

OSNOVA:

1. Historický úvod
2. Biologický kontext
 - 2.1. Darwinova teorie přirozeného výběru
3. Úvod do problematiky
 - 3.1. Prohledávací prostor
4. Obecný genetický algoritmus
5. Jednoduchý genetický algoritmus
6. Schémata
 - 6.1. Věta o schématech
 - 6.2. Explorace x exploatace
 - 6.3. Hypotéza o stavebních blocích
 - 6.4. Statická hypotéza o blocích
7. Modifikace
 - 7.1. Inverze
 - 7.2. Křížící body
 - 7.3. Jiné operátory
 - 7.4. Turnaj
 - 7.5. Elitářství
8. Genetické programování
9. Výhody nevýhody GA
 - 9.1. Silné, slabé algoritmy
 - 9.2. Vhodné, nevhodné úlohy
 - 9.3. Nebezpečí GA
 - 9.4. Ukončovací podmínka

1. Historický úvod

Genetické algoritmy (GA) se inspirovaly myšlenkou Darwinovy teorie přirozeného výběru a genetikou, jejíž základy položil J.G.Mendel v 19.století.

První zmínky o evolučních myšlenkách se objevily v 60. letech 20. století, kdy vydal I. Rechenberg svou práci o evolučních strategiích (v originále *Evolutionsstrategie*).

Nejdůležitější osobností se stal John Holland, který obor založil. Ve spolupráci se svými studenty vydal v roce 1975 zásadní knihu *Adaptace v přirozeném a umělém prostředí* (v originále *Adaptation in Natural and Artificial Systems*).

V roce 1992 byly genetické algoritmy (GA) použity na vývoj samostatných programů řešící s větším stupněm volnosti zadaný problém, kde kromě řešení byl znám i jeho postup. Tato metoda se nazývá "genetické programování".

2. Biologické pozadí

Každý žijící organismus se skládá z buněk a téměř každá buňka má jádro. Toto jádro je pro rozmnožování a tím i pro evoluci naprosto klíčové. Skládá se z různých velké sady chromozomů. Chromozóm je řetězec kyseliny deoxyribonukleové, zkráceně DNA. V každé sadě chromozomů je zakódován "předpis na výrobu" celého organismu.

Kompletní genetický materiál (všech chromozómů) se nazývá genom. Informace, která je uložena v jedné sadě chromozómů, se nazývá genotyp. Pokud se podle tohoto předpisu vytvoří organismus, jeho popis (=kompletní informace) tohoto konkrétního jedince se nazývá fenotyp.

DNA je složena z logických částí, genů. Každý gen je nějaký blok DNA, který kóduje jednu vlastnost organismu (např. barvu očí). Množina všech možností pro danou vlastnost (např. hnědé, modré, černé) se nazývá alela.

2.1 Darwinova teorie přirozeného výběru

Podle této teorie je hlavní hybnou silou evoluce přírodní výběr. Přežívají pouze jedinci, kteří jsou na prostředí lépe vybaveni a jsou nejúspěšnější při reprodukci. Tempo evoluce neboli proces vznikání nových druhů se děje pozvolným nepřetržitým způsobem. Hlavním předpokladem pro výběr je proměnlivost znaků organismu, jak mezi jedinci v jedné generaci, tak mezi generacemi různými. Jedinci téhož druhu se od sebe liší svými znaky a přírodní výběr zvýhodňuje nositele odchylek, které se v prostředí lépe uplatnili. Mechanismy vzniku různorodosti jedinců je křížení a náhodná mutace.

3. Úvod do problematiky

3.1 Prohledávací prostor

Nejdříve si trochu zobecníme pojem řešení. Řešení pro nás bude jakákoli přípustná hodnota, který nám na požadovaný problém dá odpověď. U každého nalezeného řešení budeme určovat míru správnosti. A my chceme vyřešit problém, co nejlépe, proto budeme hledat maximum. Prohledávací prostor bude prostor všech řešení, který rozšíříme o vlastnost míru správnosti (později fitness). Každá dvě různá řešení problému budou reprezentována dvěmi různými body v prohledávacím prostoru a každý bod v prostoru reprezentuje nějaké řešení. Tím se naše úloha rovná nalezení maxima v prohledávacím prostoru.

4. Obecný genetický algoritmus

základní pojmy

Genetická informace - zakódované řešení našeho problému

Generace - skupina jedinců

Jedinec - nositel genetické informace (jedno konkrétní zakódované řešení)

Genotyp - genetická informace

Fenotyp - zobrazení genetické informace do konkrétního jedince

Chromozóm - genetická informace ve tvaru řetězce

Alea - určité místo (pozice) v chromozómu

Gen - konkrétní symbol v chromozómu

Fitness - síla jedince v generaci, musí být definována pro všechny jedince

Rodič - jedinec vstupující do rekombinace

Potomek (Dítě) - jedinec, který je výsledkem rekombinace

Rekombinace - křížení a mutace

Křížení - konstrukce nových jedinců (potomků) z vybraných jedinců generace (rodičů)

Mutace - změna hodnoty v chromozómu (=změna genu v chromozómu)

Selekce - výběr jedinců, kteří přežívají v generaci

Konkrétní podoba genotypu vždy záleží na problematice, která je řešena. Nejvíce používané jsou binární řetězce. Pokud si problém žádá nebo se přirozeně nabízí reprezentace genotypu jiná, použijeme ji, budou se nám později lépe konstruovat genetické operace.

