{x}$ po skončení simulovaného žíhání nemusí být rovný řetězci $\boldsymbol{x}^*$.

Simulované žíhání:

```
1     x:=náhodně generovaný binární řetězec;
2     T:=T_max; x*:=x; k:=1;
3     WHILE (T>T_min ) and (k>0) DO
4     BEGIN t:=0; k:=0;
5           WHILE (t<t_max) and (k<k_max) DO
6           BEGIN t:=t+1;
7                 x′:=transformace(x);
8                 IF f(x′) ≤ f(x) THEN Pr:=1 ELSE
                  Pr:=exp(–(f(x′) – f(x))/T);
9                 IF random<Pr THEN
10                BEGIN x:=x′; k:=k+1;
11                      IF f(x) < f(x*) THEN x*:=x;
12                END;
13          END;
14    T:=α·T;
15    END;
```

V literatuře [13,14] existuje podrobná teorie simulovaného žíhání. Byly dokázány existenční teorémy, za jakých podmínek simulované žíhání poskytuje globální minimum funkce $f(\boldsymbol{x})$ v definičním oboru $\boldsymbol{x}$. V pracích [16,17] je navržené rozšíření simulovaného žíhání směrem ke genetickému algoritmu. Místo jednoho binárního řetězce se současně optimalizuje simulovaným žíháním malá populace binárních řetězců, které si s malou pravděpodobností vymění informaci operací totožnou s křížením z genetického algoritmu.

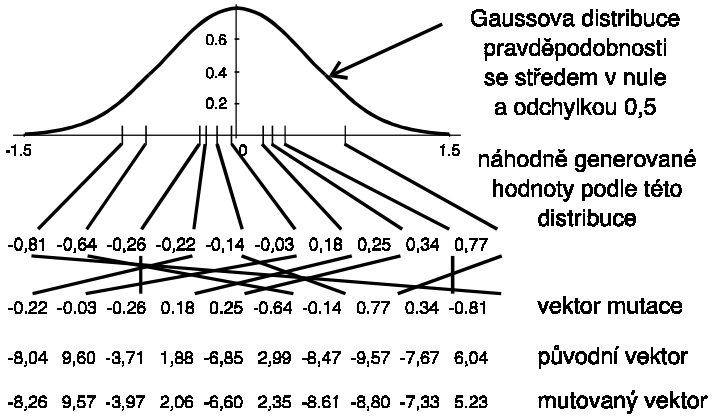
9.6 Evoluční strategie

Evoluční strategie patří historicky mezi první úspěšné stochastické algoritmy. Byla navržena už počátkem 60-tých let Rechenbergem a Schwefelem [18-20, 4]. Vychází ze všeobecných představ přirozeného výběru, ovšem o mnoho vágnějších než například u genetického algoritmu. Navíc, na rozdíl od předcházejících stochastických metod, evoluční strategie není založena na binární reprezentaci proměnných, manipuluje přímo s "reálnou" reprezentací proměnných. Pro neuronové sítě může být tedy použita pouze pro optimalizaci vah nebo jiných proměnných, nikoli pro optimalizaci topologie sítě.

Základem evoluční strategie je následující předpis, který "mutuje" aktuální řešení $\boldsymbol{x}$ na nové řešení $\boldsymbol{x}'$ (viz obr. 9.7),

$$\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{r}(0,\sigma)\,, \tag{9.10}$$

kde $\boldsymbol{r}(0,\sigma)$ je vektor nezávislých náhodných čísel s nulovou střední hodnotou a směrodatnou odchylkou σ.



Obrázek 9.7. Příklad mutace vektoru s reálnými proměnnými za použití Gaussovy distribuce pravděpodobnosti pro určení malé náhodné změny (ne vždy je nutné mutovat všechny hodnoty vektoru, stačí si náhodně vybrat, který prvek se bude mutovat).

Problém akceptace nového řešení $\boldsymbol{x}'$ je striktně deterministický, řešení $\boldsymbol{x}'$ je akceptované (úspěšné), jestliže $f(\boldsymbol{x}') < f(\boldsymbol{x})$. Směrodatná odchylka σ se v průběhu evoluční strategie mění podle pravidla 1/5 úspěšnosti. Nechť $\varphi(k)$ je koeficient úspěšnosti definovaný jako poměr počtu úspěšných mutací v průběhu posledních k iterací k počtu k iterací, ze kterých byla úspěšnost měřena, potom

$$\sigma' = \begin{cases} c_d.\sigma & (\varphi(k) < 1/5) \\ c_i.\sigma & (\varphi(k) > 1/5) \\ \sigma & (\varphi(k) = 1/5) \end{cases} \tag{9.11}$$

kde $c_i > 1$ a $c_d < 1$ řídí zvětšování resp. zmenšování směrodatné odchylky, v literatuře [18] jsou tyto koeficienty specifikované $c_d = 0{,}82$ a $c_i = 1/c_d = 1{,}22$. Algoritmus evoluční strategie v pseudopascalu má tento tvar:

