



MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

Přírodou inspirované prohledávací algoritmy

Roman Neruda, Martin Pilát

22. ZÁŘÍ 2017

Úvod

Prohledávání prostoru řešení parametricky zadaných úloh je jedním z hlavních problémů mnoha oblastí informatiky i matematiky. Aplikace prohledávání jsou nesčetné a zahrnují optimalizaci reálných parametrů, kombinatorickou optimalizaci, automatický vývoj programů, učení robotů nebo návrh architektury a učení neuronových sítí. Přírodou inspirované algoritmy přinesly do tohoto oboru několik nových přístupů, které jsou schopny efektivně pracovat i ve vysoce dimenzionálních prostorech, optimalizovat nelineární funkce s lokálními extrémy a pracovat v tak strukturovaných prostorech jako jsou syntaktické stromy, grafy nebo neuronové sítě.

Výhodou většiny přístupů zmíněných v tomto článku je jejich obecnost. Evoluční i rojové techniky můžeme chápat jako metaheuristiky, které nevyžadují mnoho specifických informací o řešené úloze, kromě ohodnocení jednotlivých kandidátů řešení prostřednictvím účelové funkce nazývané též ohodnocovací funkce nebo fitness. Výhodou tohoto přístupu je snadná aplikovatelnost generického algoritmu na široké spektrum úloh. Na druhou stranu, existují teoretické i praktické důvody pro přizpůsobení obecných prohledávacích heuristik dané úloze. Takto přizpůsobené metody obvykle dosahují lepších výsledků a zároveň si uchovávají obecné výhody těchto technik, jako je odolnost proti uváznutí v lokálních extrémech.

Hlavním rysem zde uváděných přírodou inspirovaných prohledávacích technik je populační přístup. Na rozdíl od většiny algoritmů lokálního prohledávání, které zkoumají nejbližší okolí jednoho bodu v prohledávacím prostoru, evoluční techniky pracují s populací desítek až tisíců řešení, které paralelně prohledávají prostor řešení a navzájem se ovlivňují.

Lokální prohledávací metody šité na míru určitému problému typicky mají výhodu rychlejšího lokálního prohledávání díky využití specifických informací o problému, jako je například informace o gradientu účelové funkce. V dalším textu se zmíníme také o možnosti hybridizovat některé evoluční algoritmy pomocí specifického lokálního prohledávání. Tato technika se v praxi osvědčuje zrychlením konvergence algoritmu, i když někdy přináší větší nebezpečí uváznutí v lokálních extrémech účelové funkce. Je zajímavé, že z hlediska původní biologické inspirace představuje tato hybridizace překročení darwinistického rámce a přináší lamarckistické či epigenetické principy.

<!-- update konec uvodu

V následujícím textu nejprve stručně zmíníme inspiraci a obecné rysy evolučních algoritmů a pak se budeme věnovat několika konkrétním oblastem vycházejícím z těchto obecných principů. Nejprve popíšeme tradiční

genetické algoritmy pracující původně nad binárně zakódovanými jedinci. Dále budeme hovořit o evolučním programování, oblasti která stírá rozdíl mezi genotypem (zakódováním jedince pro účely evolučního prohledávání) a fenotypem (vlastním modelem, který vznikne dekodováním genotypu). Evoluční strategie byly prvním přístupem specializujícím se na optimalizaci reálných parametrů a také přinesly první koncept meta-evoluce, optimalizace parametrů evoluce pomocí vlastního evolučního algoritmu. Genetické programování je příkladem úspěšné evoluce složitých struktur syntaktických stromů počítačových programů. Neuroevoluce ukazuje možnosti využití evolučních technik pro určení struktury i parametrů modelů neuronových sítí. Oblast rojových algoritmů se inspiroje chováním společenského hmyzu a hejn pro efektivní prohledávání různých prostorů, např. reálných euklidovských prostorů nebo hledání cest v grafech.

Evoluční algoritmy

Evoluce, geny a DNA

Obecné schéma evolučního algoritmu

Oblast evolučních výpočtů či algoritmů (v angličtině evolutionary computing) zastřešuje několik proudů, které se zpočátku vyvíjely samostatně. Za prehistorii této disciplíny lze považovat Turingovy návrhy na využití evolučního prohledávání z roku 1948 ¹ a Bremermannovy první pokusy o implementaci optimalizace pomocí evoluce a rekombinace z roku 1962 ². Během šedesátých let se objevily tři skupiny výzkumníků, kteří nezávisle na sobě vyvíjely a navrhly své varianty použití evolučních principů v informatice. Holland publikoval v roce 1975 svůj návrh genetických algoritmů ³, zatímco skupina Fogela a spolupracovníků vyvinula metodu nazvanou evoluční programování ⁴. Nezávisle na nich přišli Rechenberg a Schwefel v Německu na metodu nazvanou evoluční strategie ⁵. Až do přelomu osmdesátých a devadesátých let existovaly tyto směry bez výraznější interakce, ale poté se spojily do obecnější oblasti evolučních algoritmů. V té době Koza vytváří metodu genetického programování, Dorigo publikuje disertaci s návrhem mravenčích optimalizačních algoritmů a vznikají první pokusy o aplikaci evoluce na vývoj umělých neuronových sítí.