Základní algoritmus GA je postaven podle dříve zmíněné teorii přirozeného výběru, tak aby v populaci měli větší šanci přežít lepší jedinci (=lepší řešení problému). Každý jedinec má svoji genetickou informaci, která je ohodnocena funkcí fitness. Jedinci se mohou křížit (vzniká tak nové uspořádání genů) nebo mutovat (mohou vznikat nové geny). Tlak selekce nám zajišťuje, že v populaci budou narůstat stále silnější jedinci, než do doby, kdy populace narazí na přijatelné řešení (určeno v podmínce ukončení), optimum (ideální řešení) nebo lokální maximum (populace se dlouhodobě nezlepšuje), které nebude schopno opustit. Více viz 9.4 Ukončovací podmínka.

Klasický implementační algoritmus:

Poznámka: $G(t)$... populace v čase t

```
t := 0;
```

```
  // nastav čas na nulu
```

```
inicializace  $G(t)$ ;
```

```
  //inicializují, zpravidla náhodně, populaci
```

```
spectiFitness  $G(t)$ ;
```

```

// spočítej zdatnost všech jedinců v populaci
while (not podmínkaUkončení)
    // test podmínky pro ukončení (čas, zdatnost, atd.)
    t := t + 1;
    // zvětši čítač času
    G' := vyberRodice G (t);
    // vyber jedince, kteří budou mít potomky
    krizeni G' (t);
    // rekombinuj "geny" vybraných rodičů
    mutace G' (t);
    // proved' náhodné mutace
    spoctiFitness G' (t);
    // spočítej zdatnost jedinců v nové populaci
    G := provedSelekcniTlak G, G' (t);
    // vyber ty, kteří přežijí - podle zdatnosti
end while

```

5. Jednoduchý genetický algoritmus

Parametry genetického algoritmu:

- reprezentace: binární řetězec
- inicializace: náhodná
- operátory:
 - mutace: náhodná změna 1 bitu
 - křížení: jednobodové
 - výběr: ruletová selekce

Binární řetězec kóduje problém, pokud například stačí pro zachycení všech stavů řešení 24 stavů, zakóduje se řešení do 5 bitů. Nastává však problém, že někteří jedinci nereprezentují smysluplná řešení. Nabízí se několik možností odstranění problému.

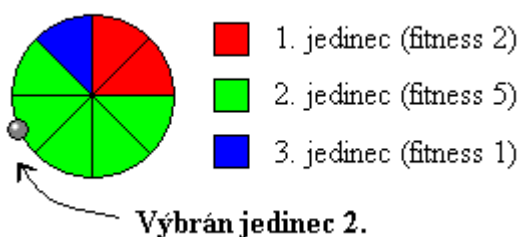
1. Dodefinování fitness nulou, pak jedinec nebude nikdy vybrán a nestane se základem další generace.
2. Pokud to charakter problematiky dovolí, zvětšíme přesnost řešení. Pokud se řešení nalézá v intervalu od 0 do 1, a stačí přesnost na 25 dílků, pak se interval rozdělí na víc dílků v počtu první větší mocniny dvojky, v tomto případě 32.
3. Pokud není problém vhodný k zvětšení přesnosti řešení, lze po provedení operace, po které může vzniknout jedinec mimo rozsah řešení (po rekombinaci), zavolat opravná funkci, která takové jedince detekuje a případně odkáže na provedení "nepovedené" rekombinace znovu.

Tak nebudou "jedinci mimo rozsah" zabírat místo v populaci jedincům platným. Pokud by byl počet řešení ve tvaru 2^n+1 , může být počet platných a neplatných jedinců téměř stejný a tím výpočetní síla algoritmu klesá až na polovinu.

Ruletová selekce

Selekce je „hnacím motorem“ vývoje. Vytváříme konstrukce, které propouští do dalších generací jedince s kombinací kvality (fitness) a náhody (zachování členitosti populace). Čím kvalitnější jedinec je, tím větší má šanci, že bude vybrán do nové generace. To statisticky zaručuje zkvalitňování populace.

Je kolo, jehož obvod se rovná součtu fitness všech jedinců v populaci, kde každému jedinci přísluší právě taková délka jakou má fitness, náhodně se hodí na odvod kola kulička a do které dopadla výseče, ten jedinec je vybrán. Na tomto principu funguje ruleta.

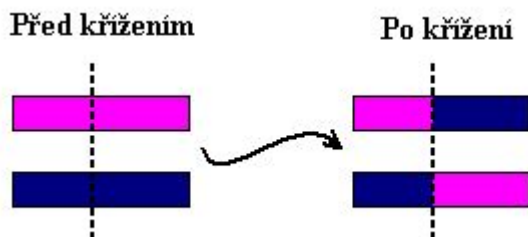


Taková realizace je samozřejmě v praxi naprosto nesmyslná, pouze slouží jako představa úměrnosti šance vybrání k fitness. Z populace se jedinec nebere, pouze se pro křížení nebo mutaci vyrobí lokální kopie, která po modifikaci náleží do nové populace. Pak má jedinec možnost být vybrán několikrát. Tento proces se opakuje do té doby, než je nová populace stejně velká jako původní.

Jednobodové křížení

Křížení je metoda, jejímž použitím získáme nové jedince v řešení, kteří nebyli součástí inicializace populace. Bez ní by generace po nějakém čase byla homogenní. To znamená, že všichni jedinci by byli tvořeny chromozómem nejsilnějšího jedince z náhodné inicializace (neuvažujeme vliv mutací). Tato metoda nám rozšiřuje prohledávaný prostor tím, že se snaží skládat nová řešení z částí již existujících. Obecně platí, že možnost vstupu jedinců do křížení, by měla být úměrná jejich fitness funkci.