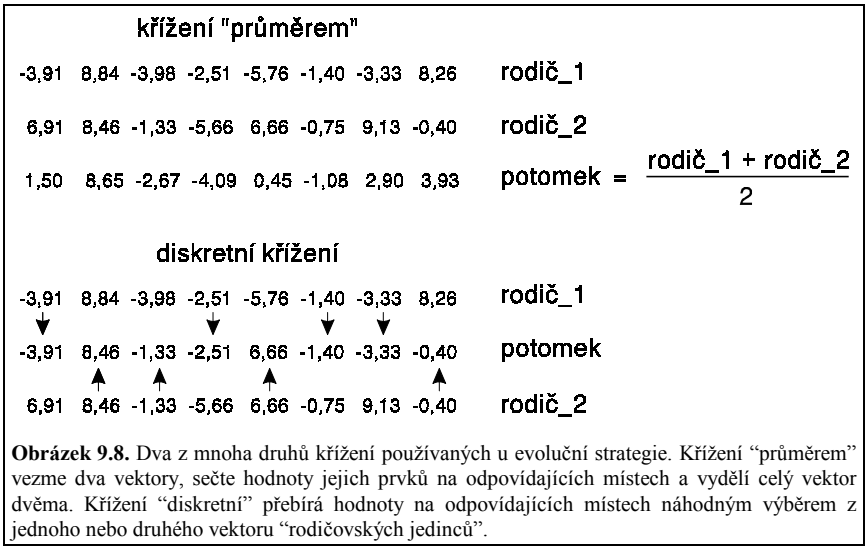
Evoluční strategie:

```
1    x:=náhodně generovaný vektor reálných proměnných;
2    t:=0; σ:=σini; x*:=x;
3    WHILE t<tmax DO
4    BEGIN i:=0; k:=0;
5          WHILE i<imax DO
6          BEGIN i:=i+1; x′:=x+r(0,σ);
7                IF f(x′) < f(x) THEN
8                BEGIN k:=k+1; x:=x′;
9                IF f(x) < f(x*) THEN x*:=x;
10               END;
11         END;
12         IF k/imax<0,2 THEN σ:=cd·σ ELSE IF k/imax>0,2 THEN σ:=ci·σ;
13   END;
```

Proměnná t je počitadlo epoch evoluční strategie. Algoritmus obsahuje dva WHILE-cykly, vnější a vnitřní. Ve vnitřním cyklu se pro dané σ opakuje elementární krok evoluční strategie i_{max}-krát, přičemž proměnná k zaznamenává úspěšnost mutací v tomto vnitřním cyklu. Vnější cyklus, s počitadlem t, aplikuje pro různé hodnoty σ evoluční strategii t_{max}-krát. Směrodatná odchylka σ je v 2. řádku inicializována hodnotou σ_{ini}. V 6. řádku se vykonává modifikace řešení $\boldsymbol{x}$ pomocí generátoru náhodných čísel s nulovou střední hodnotou a se směrodatnou odchylkou σ. Volba základních parametrů evoluční strategie (t_{max}, σ_{ini}, i_{max} a k_{max}) si vyžaduje určité experi-mentování, pomocí kterého tyto konstanty nastavíme. Obvykle je σ_{ini} blízké jedničce, a konstanta i_{max} se rovná řádově tisícům.

Podobně, jako pro simulované žíhání, bylo dokázáno i pro evoluční strategii [18], že potenciálně poskytuje globální extrém optimalizované funkce $f(\boldsymbol{x})$. Schwefelem se spolupracovníky [18-20] byly navrhnuty další sofistikovanější verze evoluční strategie, takže v současnosti je možné už mluvit o celé třídě evolučních strategií. Pracuje se zde poté s celým souborem vektorů $\boldsymbol{x}$, a kromě mutace je zde používáno křížení, t.j. částečná výměna informací mezi vektory reálných čísel (viz obr. 9.8), na rozdíl od křížení bitových řetězců používaného u dále probíraných genetických algoritmů. Nejlepší jedinci se poté vyberou z takto navrhnutých vektorů a z původních "rodičovských" vektorů.

Kromě vlastní hodnoty proměnné ve vektoru může být každá proměnná charakterizována i vektorem "strategických" proměnných. Totiž, některé proměnné jistě mají větší vliv na hodnotu funkce než jiné proměnné, a proto by i jejich změny měly mít jiné měřítko. To může být zabezpečeno tím, že každá proměnná má svůj vlastní rozptyl. Pro dvě proměnné potom vrstevnice pravděpodobnosti umístění mutovaného vektoru nepředstavují kružnici, ale elipsu. Tato elipsa však je orientována ve směru souřadnicových os. Můžeme si však představit, že optimum není umístěno ve směru delší osy elipsy, ale někde našikmo. Ideální by pak bylo, kdybychom mohli tuto elipsu vrstevnic pravděpodobností umístění mutovaného vektoru natočit tak, aby hlavní osa elipsy směřovala k optimu. Tak by mutovaný vektor měl největší šanci co nejvíce se přiblížit k optimu. Toto natočení lze zajistit kovariancemi. Vektor "strategických" proměnných tedy může pro n-rozměrný vektor $\boldsymbol{x}$ zahrnovat n rozptylů $c_{ii} = \sigma_i^2$ stejně jako $n(n-1)/2$ kovariancí c_{ij} zobecněné n-rozměrné normální distribuce s hustotou pravděpodobnosti vektoru mutací z



Obrázek 9.8. Dva z mnoha druhů křížení používaných u evoluční strategie. Křížení "průměrem" vezme dva vektory, sečte hodnoty jejich prvků na odpovídajících místech a vydělí celý vektor dvěma. Křížení "diskretní" přebírá hodnoty na odpovídajících místech náhodným výběrem z jednoho nebo druhého vektoru "rodičovských jedinců".

$$p(z) = \sqrt{\frac{\det A}{(2\pi)^n}} \exp\left(-\frac{1}{2} z^T A z\right) \tag{9.12}$$