U zrodu různých variant evolučních algoritmů stála inspirace přírodními jevy, konkrétně jde o Darwinovu teorii přírodního výběru a zjednodušené principy genetiky, které poprvé načrtl Mendel. Z genetiky se evoluční algoritmy inspirovaly diskretní reprezentací genotypu, z biologické evoluční teorie používají Darwinovu myšlenku o výběru jedinců v prostředí s omezenými zdroji, který závisí na míře přizpůsobení se jedinců danému prostředí.

Základní obecnou myšlenku evolučních algoritmů lze vyjádřit následujícím způsobem. Mějme populaci jedinců v prostředí, které určuje jejich úspěšnost — fitness. Tito jedinci navzájem soupeří o možnost reprodukce a přežití, která závisí právě na hodnotě fitness. Jde tedy o množinu kandidátů na řešení problému definovaného prostředím. Způsoby reprezentace jedinců, jejich výběru a rekombinace závisí na konkrétním dialektu evolučních algoritmů, které probereme vzápětí.

Základní princip fungování evolučních algoritmů je tedy následující ⁶. Na začátku algoritmu vygenerujeme (nejčastěji náhodně) první iniciální populaci jedinců. Všechny jedince v populaci ohodnotíme ohodnocovací funkcí. Hodnota této funkce určuje šanci výběru jedinců během rodičovské selekce. Vybraní jedinci jsou potom rekombinováni pomocí rekombinačního operátoru, který typicky ze dvou jedinců vytváří jednoho či dva potomky, a

¹

Missing ref.

²

Missing ref.

³ John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992

⁴ David B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, USA, 1995

⁵ Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies – a comprehensive introduction. 1(1):3–52, May 2002

⁶ Agoston E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, 2003

procedure EVOLUČNÍ ALGORITMUS

 $t \leftarrow 0$ *Inicializuj* populaci P_t náhodně vygenerovanými jedinci*Ohodnoť* jedince v populaci P_t **while** neplatí *kritérium ukončení* **do** vyber z P_t rodiče *Rodičovskou selekcí* *Rekombinací* rodičů vzniknou potomci *Mutuj* potomky *Ohodnoť* potomky *Enviromentální selekcí* vyber P_{t+1} z P_t a potomků $t \leftarrow t + 1$ **end while****end procedure**

Algoritmus 1: Schéma evolučního algoritmu

pomoci operátoru mutace, který typicky provádí drobné změny jednoho jedince. Tímto postupem si vytvoříme množinu nových kandidátů řešení, a tito noví jedinci potom soutěží s původními jedinci o místo v nové populaci. Výběr jedinců do nové populace (tedy jakési slítí rodičů a potomků) má na starosti enviromentální selekce beroucí v úvahu fitness jedinců a případně další ukazatele jako je například stáří jedinců. Tím je vytvořena nová generace a tento cyklus pokračuje do splnění určitého kritéria ukončení, což je nejčastěji dostatečně dobrý nejlepší jedinec nebo předem určený počet generací.

Genetické algoritmy

Genetické algoritmy

Genetické algoritmy jsou asi nejznámější součástí evolučních výpočtů a v různých obměnách se používají hlavně při řešení optimalizačních úloh. Je zajímavé, že původní Hollandovou motivací při návrhu genetického algoritmu bylo studovat vlastnosti přírodou inspirované adaptace⁷. Velká část původní literatury byla věnována popisu principů, jak genetický algoritmus pracuje při hledání řešení úlohy. Zajímavé jsou paralely s matematickým problémem dvourukého bandity, který je příkladem na udržování optimální rovnováhy mezi explorační a exploatací.

Nejjednodušší varianta genetického algoritmu pracuje s binárními jedinci, to znamená, že parametry řešení úlohy je nutno vždy zakódovat jako binární řetězce. Tento přístup je výhodný z hlediska jednoduchosti použitých operátorů, ale binární zakódování na druhou stranu nemusí být nejvhodnější reprezentací problému. Způsob fungování jednoduchého genetického algoritmu je také poměrně jednoduchý. Algoritmus přechází mezi populacemi řešení tak, že nová populace zcela nahradí předchozí. Výběr rodičů je často realizován tzv. ruletovou selekcí, která vybírá jedince náhodně s pravděpodobností výběru úměrné jejich fitness. Rekombinačním operátorem je jednobodové křížení, které náhodně zvolí stejnou pozici v rodičích a vymění jejich části. Pravděpodobnost uskutečnění operace křížení je jedním z parametrů programu a obvykle je poměrně vysoká (0,5 i více). Mutace provádí drobné lokální změny tak, že prochází jednotlivé bity řetězce a každý bit s velmi malou pravděpodobností změní. Pravděpodobnost mutace je typicky nastavena, tak aby došlo průměrně ke změně jednoho bitu v populaci (oblíbená dolní mez) nebo v jedinci (horní mez).