Pomocí selekce se vyberou dva jedinci, kteří vstoupí do křížení. Vybere se náhodně jedno místo v chromozómu a na tomto místě se konce jedinců vymění. Jiný název pro toto křížení je cross-over.



Náhodná změna jednoho bitu

Mutace rozšiřuje prohledávaný prostor o řešení, které není možno dosáhnout křížením, protože nebyla součástí inicializace populace.

Máme-li fiktivní populaci sestavenou z jedinců:

jedinec A 01011

jedinec B 11010

jedinec C 01010

jedinec D 00001

jedinec E 10011

Můžeme tuto populaci křížit všemi variantami, ale obsahuje-li ideální řešení 1 na třetí pozici, nenalezneme jej a to z toho důvodu, že naše populace jedničku na třetím neobsahuje a tím ji nebude obsahovat jakákoliv následující generace.

Technikou jakou můžeme dostat do populace jedničku na třetí místo, je právě mutace. Aplikujeme ji na nové jedince (potomky), kdy náhodně dle konstanty mutace (to je pravděpodobnost četnosti mutace) provádíme změny chromozómu.

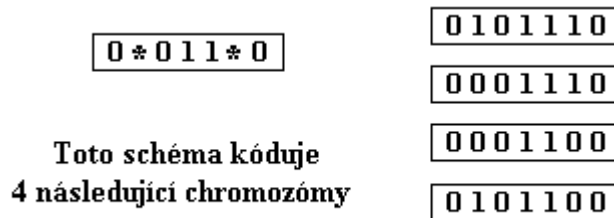
Pro každý bit se počítá mutace zvlášť, pokud se má provést mutace, provede se inverze bitu.



Každá z modifikujících operátorů se neprovádí při každém výběru, pouze s jistou pravděpodobností. Pravděpodobnost pro křížení se doporučuje mezi 0,6 - 1. Pravděpodobnost pro mutaci je mezi 0,05 - 0,15 pro každý bit. Je velmi neobratné počítat zda dojde k mutaci pro každý bit. Při realizaci se většinou zjistí zda v chromozómu vůbec k mutaci dojde a pokud ano, tak se zvolí náhodné místo v chromozómu a tam se mutace provede. Není to přesně podle pravidel, ale je to časově mnohem méně náročné. A pro GA je důležité, že uplatnění operátorů trvá velmi krátce. Takových cyklů se totiž musí provést mnoho.

6. Schémata

Pro matematický popis a analýzu a důkazy úspěšnosti GA slouží teorie tzv. schémat. Zakládá se na poměrně jednoduché myšlence. Sledujeme podmnožiny genotypu řešení a jejich vliv na kvalitu jedince. Schéma je slovo v abecedě $\{0,1,*\}$. Je to v podstatě šablona, která nám má ulehčit manipulaci s větším množstvím podobných chromozómů. Jedno schéma reprezentuje množinu řetězců, kde na pozici metaznaku $*$ se může vyskytnout jak 0 tak i 1.



Tyto chromozómy jsou instancemi daného schématu. Ale také chromozóm 0101110 je instancí schématu $0*01110$, $****110$, $*101***$, atd. To znamená, že při pracování s řetězcem se prohledává více schémat najednou, jde o tzv. implicitní paralelizmus schémat.

Nechť r je počet $*$ ve schématu, potom počet řetězců, které schéma reprezentuje je 2^r . Řetězec délky m je reprezentován 2^m schématy. V populaci o velikosti n je 2^m až $n \cdot 2^m$ schémat.

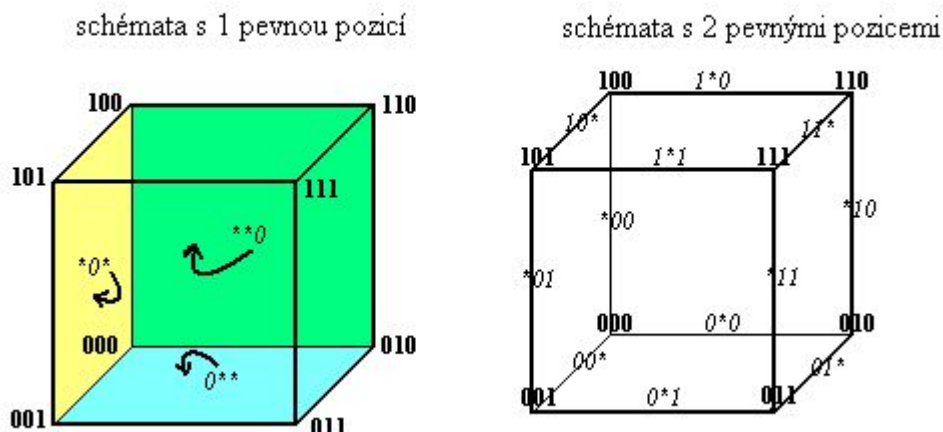
Jak zachází genetický algoritmus se schématy? K popisu nám poslouží následující 3 charakterizující vlastnosti schématu. Pevná pozice je taková, na které je buď 0 nebo 1.