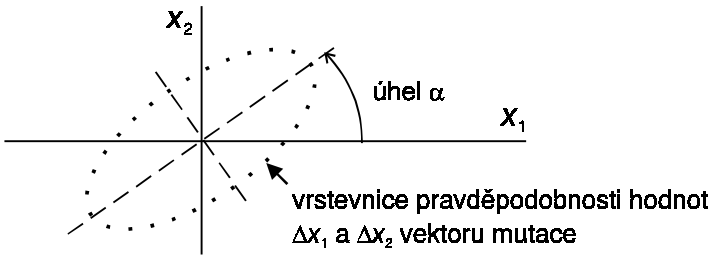
Pro zajištění kladnosti a konečnosti kovarianční matice A^{-1} algoritmus používá odpovídající rotační úhly α_j ($0 \le \alpha_j \le \pi$) místo koeficientů c_{ij}. Mutace jsou potom prováděny následovně

$$\begin{aligned} \sigma_i' &= \sigma_i \exp(\tau_0 \Delta\sigma_0) \exp(\tau \Delta\sigma_i) \\ \alpha_j' &= \alpha_j + \beta \Delta\alpha_j \\ x_i' &= x_i + z_i(\sigma', \alpha') \end{aligned} \tag{9.13}$$

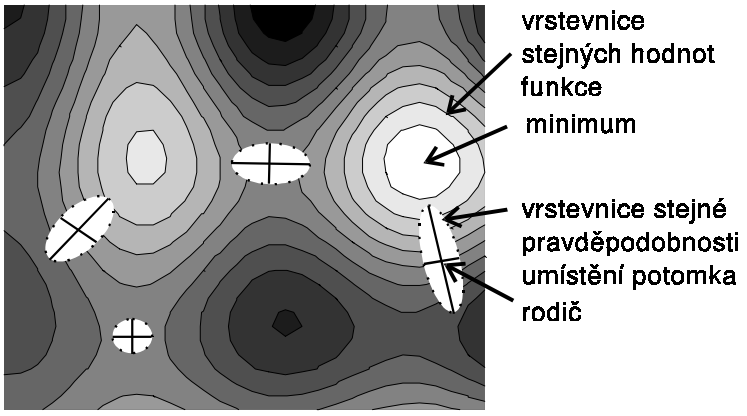
Tímto způsobem jsou mutace proměnných korelovány pomocí hodnot vektoru α, a σ poskytuje "měřítko" lineární metriky. Mutace $\Delta\sigma$ a $\Delta\alpha$ mají opět normální distribuci se středem v nule a směrodatnou odchylkou rovnou jedné, a konstanty $\tau_0 \propto 1/\sqrt{2\sqrt{n}}$ a $\tau \propto 1/\sqrt{2n}$ a $\beta \approx 0{,}0873$ ($=5^o$). Hodnota $\Delta\sigma_0$ reduko-vaná násobením τ_0 je globálním parametrem, zatímco $\Delta\sigma_i$ je individuálním parametrem generovaným zvlášť pro každou proměnnou z vektoru $\boldsymbol{x}$, dovolujíc tak individuální změny "průměrných" změn σ_i každé proměnné x_i vektoru $\boldsymbol{x}$. Bohužel, generovat náhodný vektor mutací s distribucí pravděpodobnosti $p(z)$ uvedenou v (9.12) nemusí být triviálním problémem. V praxi se tento problém nahrazuje postupnou rotací. Řekněme, že máme dvě proměnné x_1 a x_2 se směrodatnými odchylkami σ_1 a σ_2, takže vrstevnice stejných hodnot pravděpodobnosti by tvořily elipsu s osami rovnoběžnými s hlavními osami. Korelační koeficient c_{12} odpovídá úhlu α, o který se tato elipsa pravděpodobnosti pootočí (viz obr. 9.9 a obr. 9.10).

Takže jsou-li původní odchylky proměnných x_1 a x_2 se směrodatnými odchylkami σ_1 a σ_2 rovny Δx_1 a Δx_2, pak odchylky upravené pomocí korelačního koeficientu mají tvar $\Delta x_1' = \Delta x_1 \cos\alpha - \Delta x_2 \sin\alpha$, $\Delta x_2' = \Delta x_1 \sin\alpha + \Delta x_2 \cos\alpha$.

Pro tři proměnné by musely být provedeny tři následné rotace, v rovině (Δx_1, Δx_2) o úhel α_1 s výsledkem $\Delta x_1'$ a $\Delta x_2'$, v rovině ($\Delta x_1'$, Δx_3) o úhel α_2 s výsledkem $\Delta x_1''$ a $\Delta x_3'$, a v rovině ($\Delta x_2'$, $\Delta x_3'$) o úhel α_1 s výsledkem $\Delta x_2''$, $\Delta x_3''$. Výsledné změny proměnných by tedy byly $\Delta x_1''$, $\Delta x_2''$ a $\Delta x_3''$.



Obrázek 9.9. Pootočení elipsy stejných hodnot pravděpodobnosti vygenerování změn Δx_1 a Δx_2.



Obrázek 9.10. Zobrazení vrstevnic hodnot funkce dvou proměnných s optimy označenými světlou barvou. Bílé tečkované ovály označují místa stejné pravděpodobnosti umístění potomka vzniklého mutací z rodiče umístěného ve středu oválu označeného křížením os. Delší osa oválů je nasměrována směrem k optimu, tak jak se to "jedinec" naučil v průběhu optimalizace, kdy každá z proměnných si uschovává vlastní hodnotu směrodatné odchylky, a ještě se uschovává výhodný směr korelovaných odchylek.