řád schématu - počet pevných pozic ve schématu

definující délka - vzdálenost mezi první a poslední pevnou pozicí

fitness schématu - vyberu všechny jedince z dané konkrétní populace, které odpovídají schématu, ty vyhodnotím a udělám aritmetický průměr

Prohledávací prostor pro řetězce délky 3



6.1 Věta o schématech

Krátká nadprůměrná s malým řádem schémata se v populaci během GA exponenciálně množí.

důkaz:

populace v čase $t, t+1$ - $P(t), P(t+1) = n$ jedinců délky m

$F(S)$ - fitness schématu S

$C(S,t)$ - četnost schématu S v populaci $P(t)$

Cílem je odhadnout $C(S,t+1)$. Rozdělíme důkaz na několik částí podle vlivů, které na schéma působí.

selektce:

$p_s(V)$ - pravděpodobnost vybrání řetězce $V =$ přežití

$$p_s(V) = \frac{F(V)}{F(t)}, \text{ kde } F(t) = \sum_{u \in F(t)} F(u)$$

$p_s(S) = \frac{F(S)}{F(t)}$ - pravděpodobnost vybrání schématu S

$$C(S,t+1) = C(S,t) \cdot n \cdot p_s(S)$$

$$n \cdot p_s(S) = \frac{n \cdot F(S)}{F(t)} = \frac{F(S)}{F(t)_{\text{prům.}}}$$

Pokud je nadprůměrný o e %, potom

$$C(S,t+1) = C(S,t) \cdot (1+e)$$

$$C(S,t+1) = C(S,0) \cdot (1+e)^t$$

Problém je, že se fitness schématu mění!

křížení:

$p_d(S)$ - pravděpodobnost nepřežití

$p_d = \frac{d(S)}{m-1}$, kde $m-1$ je počet mezer v chromozómu

$$p_s = 1 - \frac{d(S)}{m-1}$$

p_c - pravděpodobnost křížení

$$p_s(S) \geq 1 - \frac{p_c \cdot d(S)}{m-1}$$

po selekci a křížení:

$$C(S,t+1) \geq C(S,t) \cdot \frac{F(S)}{F(t)_{\text{prům.}}} \cdot \left[1 - \frac{p_c \cdot d(S)}{m-1} \right]$$

mutace:

p_m - 1 bit nepřežije; $1-p_m$ - bit přežije

$$p_s(S) = (1-p_m)^{o(S)} = \text{přibližně} = 1 - p_m \cdot o(S)$$

přibližně při vlastnosti $p_m \ll 1$ a takové p_m se volí.

vše dohromady:

$$C(S,t+1) \geq C(S,t) \cdot \frac{F(S)}{F(t)_{\text{prům.}}} \cdot \left(1 - \frac{p_c \cdot d(S)}{m-1} - p_m \cdot o(S) \right)$$

Je vidět, že krátká nadprůměrná schémata s malým řádem jednoznačně rostou, exponenciálně. Problém byl v některých nepřesných předpokladech. Například o konstantní fitness schématu, která je bohužel závislá na populaci. Přesto tato věta platí, jen rychlost množení schémat může být přibližná. ■

6.2 Explorace, exploatace

explorace = výzkum, průzkum, bádání v prostoru

exploatace = hospodaření, zužitkování, využití (např. informace)

Nejlépe se problém expl. a expl. demonstruje na specifické úloze. Dvou-ruký bandita je úloha s automatem, který má dvě páky (ruce). Máme n mincí. Do automatu házíme mince a on zpět vyplácí střední hodnotu m_1, m_2 a rozptylem s_1, s_2 pro každou z pák. Musíme své mince co nejlépe využít. Musí se najít hranice využitelnosti, jinými slovy kolik mincí znehodnotíme na nalezení horší ruky (která vyplácí méně) a zbytek investujeme do ruky lepší. Cíl je maximalizovat zisk. Musíme využít ideálně exploraci a exploataci, explorace znamená nalezení horší ruky a exploatace tuto informaci využije.

Bandita má k dispozici také prohledávací prostor, stejně jako GA. Tyto úlohy jsou podobné i v tom, že chceme najít optimální poměr mezi časem věnovaným prozkoumáváním prostoru a využíváním jejich nejlepších míst, to se rovná nalezení globálního maxima.

GA alokuje exponenciálně mnoho místa nadějným schématům a proto řeší problém explorace a exploatace optimálně. Schémata hrají mnohorukého banditu. Kde výhra se rovná místo v populaci.

Jakého banditu ale hrají? Původně se domnívalo, že je to bandita 3^m -ruký, ale není to pravda (kde m je délka řetězce).

Proč spolu některá schémata nesoutěží? Soutěží pouze v tom, zda na jejich pevné pozici bude 1 nebo 0. Z toho vyplývá, že schémata řádu k hrají 2^k -rukého banditu. Ten kdo vyhraje má exponenciálně mnoho míst v populaci.

Platí pouze tehdy, zda jsou GA schopny správně odhadnout fitness schémat.

Uvedu na příkladu:

$F(x) = 2$ pro $x = 111**...*$

$F(x) = 1$ pro $x = 0****...*$

$F(x) = 0$ jinak

pro schéma $F(1**...*) = 0,5$

(průměr z $F(111*...*)=2$, $F(101*...*)=0$, $F(110*...*)=0$, $F(100*...*)=0$)

Jenže GA zde nevzorkují schémata nezávisle, neodhadnou jejich skutečnou (tzv. statickou) fitness. Spočítají fitness z populace, ale skutečná statická je jiná.

V tomto případě dostane v populaci místo schéma $111*...*$ a převáží a tím se spočítaná fitness schématu z populace bude blížit 2, místo statických 0,5.

Navíc selekce nepracuje reálně, schéma nadprůměrné zpočátku nemusí být nadprůměrné stále.

Závěr je, že na rozdíl od bandity, kde ruce byly na sobě naprosto nezávislé, u GA jsou závislé. Přesto se na úloze o banditovi velmi pěkně ukazuje, jak přibližně GA pracují, protože řeší problém expl. x expl. stejně. Pouze při pozdější konvergenci se chovají jinak.

6.3 Hypotéza o stavebních blocích

GA hledá svoje optimum tím, že dává přednost a přemísťuje krátká nadprůměrná schémata nízkého řádu, nazývana stavební bloky.

6.4 Statická hypotéza o blocích.

Grafenstette 91: Lidé předpokládají, že GA konverguje k řešení se skutečně nejlepší statickou fitness, ale ne k těm skutečně existujícím v populacích s nejlepší pozorovanou fitness.

Konvergence: problémem je, že když začneme někam konvergovat, případy schémat jsou závislá.

Navíc existuje rozdíl mezi statickou (správnou) fitness a vypočítanou fitness z populace viz. příklad z odstavce 6.2.

7. Modifikace GA

7.1 Inverze

Vezmeme chromozóm (z GA) jako soubor různých vlastností, kde každá vlastnost má své pevně dané místo a je reprezentována několika pozicemi v řetězci, ne nutně (avšak zpravidla) vedle sebe. V přírodě to tak zdaleka nefunguje. Některá místa v chromozómu nejsou ani nositeli vlastnosti, ale jen pomocné informace, stejně tak nezávisí, jak jsou vlastnosti za sebou uspořádány (to není úplná pravda, ale provázání je natolik daleko od fixního uspořádání, že se vyplatí nahlížet na problém jako by byly bez vazby).

Proto se Holland chtěl díle přiblížit k přírodě a navrhl modifikaci své původní reprezentace chromozómu.

Řetězec má na jedné pozici uspořádanou dvojici, kde první číslo udává pozici v řetězci, druhá jeho hodnotu.

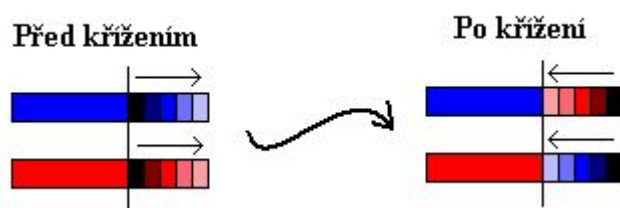
původní reprezentace

nová reprezentace



Hlavním důvodem pro novou reprezentaci byla snaha vyřešit problém s roztroušením vlastností. To znamená, že jednu vlastnost nekódují bity vedle sebe, ale několik bitů roztroušených po řetězci. Pak se nedá uplatnit hypotéza o blocích, protože GA není schopný zajistit, že při křížení mimo danou vlastnost tuto vlastnost zachová, což je hlavní předpoklad HoSB.

Křížení je v tomto případě odlišné od SDA (Single GA = jednoduchý GA). Do křížení vstupují dva rodiče a na náhodném místě si vymění své konce, avšak v obráceném pořadí.



Křížení probíhá v jiných kouscích, proto se k sobě mohou dostat bity původně nesousední a přitom neporušit význam zakódování.

Mutace je velmi obdobná, uhoď se náhodně do bitu a tam provede inverze bitu, kde bit se bere jako druhý člen v uspořádané dvojici.

I když tato metoda řeší problém roztroušených vlastností, v praxi se příliš nepoužívá, nejen proto, že implementace je složitější, ale ani nedávala takové výsledky, jaké se od ní očekávali.

7.2 Křížící se body

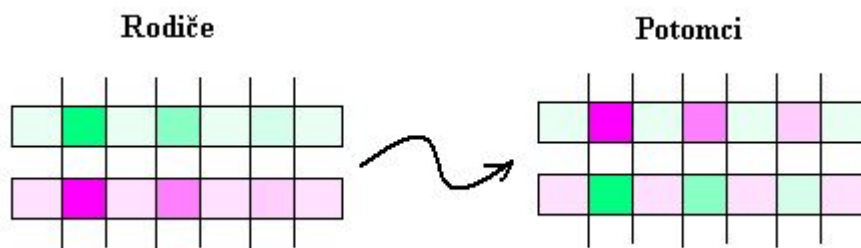
Na základě hypotézy o blocích se zdálo velmi výhodné křížit pouze v některých místech a to právě na hranicích stavebních bloků, nerozbijí se tím některé dílčí úspěchy, kterých GA v průběhu výpočtu dosáhl. Proto se do reprezentace řetězce přidá několik míst pro uložení pozic, kde bude povoleno křížit. V těchto uložených pozicích se dá křížit a necháme takto rozšířený chromozóm vyvíjet pomocí GA. A tak se i vyvíjí i pomocné pozice.



7.3 Jiné operátory křížení

Operátor CX

Dalším známým operátorem je cyclic-cross-over CX, který druhým nejpoužívanějším po CO (cross-over). Další možný název je vícebodové křížení. Vždy po několika bitech se určitý počet bitů vymění s druhým rodičem.

Operátor PMX.

(=Partially mapped crossover).

Uvedu na příkladu:

Rodiče se rozdělí náhodně na tři části, kde podle prostřední se konstruuje přepisovací pravidla. Prostřední část se přehodí jako při mnohobodovém křížení.

A: abc defg hij
 B: aib dfcj geh

Konstrukce přepisovacích pravidel:

$d \Leftrightarrow d$, $e \Leftrightarrow f$, $f \Leftrightarrow c$, $g \Leftrightarrow j$

An: *** dfcj ***

Bn: *** defg ***

Doplní se nekonfliktní geny. Ne Konfliktní geny jsou takové, které se nevyskytují v přepisovacích pravidlech.