9.7 Genetické algoritmy

Genetický algoritmus, vycházející ze všeobecných představ Darwinovy teorie přirozeného výběru, byl původně navržen Hollandem [21] jako učící se algoritmus schopný adaptivně reagovat na měnící se prostředí. Ovšem, hned po jeho vzniku se ukázalo [1,22-23], že je vhodnou metodou na hledání globálního minima úloh, které doposud nebyly řešitelné nebo byly řešitelné jen velmi obtížně. Kromě nastavování parametrů pro neuronové sítě mezi úspěšné aplikace genetických algoritmů patří rozvrhy práce pro stroje v továrnách, teorie her v managementu, všechny možné obtížně řešitelné optimalizační problémy multimodálních funkcí ve vědeckotechnických výpočtech — třeba hledání prostorového uspořádání molekul, řízení robotů, rozpoznávací systémy. V informatice jsou genetické algoritmy zvláště populární pro výhodnou možnost implementace na víceprocesorových počítačích, kde každý procesor "obhospodařuje" jeden chromozóm. V nově se rodící informatické disciplíně nazvané Umělý Život, tvořené ponejvíce simulacemi vývoje hnaného Darwinovským požadavkem přežití nejschopnějších, jsou genetické algoritmy integrální součástí většiny aplikací. Další možností využití genetických algoritmů je strojové učení s klasifikačními systémy [23] a umělá inteligence, kde ale za klasifikační postup je možné z nejobecnějšího hlediska považovat třeba jakýkoli počítačový program. V poslední době je populární také genetické programování, které ve své nejjednodušší podobě nehledá pouze ideální nastavení parametrů regresní funkce, ale i funkci samotnou (většinou složenou z několika základních matematických funkcí). Tomuto přístupu se zde však nebudeme věnovat.

Na rozdíl od neuronových sítí, které se snaží "polapit" efektivitu jednotlivého "mozku" (počet neuronů v umělých neuronových sítích je však z technických důvodů o mnoho řádů menší, než je v jakémkoli lidském nebo zvířecím mozku), genetické algoritmy svou stavbu spojují s vývojem celého společenství. Snaží se využít genetických představ o hnacích silách evoluce živé hmoty. Ačkoliv genetické algoritmy nemají již prakticky s biologií nic společného, udržely si biologickou terminologii. Omezíme-li použití genetických algoritmů na optimalizaci funkce, evolucí jsou míněny postupné změny proměnných vedoucí k nalezení extrému funkce. Soubor proměnných vstupních veličin funkce tvoří jedince. Není zde však tak důležitý jedinec, jako postupný vývoj, kooperace a fungování populace — souboru jedinců. Neúspěšní jedinci vymírají, úspěšní přežívají a množí se. Hybnou silou změn jsou mutace a křížení (výměna "genetické" informace mezi jedinci). Každý jedinec je v algoritmu reprezentován svým lineárně uspořádaným informačním obsahem (formálně nazývaným chromozóm).

Nechť P je populace obsahující M chromozómů. Pod chromozómem budeme rozumět binární řetězec délky, která je konstantní pro všechny chromozómy z populace P. Chceme-li optimalizovat funkci N proměnných, chromozóm odpovídá binární reprezentaci za sebou seřazených binárních reprezentací jednotlivých proměnných. Každý chromozóm $\boldsymbol{x} \in P$ je ohodnocený silou (fitness) $s(\boldsymbol{x})$ tak, že chromozómu $\boldsymbol{x}$ s malou (velkou) funkční hodnotou $f(\boldsymbol{x})$ je přiřazená velká (malá) síla $s(\boldsymbol{x})$. Mezi chromozómy z populace P probíhá proces reprodukce, jehož výsledkem je nová populace P' obsahující stejný počet chromozómů jako původní populace P. Reprodukce se skládá z následujících částí:

(1) Výběr chromozómů. Do procesu reprodukce vstupují dvojice chromo-zómů $\boldsymbol{x},\boldsymbol{x}' \in P$, které jsou náhodně vybrané, přičemž pravděpodobnost výběru je úměrná silám $s(\boldsymbol{x})$ a $s(\boldsymbol{x}')$.

(2) Křížení chromozómů. Vybrané chromozómy $\boldsymbol{x}$ a $\boldsymbol{x}'$ si vymění náhodně vybrané části chromozómů. Nechť $\boldsymbol{x}=(a_1 \ldots a_i \ldots a_N)$ a $\boldsymbol{x}'=(b_1 \ldots b_i \ldots b_N)$ jsou dva chromozómy a index $1 \leq i < N$ je náhodně zvolený. Potom jejich křížením dostaneme dva nové chromozómy $\hat{\boldsymbol{x}}=(a_1 \ldots a_{i-1} b_i \ldots b_N)$ a $\hat{\boldsymbol{x}}'=(b_1 \ldots b_{i-1}\, a_i \ldots a_N)$. To znamená, že chromozómy $\hat{\boldsymbol{x}}$ a $\hat{\boldsymbol{x}}'$ vznikly z původních chromozómů tak, že si vyměnily bitové podřetězce za i-tou komponentou.

(3) Mutace chromozómů. Chromozómy $\hat{\boldsymbol{x}}$ a $\hat{\boldsymbol{x}}'$ se podrobí mutaci, kde se náhodně vybrané komponenty binárních řetězců změní na jejich komplementy, t.j. $0 \rightarrow 1$ a $1 \rightarrow 0$. Pro lepší pochopení tohoto pojmu uvedeme jednoduchý příklad. Nechť (00**1**100**0**1) je bitový řetězec – chromozóm délky 8, v procesu mutací v náhodně vybraných polohách 3 a 7 se mění komponenty, dostaneme nový chromozóm (00**0**100**1**1).

Podstatným rysem genetického algoritmu je jeho úplná stochastičnost, v každém kroku jsou operace důsledně vykonávány náhodně. Genetický algoritmus v pseudopascalu je uveden na následující straně.