An: ab* dfcj hi*
 Bn: aib defg **h

Doplní se geny dle přepisovacích pravidel.

An: abf dfcj hig
 Bn: aib defg jfh

Operátor ERX.

(= Edge recombination crossover), ač tento operátor působí trochu vykonstruovaně, je experimentálně dokázáno, že je nejúspěšnější pro úlohu obchodního cestujícího.

Uvedu ho raději hned na příkladu, protože se bude lépe popisovat.

A: a b c d e f g h i j
 B: a i b d f c j g e h

Sestaví se pro každý gen seznam jeho sousedů.

gen	sousedé
<i>a</i>	<i>b, h, i, j</i>
<i>b</i>	<i>a, c, i, d</i>
<i>c</i>	<i>b, d, f, j</i>

<i>d</i>	<i>c, e, b, f</i>
<i>e</i>	<i>d, f, g, h</i>
<i>f</i>	<i>e, g, d, c</i>
<i>g</i>	<i>f, h, j, e</i>
<i>h</i>	<i>g, i, e, a</i>
<i>i</i>	<i>h, j, a, b</i>
<i>j</i>	<i>i, a, c, g</i>

Nový jedinec je konstruován tak, pro každý gen jsou náhodně vybírání jeho sousedi z jeho seznamu sousedů, přičemž přednost má vždy ten soused, který má svůj seznam sousedů nejkratší.

A(nový): a

gen	sousedé
<i>a</i>	<i>b, h, i, j</i>
<i>b</i>	<i>a, c, i, d</i>
<i>c</i>	<i>b, d, f, j</i>
<i>d</i>	<i>c, e, b, f</i>
<i>e</i>	<i>d, f, g, h</i>
<i>f</i>	<i>e, g, d, c</i>
<i>g</i>	<i>f, h, j, e</i>
<i>h</i>	<i>g, i, e, a</i>
<i>i</i>	<i>h, j, a, b</i>
<i>j</i>	<i>i, a, c, g</i>

Musíme zvolit náhodně mezi prvky *b, h, i, j* - například *j*

A(nový): a j

gen	sousedé
<i>b</i>	<i>c, i, d</i>
<i>c</i>	<i>b, d, f, j</i>
<i>d</i>	<i>c, e, b, f</i>
<i>e</i>	<i>d, f, g, h</i>
<i>f</i>	<i>e, g, d, c</i>
<i>g</i>	<i>f, h, j, e</i>
<i>h</i>	<i>g, i, e</i>
<i>i</i>	<i>h, j, b</i>
<i>j</i>	<i>i, c, g</i>

Nyní má *i* nejméně sousedů, volíme proto *i*, atd.

Je vidět, že tento operátor je velmi zvláštně tvořen, je to dané velkými specifiky ÚOC (úlohy obchodního cestujícího), ale i tak se dá přistupovat ke GA. Jejich výhodou jako slabého algoritmu je, že se dá přizpůsobovat té úloze, na kterou je právě používáme. Tím se algoritmus stává silnějším a rychlejším. Je zřejmé, že právě tento operátor by na hledání maxima funkce příliš nefungoval.. Někdy je používání svých vlastních přizpůsobených operátorů dokonce nutnost.

Pokud charakterizací úlohy se jiný operátor zdá přirozený, je většinou velmi vhodné jej použít. Držet se za každou cenu SGA není příhodné, i když často také funguje.

7.4 Turnaj.

Turnaj je v poslední době nejvíce používanou technikou v aplikacích GA. Hlavním důvodem je jednoduchost implementace při zachování kvality selekčního tlaku.

Z populace jsou vybíráni jednotlivci (nemusí jít obecně o dva) a ti se podrobují „souboji o přežití“, který spočívá v tom, že jedinec z největší fitness funkcí přežívá a postupuje do dalšího kola. Tím se splňuje náš základní předpoklad, aby přežívali jedinci z vyšší kvalitou, ale jedinci vstupující do souboje jsou vybíráni náhodně, což zaručuje různost populace.

Příklad:

Jedinec1.Fitness = 4

Jedinec2.Fitness = 4,2

Jedinec3.Fitness = 4,05

Jedinec4.Fitness = 4,17

souboj	vítěz
(1:2)	2
(3:1)	3
(4:2)	2
(4:3)	4

Z příkladu je patrné, že je-li do souboje vybrán nejsilnější jedinec generace vždy postoupí a nejslabší jedinec generace nikdy nepostoupí

Příklad zapsán ve virtuálním jazyku:

```

For i=1 To Generace.DejStanovenyPocetJedincuVGeneraci
  begin
    JedinecA = Generace.VyberNahodnehoJedince
    JedinecB = Generace.VyberNahodnehoJedince
    if (JedinecA.Fitness >= JedinecB.Fitness) then
      Generace.NovaGenerace.Add(JedinecA);
    else
      Generace.NovaGenerace.Add(JedinecB);
  end;

```

K postupu můžeme připojit i náhodu, tj. vítěz postoupí bude-li náhodné číslo větší než stanovená konstanta. Tato úprava simuluje lineární pořadovou selekci.

7.5 Elitářství

Předešlé techniky obecně nezaručují postup nejlepšího jedince do nové generace. Zvláště v malých populacích je tato ztráta vnímána velmi negativně. Experimenty prokázaly, že ztráta nejlepších jedinců může být opakována a znovuvytvoření jedince není automatické.

Elitářství je jednoduchá technika, kterou vybíráme určitý (velmi malý, např. je vybrán pouze jeden) počet nejlepších jedinců a ti se nepodrobují selekčnímu tlaku, ale postupují do nové generace přímo. Další technikou je uchovávat pouze jednoho nejlepšího jedince, ale naprosto mimo populaci, pouze jako nejlepší nalezené řešení. Tuto verzi nazýváme tajné elitářství. Pomůže nám to nezabřednout do lokálních maxim. O tom podrobněji dále ve výhodách a nevýhodách GA.

8. Genetické programování

Zvláštní skupinou samostatně se vyvíjející, která čerpá z GA je „genetické programování“ (GP). GP má stejné techniky jako GA, ale provádí je nad datovou strukturou, většinou se jedná o n-ární strom. V uzlech jsou objekty ze dvou množin:

množina funkcí (pro každou funkci je známa arita (=počet argumentů))

množina terminálů (konstanty, lze je chápat jako funkce s aritou 0).

Funkce mohou být :

aritmetické (+, -, *, / ...)

algebraické (sin, cos, exp, log, ...)

logické (not, and, or, xor...)

podmíněné operátory (if – then – else, ...)

Terminální symboly mohou být:

vstupní proměnné programu

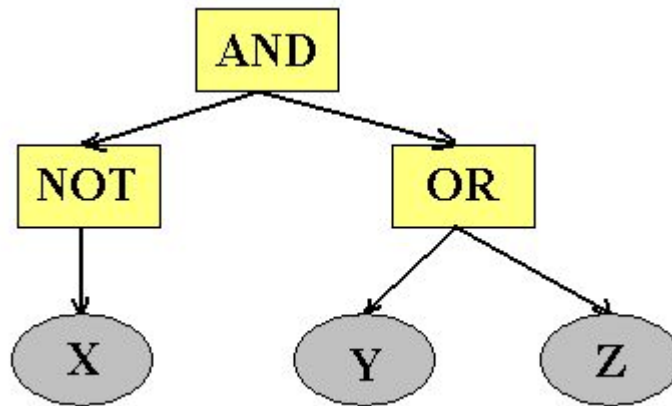
celočíselné, reálné, logické ... konstanty

funkce bez argumentů mající vedlejší efekt.

Příklad:

množina funkcí (NOT, AND, OR)

množina terminálů (X, Y, Z)



Lze interpretovat jako prefixový výraz (AND (NOT X) (OR Y Z)).

U GP máme tyto základní operace: křížení, selekce. Mutace se používají (na rozdíl od GA) jen zřídka.

Výhodou GP oproti GA je, že nezískáme jen obecný model řešící problematiku, ale i popis jak je problematika řešena (třeba zcela konkrétní vzorec, viz příklad).

Zajímavou zkoušku je zadat vstupy a výstupy nějakého vzorce a pak sledovat jak GP hledá správnou toho vzorce.

Problém: najít ekvivalent vzorce $\cos 2X$, pomocí funkce \sin .

Řešení: $\sin(-(-2 * X^2))(\sin(\sin(\sin(\sin(\sin(\sin(\sin(*(\sin(\sin 1))(\sin 1))))))))))$

Zapsáno v **infixovém** **tvary:**
 $\sin(2 - 2x - \sin(\sin(\sin(\sin(\sin(\sin(\sin 1) \cdot \sin 1))))))$

Program si dokázal pro výpočet „zhotovit“ přibližnou hodnotu Ludolfa čísla rekurzivním použitím funkce sinus. Je uváděno, že numerická analýza pro pochopení, proč vztah dává správné výsledky, byla netriviální.

Pokud se budeme na strom dívat jako na program, potom nás nemůže nenapadnou velmi vhodný kandidát na programovací jazyk - Lisp, jehož každý program se dá napsat v reprezentaci stromu.

Jako křížení lze použít výměny podstromů. A i když se mutace v GP typicky nepoužívá lze mít i mutaci v podobě výměny náhodného podstromu nově vytvořeným.

Velkým problémem u takto tvořených stromů je jejich nárůst do obrovských rozměrů. Jedním řešením je zahrnout parametr velikosti stromu o fitness v nepřímé úměrnosti. Dalším šikovným řešením je znovupoužití částí kódu, kód tak může nést mnoho informací i přes jeho relativně malou velikost. Je to obdoba cyklů ve strukturovaném programování.

V GP hledáme většinou s malým počtem parametrů, pak máme rozumnou šanci, že se námi vytvořené řešení uplatní i na netrénované množině úloh.

9. Výhody, nevýhody GA

9.1 Silné, Slabé algoritmy

Co jsou to silné a slabé algoritmy. Hlavním rozdílem jsou požadavky na znalost řešené problematiky a tím pádem i prohledávacího prostoru. Silné algoritmy požadují větší znalost, například Dijkstrův algoritmus, data musí být reprezentována grafem, hledá pouze nejkratší cestu a je k tomu uzpůsoben na úkor obecnosti. Slabý algoritmus je například hill-climbing (což je hledání extrémů tak, že jdu směrem největšího růstu) tento algoritmus nemá žádné bližší nároky. Naprosto čistým zástupcem slabých algoritmů je i GA.

Výhody

Slabý algoritmus pro svou obecnost je použitelný pro velkou skupinu problémů, ale platí se za to pomalejším hledání nejlepšího řešení. Zatímco silné jsou díky své specializovanosti rychlé, ale mají úzký záběr použití.

9.2 (Ne)vhodné úlohy

PROMĚNLIVOST

GA jsou ze své podstaty (díky inspiraci) vhodné na problémy, které se v čase mění. Genetickým algoritmům je vlastní přizpůsobování. Ony se i při statické úloze přizpůsobují ideálnímu řešení.