Proměnná t je diskrétní čas (počitadlo epoch), genetický algoritmus je ukončený, když $t=t_{\max}$, kde $t_{\max}$ je předepsaný počet epoch genetického algoritmu. P_t v algoritmu označuje populaci chromozómů v čase t, cyklus se opakuje $t_{\max}$-krát, Q je subpopulace chromozómů — potomků, které byly vytvořeny křížením rodičovských chromozómů z předcházející populace P_{t-1} a následnými mutacemi. Rodičovské chromozómy se vybírají "kvazináhodně", pravděpodobnost výběru je úměrná jejich síle. Symbol R označuje subpopulaci náhodně vybraných chromozómů s nejmenší silou. Nová populace P_t je vytvořena z populace P_{t-1} tak, že nové chromozómy vytěsní část původních chromozómů (v množinovém formalismu vyjádřené příkazem $P_t:=(P_{t-1} \setminus R) \cup Q$;). Počty chromozómů subpopulací Q a R jsou ohraničeny podmínkami $|Q|<<|P_t|$ a $|Q|=|R|$, t.j. chromozómů — potomků je ve většině aplikací podstatně méně než chromozómů v celé populaci a z populace je vytěsněno právě tolik chromozómů, jako je chromozómů — potomků.

Genetický algoritmus:

```
1    P0:={náhodně vygenerovaná populace chromozómů}; t:=0;
2    Ohodnoť každý chromozóm z populace P0 silou;
3    WHILE t<tmax DO
4    BEGIN t:=t+1;
5    Q:={kvazináhodně vybrané dvojice chromozómů z Pt-1 s největší silou pomocí
     rulety}
6    Q:=Operace_křižení(Q);
7    Q:=Mutace_jednotlivých_chromozómů(Q);
8    Ohodnoť každý chromozóm z Q silou.
9    R:={kvazináhodně vybrané chromozómy z Pt-1 s nejmenší silou};
10   Pt:=(Pt-1 \ R)∪Q;
11   END;
```

Kritickým místem algoritmu je tvorba nových chromozómů v 5-7 řádku. Jak bylo již výše uvedeno, proces reprodukce obsahuje výběr chromozómů, jejich křížení a mutaci. V průběhu celého výpočtu v každé epoše zaznamenáváme minimální hodnotu optimalizované funkce, po jeho ukončení tato hodnota reprezentuje výslednou minimální hodnotu funkce

$f(\boldsymbol{x})$ v oblasti $\boldsymbol{x}=\langle a,b\rangle^N$.

Pokud na rozdíl od dále uvedeného příkladu optimalizujeme pouze váhy neuronové sítě, pak nemusíme nic trénovat a tréninkovou množinu používáme přímo k ohodnocení sítě. Testovací množinu použijeme až k otestování výsledné sítě po skončení genetického algoritmu.

Síla chromozómů je určená podle následujícího postupu [23]: Vypočítáme funkční hodnoty funkce $f(\boldsymbol{x})$, které uspořádáme podle rostoucích funkčních hodnot, t.j. první (poslední) chromozóm má nejmenší (největší) funkční hodnotu. Chromozómu x_i z populace přiřadíme sílu podle vzorce

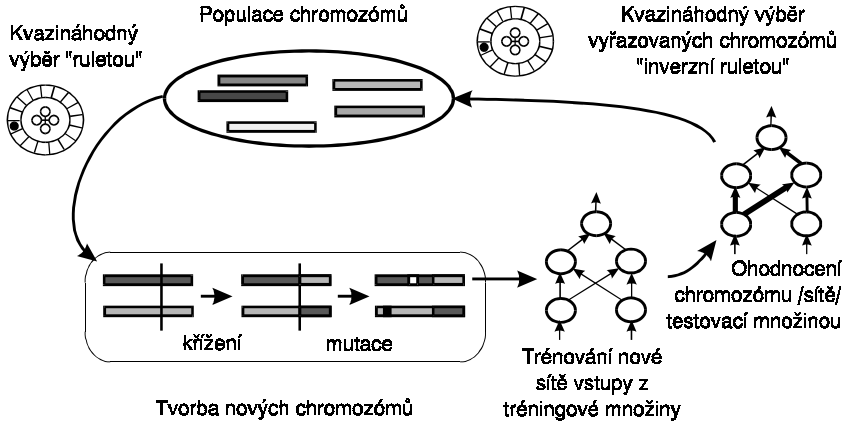
$$s(x_i) = \frac{1}{1-M}((1-\varepsilon)i + \varepsilon - M) \tag{9.14}$$

kde ε je malé kladné číslo, zvolili jsme $\varepsilon = 0{,}05$ a M je počet chromozómů.

Náhodnost výběru chromozómů do procesu reprodukce je realizována tak, aby byla úměrná jejich síle, což se uskutečňuje pomocí tzv. rulety [23]. Předpokládejme, že jsme rozdělili jednotkovou kružnici na M oblouků s délkami úměrnými velikostem sil. Náhodně zvolíme číslo $r \in \langle 0,1)$, potom toto číslo leží na některém oblouku jednotkové kružnice, jeho index odpovídá chromozómu, který je vybrán do procesu reprodukce. Tento postup připomíná ruletu, kulička se zastaví v oblouku (poloze) s pravděpodobností úměrnou délce oblouku — síle chromozómu. Ruleta nám s největší pravděpodobností vybírá ty chromozómy, které mají největší sílu. Ovšem i chromozómy s malou silou mají určitou malou šanci být vybrány do reprodukce. Tímto jednoduchým způsobem se realizuje základní atribut přirozeného výběru, a to, že do procesu reprodukce vstupují jedinci — chromozómy náhodně s pravděpodobností úměrnou jejich síle (viz obr. 9.11).