"TVAR" PROSTORU

Nevhodné pro GA jsou prostory, které charakterizuje velká spousta lokálních maxim oddělených od sebe „hlubokým údolím“. Lze si to představit ve 2D jako velmi rozkmitanou křivku. V takových prostorech nalezení rozumného řešení je spíše náhodou než výsledkem „práce“ GA.

DATA ZATÍŽENÁ CHYBOU

Další skupinou jsou problémy, jejichž data jsou zatížena chybou, ty dokáže GA částečně "filtrovat" díky svému přizpůsobení.

JEDNODUCHOST

Pro jednoduché problémy je řešení pomocí GA příliš robustní, pro tyto problémy jsou známé jiné, mnohem rychlejší (často silné) algoritmy. Na tyto problémy se nevyplatí GA použít.

Naopak velmi vhodný přístup pomocí GA je k problémům, které nemají žádné "rozumně" rychlé řešení. Tam, kde známe klasické řešení dané problematiky a toto řešení je pro nás únosné co do složitosti (polynomiální), bychom neměli používat GA, ale dát přednost „klasice“.

Tato vlastnost by se dala ještě zobecnit. GA jsou ideální pro ty úlohy, které mají složitý, typicky obrovský, těžko popsateľný prohledávací prostor řešení.

Například to jsou problematiky, kde správné řešení neumíme vypočítat a to z důvodů:

- a) neznáme správné vztahy (např. vzorce)
- b) známe postup řešení, ale nejme jej schopni realizovat.

Např. vezmeme si případ b) a bude hledat optimální variantu posloupnosti nějakých objektů, které budou řešením určitého problému. Postup je znám. Provedeme všechny možnosti uspořádání a vybereme optimální. Tento postup má však jeden háček a to, že počet uspořádání je faktoriál počtu objektů.

Počet objektů	Počet posloupností
1	1
2	2
3	6
4	24
10	3628800
20	2.4 E+18
30	2.7 E+32
40	8.2 E+47
50	3.0 E+64
60	8.3 E+81

Již pouhá posloupnost několika desítek objektů dává tolik možností jak objekty uspořádat, že je zcela zbytečné zabývat se realizací dané problematiky tímto postupem. Ač byly GA konstruovány hlavně pro adaptivní úlohy, používají se převážně jinde, velkou skupinu použití jsou NP-úplné problémy, kde GA slaví velké úspěchy. Viz. speciální operátor pro křížení na úlohu obchodního cestujícího.

GA řeší i úlohy ve vícerozměrných prostorech a řeší se velmi elegantně tak, že každý rozměr zakóduje samostatně a pak je spojíme do jednoho chromozómu. Tím získáváme možnost prohledávání n-rozměrných prostorů. Experimenty prokázaly, že při křížení jedinců není nutno zohledňovat místa křížících bodů a kódování rozměrů (křížící bod může být na libovolném místě chromozómu).

9.3 Nebezpečí GA

Pokud dáme GA dostatečně mnoho času, tak si můžeme být jisti, že za pomoci tajného elitářství dostaneme vždy maximum, bohužel toto maximum nemusí být globální. A uvíznutí na lokálním, neglobálním maximu je největší nebezpečí GA. V zásadě platí pravidlo, že předčasná konvergence škodí. Může se totiž stát, že se celá populace usídli v nějakém velkém lokálním maximu (to pro nás znamená zúžení prohledávacího prostoru), protože se tam inicializovala a selekční tlak a křížení nám nedovolí překročit někdy velká údolí. Jak zajistit,

aby GA i ve velmi strmém terénu nekonvergoval brzo? Jedno řešení je, aby mutace byla větší a dovozovala nám objevovat nové prostory, jenže spoléhat na slepou mutaci je příliš jednoduché. Další lepší možnost je zploštit terén, aby bylo snazší překlenout údolí, které třeba může vést k globálnímu maximu.

Zploštění terénu lze implementovat zlogaritmováním fitness funkce, pak se extrémny trochu vyrovnají a z prosté funkce logaritmu se nám nepomíchá uspořádání jedinců. Pak jsou zachovány lokální maxima a jsou lépe přístupné, pokud se nám populace usídí na potenciálním nejlepším řešení, můžeme provést operaci opačnou a zextrémníme prostor, tím se stanou vrcholky strmější, a my dosáhneme daleko přesnějších výsledků.

Jakou výhodu má tajné elitářství oproti normálnímu? Pokud zachováváme několik nejlepších jedinců přímo v populaci, může se stát, že při pokusu uniknout z lokálního maxima do globálního nás budou tyto nejlepší dosud objevení jedinci tahat zpátky do klamného maxima. Tajné elitářství nám zajistí zachování nejlepšího a zároveň zachová přirozený průběh GA.

9.4 Ukončovací podmínka

Kdy vlastně máme GA ukončit? čím déle, tím lépe, pokud používáme elitářství, nejlepší jedinec se nám nemůže ztratit a GA se může jen zlepšit. Ukončovací podmínka pak může být čistě časová záležitost.

Další možnost ukončení je taková, že v průběhu mnoha generací se nám nejlepší jedinec vůbec nezlepšil a tak vše nasvědčuje tomu, že GA našel nejlepší řešení.

Pokud například známe odhad řešení, což je výjimečná situace, můžeme jej částečně do ukončovací podmínky zabudovat, kupříkladu nám stačí 99% jedinec z optimálního řešení.