Nechť $\boldsymbol{x}$ a $\boldsymbol{x}'$ je dvojice chromozómů z populace, která byla vybrána ruletou. První krok reprodukce je křížení těchto chromozómů. Pokud se v populaci nahrazují jen některé chromozómy, a zbylé přežívají, křížení se provádí automaticky u všech vybraných chromozómů (jinak je možné přežívání některých chromozómů do další generace nahradit např. tím, že křížení se neprovádí u všech vybraných chromozómů, některé náhodně vybrané dvojice se jen kopírují). Mutace výsledných chromozómů z křížení má také stochastický charakter. Postupně se realizuje od první komponenty chromozómu (binárního řetězce obsahujícího 0 a 1) s pravděpodobností P_{mut}. Jestliže náhodné číslo $r \in \langle 0,1)$ vyhovuje podmínce $r \le P_{mut}$, potom danou komponentu zaměníme za její komplement, t.j. $0 \to 1$ nebo $1 \to 0$. Ukazuje se, že pravděpodobnost mutací musí být poměrně malá (i v přírodě jsou mutace velmi vzácné), např. $P_{mut} = 0{,}0001$ (t.j. v binárním řetězci se změní jen 0,01% komponent).

Mutace je v obecnosti malá náhodná změna jedné či několika proměnných (prvků chromozómu), která ovlivní řešení, ať už kladně nebo záporně. Může přitom jít i o reprezentaci dat reálnými čísly. Mutace je nutná k tomu, aby se zamezilo přílišné specializaci (t.j. zapadnutí celé populace řešení do jednoho suboptimálního minima), aby vždy byla možnost vytvoření zásadně nových chromozómů odpovídajících lepšímu řešení. Mutace přinášejí do chromozómů novou informaci. Kdyby však existovaly pouze mutace, genetický proces by se nelišil od metody náhodného prohledávání. Efektivita je zajištěna rekombinací neboli křížením, což je smixování dvou rodičovských chromozómů (souborů proměnných funkce) tak, aby vytvořily jiné dva chromozómy vzájemnou výměnou



Obrázek 9.11. Návrh topologie neuronové sítě pomocí genetického algoritmu. Každý z chromozómů je převeden na neuronovou síť, ta je inicializována náhodnými vahami, které jsou optimalizovány klasickým učícím algoritmem pomocí tréninkové množiny. Síť je poté testována testovací množinou a na základě výsledné chyby je nepřímo úměrně stanovena síla chromozómu. Tento chromozóm pak vytěsní kvazináhodně vybraný slabší chromozóm z populace. Výběr nových chromozómů ke křížení a mutaci se také děje pomocí "rulety".

některých hodnot proměnných. Reprodukcí dvou silných jedinců (chromozómů odpovídajících lepším řešením) dostaneme s vysokou pravděpodobností i silné potomky. Populace chromozómů je sada souborů hodnot proměnných, kde každý soubor může dávat jinou funkční hodnotu. Jednotlivé chromozómy bojují o přežití, t.j. do následující populace (nové generace) jsou vybírány s větší pravděpodobností soubory hodnot proměnných odpovídající lepšímu řešení (extrému funkce).

Otázka je, proč vlastně genetické algoritmy fungují. Příčinu je třeba hledat především ve výměně informací mezi chromozómy. Můžeme si představit, že některé pozice ovlivňují řešení více než jiné, a že kombinace těchto pozic může působit nelineárně, t.j. výsledek není součtem vlivu izolovaných změn, ale spíše jejich násobkem. Důvod, proč v tvorbě takto výhodně součinných pozic funguje tak dobře křížení, hledáme pomocí tzv. schémat. Ve schématu si v bitovém řetězci nahradíme ty hodnoty, na kterých hodnota funkce tolik nezáleží, hvězdičkami. U hvězdiček je jedno, jestli bitový řetězec nabývá hodnot 0 nebo 1. Genetický algoritmus se snaží prohledávat mnoho oblastí fázového prostoru zároveň. Nechť dva podřetězce *R* a *S*, které se vyskytují v dvou různých nepřekrývajících se částech řetězce, podstatně zvyšují sílu chromozómu. Jestli pravděpodobnost výskytu každého z nich je 10^{-6}, potom se pravděpodobnost jejich současného výskytu rovná součinu $10^{-6} \cdot 10^{-6} = 10^{-12}$. Současný výskyt *R* a *S* v chromozómu jen v důsledku mutací je tedy vysoce nepravděpodobný. Avšak vyskytují-li se chromozómy s izolovanými podřetězci *R* a *S* už v populaci, křížení sloučí tyto podřetězce s velkou pravděpodobností do jednoho chromozómu.

Důvodem výrazně vyšší efektivity genetických algoritmů oproti náhodnému výběru je fakt, že chromozóm lze považovat za umístěný ve více schématech. Máme-li chromozóm 11010, je tento jediný chromozóm členem schématu 11****, ale také třeba schématu **0*0

a mnoha dalších. Relativně malý počet chromozómů tak mapuje velké množství schémat. Tento implicitní paralelismus je patrně jedním z klíčových faktorů úspěchu genetických algoritmů. Křížení tento efekt implicitního paralelismu poněkud komplikuje, poněvadž sice schémata spojuje, ale také často rozbíjí, tím pravděpodobněji, čím více jsou schémata rozložena po délce chromozómu. Tím, že dochází k rozbití schémat, dochází ale také k vytvoření nových schémat a výsledné chromozómy mohou patřit do oblasti, do které nepatřil ani jeden z rodičů. Vzhledem k tomu, že méně často dochází k rozbití schémat nul a jedniček umístěných blízko sebe, genetické algoritmy zvlášť dobře prohledávají všechny takto definované oblasti. Dochází tak k zvýšené produkci nelineárně podporovaných oblastí. To je vyjádřené Hollandovou větou z r. 1975, která tvoří teoretický základ genetických algoritmů [21]:

Věta. Nechť r je střední hodnota síly všech chromozómů v populaci, které obsahují schéma, n je počet těchto chromozómů a konečně nechť a je střední hodnota síly všech chromozómů v populaci. Potom očekávaný počet chromozómů obsahujících schéma v následující generaci je $n \cdot r/a - z$ (kde z je počet zániků schématu v důsledku křížení a mutací).

Tato věta říká, že efektivní schéma se vyskytuje v následující generaci s rostoucí frekvencí, a naopak, neefektivní schéma se vyskytuje s klesající frekvencí. Efektivita schématu závisí na poměru r/a, jestliže platí $r/a>1$ (t.j. $a<r$, čili střední síla všech chromozómů populace je menší než střední síla chromozómů obsahujících schéma), počet chromozómů obsahujících schéma roste. V opačném případě, když $r/a<1$, potom počet těchto chromozómů klesá. Tyto jednoduché úvahy jsou založeny na předpokladu, že počet zániků schémat v chromozómech v důsledků křížení nebo mutací je podstatně menší než počet jeho opakovaných vzniků v procesu reprodukce.

Chromozómy se postupně shromažďují v oblastech odpovídajících lepším řešením. To odpovídá schopnosti kombinovat v chromozómech potomků části řešení nacházející se v rodičovských chromozómech. Samozřejmě, jako se kombinují výhodné části řešení, tak se také mohou zkombinovat nevýhodné části řešení, ale takový potomek pravděpodobně vymře. Mutace, jako náhodná změna chromozómu, tvoří jen neporovnatelně malou část změn v porovnání s křížením. Obyčejně se udává, že vhodná průměrná frekvence mutací je jedno náhodné prohození nuly a jedničky na deset tisíc bitů. Mutace neurychluje nalezení řešení, ale spíše zajišťuje možnost další evoluce v případě, že se všechny chromozómy v průběhu generací následkem křížení nahrnou do jedné oblasti.

Částečnou modifikaci křížení nabízí taková výměna informací, kdy se chromozómy "nepřeříznou" na dva díly, ale na tři, a prostřední část se pak vymění. Omezení průzkumu schémat na víceméně souvislé části chromozómů se snaží odstranit operace tzv. inverze [21] (pozor, je rozdílná od inverze bitů uvedené v podkapitole 9.3). Ta může přeskupit chromozómy tak, že vstupy umístěné v původním chromozómu daleko od sebe budou v novém chromozómu u sebe. To odpovídá předefinování schémat, které jsou víceméně souvislé a ve kterých je proto průzkum intenzívnější, poněvadž nejsou tak často rozbíjeny křížením. Avšak tento přístup se nejeví příliš úspěšný, poněvadž je třeba si pamatovat původní pozice před přeskupením chromozómu. Není předem jasné, zda nové souvislé

oblasti budou úspěšnější, a provádí-li se inverze příliš často, je pak výsledkem prohledávání zase "širší" ale méně "hluboké" a genetický algoritmus špatně konverguje.

Proč při definici genetických algoritmů stále zdůrazňujeme náhodnost výběru chromozómů? Kdybychom do reprodukčního procesu vybírali jen chromozómy s největší silou, potom bychom velmi pravděpodobně podstatně ohraničili doménu, na které hledáme optimální výsledek. Genetický algoritmus by se stal velmi "oportunistickým", za dočasný lepší výsledek bychom dostali horší konečný výsledek. Chromozóm s menší silou může stále ještě obsahovat důležitou informaci využitelnou v budoucí evoluci populace. Všechny "částečně urychlující" heuristiky jsou nejen nedostatečné, ale v konečném důsledku i zavádějící, proto se je ani nepokoušíme do genetického algoritmu zabudovat.

Nevýhodou genetických algoritmů při aplikaci na topologii neuronových sítí jsou problémy se smysluplnou výměnou informací při křížení. Konkrétně jde o váhy v neuronové síti. Řekněme, že máme čtyři vstupní a čtyři skryté neurony. Podařilo se nám vygenerovat síť, ve které jsou spojeny vstupní neurony nenulovými vahami pouze s prvními dvěma skrytými neurony, t.j. druhé dva skryté neurony můžeme zanedbat. Vzhledem k tomu, že síť je v podstatě symetrická, může existovat stejně dobrá topologie zanedbávající prvé dva skryté neurony, a používající jen druhé dva skryté neurony. Při křížení pak může nastat situace, že první potomek bude používat všechny čtyři skryté neurony, a druhý potomek bude mít všechny váhy nulové. Existují různé způsoby, jak předcházet tomuto problému křížením jen víceméně podobných jedinců – sítí, ale žádný z nich se nedá považovat za uspokojivý. Pro n skrytých neuronů totiž existuje $n!$ permutací jejich postavení a tedy $n!$ ekvivalentních sítí. Kromě tohoto druhu symetrie existuje i symetrie u vah skrytých neuronů. Je-li přechodová funkce lichá (což většinou je), pak můžeme nahradit všechna znaménka vstupních a výstupních vah skrytého neuronu opačnými znaménky, a výstupy sítě se nezmění. Pro n skrytých neuronů tak máme 2^n strukturně odlišných, ale funkčně identických sítí generovaných takovýmto přehozením znamének. Při křížení těchto sítí pak dochází k nelogičnostem, protože se může lehce stát, že polovinu vah neuronu vezmeme z jedné sítě, polovinu z druhé sítě (kde byly opačná znaménka) a výsledkem je něco, co nebylo ani v jedné síti. Dochází tak spíše k mutaci, než k výměně informací. Celkově je tedy prostor vah 2^n $n!$ větší, než by ve skutečnosti měl být. Příkladem pokusu o eliminaci této redundance je křížení vah se stejným znaménkem u dvojic neuronů se stejným počtem kladných vah a záporných vah. Počet odpovídajících dvojic lze zvýšit případným přehozením všech znamének vah u neuronu.

Genetický algoritmus je definován v zásadě velmi volně a je na uživateli, aby si zvolil formu odpovídající jeho problému. Genetický algoritmus je založen na vhodné reprezentaci dat potenciálního řešení problému a musí obsahovat definici výměny informací, která vytváří z rodičovských řešení nová řešení – potomky. K hlavním parametrům algoritmu patří velikost populace (počet chromozómů) a pravděpodobnost mutace a křížení, případně spolu s metodou (procentem) vymírání méně úspěšných chromozómů tam, kde může část starých jedinců přežívat do nové populace. Ty nejlepší nově vytvořené chromozómy se mohou popřípadě převzít do nové populace všechny, a i nejlepší rodičovské chromozómy mohou případně přežívat (ale jen malé procento z nich, abychom neohraničili prostor prohledávání).

Výhodou genetických algoritmů je jejich obecnost, dají se upravit pro řešení nejrůznějších úkolů. Nevýhody genetických algoritmů vyplývají také z jejich obecné formulace, je zde spousta parametrů pro nastavení, široký výběr reprezentace dat, definicí

mutace a křížení. Z důvodů této obecnosti neexistuje pro genetické algoritmy hlubší teorie, která by zásadně pomohla při výběru reprezentace dat a nastavování parametrů. To je třeba v praxi provádět spíše metodou pokusu a omylu. Přesto se genetické algoritmy stále více využívají — nic zásadně lepšího zatím k dispozici není.

Literatura

[1] S.A. Harp and T. Samad. Genetic Synthesis of Neural Network Architecture. In: L. Davis, editor. *Handbook of Genetic Algorithms*, pp. 202-221, Van Nostrand Reinhold, New York, 1991.

[2] J.D. Schaffer, editor. Proceedings of the *Third International Conference on Genetic Algorithms,* pp. 360-397, Morgan Kaufmann Publishers, Los Altos, CA, 1989.

[3] R.F. Albrecht, C.R. Reeves and N.C. Steele, editors. *Artificial Neural Nets and Genetic Algorithms*, pp. 628-730. Springer Verlag, Wien, 1993.

[4] Z. Michalewicz. *A Genetic Algorithms + Data Structures = Evolution Programs.* Springer Verlag, Berlin, 1992.

[5] 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, editors. *Artificial Neural Nets and Genetic Algorithms*, pp. 664-669, Springer Verlag, Wien, 1993.

[6] L. Lukšan. *Metody s proměnnou metrikou*. Academia, Praha 1990.

[7] A. Brunovská. *Malá optimalizácia. Metódy, programy, príklady*. Alfa, Bratislava, 1990.

[8] F. Glover. Tabu Search-Part I. *ORSA J. Comp*., 1: 190-206, 1989; Tabu Search-Part II. *ORSA J. Comp.,* 2: 4-32, 1990.

[9] F. Glover, editor. Tabu Search. *Annals of Operations Research*, Vol.41, 1993.

[10] F. Glover, M. Laguna. Tabu search. In: C.R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems,* pp. 70-150. Blackwell Scientific Publications, Oxford, 1993.

[11] S. Kirkpatrick, C. D. Gelatt Jr. and M. P. Vecchi. Optimization by simulated annealing. *Science* 220: 671-680 (1983).

[12] J. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *J. Opt. Theory Appl.* 45: 41-51, 1985.

[13] P.M.J. van Laarhoven and E.H.L Aarts. *Simulated Annealing: Theory and Applications*. Reidel, Dordrecht, The Netherlands, 1987.

[14] R.H.J.M. Otten and L.P.P.P. van Ginneken. *The Annealing Algorithm*. Kluwer, Boston, 1989.

[15] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E.J. Teller. Equation for State Calculation for Fast Computing Machines. *J. Chem. Phys.*, 21: 1087-1092, 1953.

[16] J. Pospíchal and V. Kvasnička. Fast Evaluation of Chemical Distance by Simulated-Annealing Algorithm. *J. Chem. Inf. Comput. Sci.*, 33: 879-885, 1993.
[17] V. Kvasnička, J. Pospíchal, and D. Hesek. Augmented simulated annealing algorithm for the TSP. *Central European Journal for Operations Research and Economics*, 2: 307-317,1993.
[18] H.-P. Schwefel. *Numerical Optimization for Computer Models*. Wiley, Chichester, UK, 1981.
[19] H.-P. Schwefel and R. Manner, editors. *Proceedings of the First International Conference on Parallel Problem Solving from Nature.* Dortmund, Germany, 1990.
[20] T. Bäck, F. Hoffmeister, and H.-P. Schwefel. A Survey of Evolution Strategies. In: R.K. Belew, L.B. Booker, editors. *Proceedings of the Fourth International Conference on Genetic Algorithms,* pp. 2-9, Morgan Kaufmann, San Mateo, CA, 1991.
[21] J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.
[22] L. Davis, editor. *Genetic Algorithms and Simulated Annealing*. Morgan Kaufman, Los Altos, 1987.
[23] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.