Ruletovou selekcí si dle metafory můžeme představit tak, že kolo rulety rozdělíme na výseče odpovídající velikostí hodnotám fitness jedinců a při výběru pak n krát vhodíme kuličku. Často používaným vylepšením ruletové selekce je tzv. stochastický univerzální výběr, který hodí kuličku do rulety jen jednou a další jedince vybírá deterministicky posunem pozice kuličky o $1/n$. Tento výběr pro malá n lépe aproximuje ideální počty zastoupení jedinců v další generaci. Dalšími varianty rodičovské selekce nepracují s absolutními hodnotami fitness, ale vybírají náhodně v závislosti na pořadí jedince v populaci setříděné podle fitness, což zanedbává absolutní rozdíly mezi hodnotami. Další variantou rodičovské selekce je tzv. k -turnaj, kdy nejprve vybereme k jedinců náhodně a z nich pak vybereme nejlepšího.

⁷ John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992

procedure JEDNODUCHÝ GENETICKÝ ALGORITMUS

 $t \leftarrow 0$ *Inicializuj* populaci P_t N náhodně vygenerovanými binárními jedinci délky n *Ohodnoť* jedince v populaci P_t **while** neplatí *kritérium ukončení* **do****for** $i \leftarrow 1, \dots, N/2$ **do**vyber z P_t 2 rodiče *Ruletovou selekcí*S pravděpodobností p_C *Zkřiž* rodičeS pravděpodobností p_M *Mutuj* potomky*Ohodnoť* potomkyPřidej potomky do P_{t+1} **end for**Zahoď P_t $t \leftarrow t + 1$ **end while****end procedure**

Algoritmus 2: Schéma Hollandova genetikého algoritmu

*Operátory**GA na číslech**GA na permutacích*

V současnosti se oblast genetických algoritmů neomezuje jen na binární kódování jedinců, časté je celočíselné, permutační nebo reálné kódování, která ale vyžadují specifické operace křížení a mutace⁸. O některých se zmíníme dále.

⁸ Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)*. Springer-Verlag, London, UK, UK, 1996; and Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1996

Evoluční programování

Evoluční programování je oblast, která je na rozdíl od genetických algoritmů charakterizována velmi benevolentním přístupem ke kódování jedinců. První varianty tohoto přístupu vznikly jako metoda pro automatickou evoluci konečných automatů⁹. Jedincem v populaci je v tomto případě tedy konečný automat (zakódovaný jakkoliv) a variační operace se omezují na několik různých mutací. Pro evoluční programování je charakteristické, že mutace jsou přizpůsobené problému a je jich často více, zatímco křížení se vůbec nepoužívá.

EP na KA

EP na číslech

V dalším textu popíšeme variantu evolučního programování, která pracuje s jedinci reprezentovanými vektory reálných čísel. Rodičovská selekce je jednoduchá, každý rodič je jednou vybrán a mutací dá vznik jednomu potomkovi. Enviromentální selekce potom z množiny rodičů a potomků vybere turnajovou selekcí polovinu, která se stane další generací.

procedure META-EP

$t \leftarrow 0$

Inicializuj populaci P_t N náhodně vygenerovanými reálnými vektory

$\vec{x}^t = (x_1^t, \dots, x_n^t, \sigma_1^t, \dots, \sigma_n^t)$

Ohodnoť jedince v populaci P_t

while neplatí kritérium ukončení **do**

for $i \leftarrow 1, \dots, N$ **do**

 k rodiči \vec{x}_i^t vygeneruj potomka \vec{y}_i^t mutací:

for $j \leftarrow 1, \dots, n$ **do**

$\sigma_j' \leftarrow \sigma_j \cdot (1 + \alpha \cdot N(0, 1))$

$x_j' \leftarrow x_j + \sigma_j' \cdot N(0, 1)$

end for

 vlož \vec{y}_i^t do kandidátské populace potomků P_t'

end for

 Turnajovou selekcí vyber P_{t+1} z rodičů P_t a potomků P_t'

 Zahoď P_t a P_t'

$t \leftarrow t + 1$

end while

end procedure

Algoritmus 3: Schéma meta-evolučního programování nad vektorem reálných čísel

Mutace v našem případě pracuje tak, že se snaží o malou změnu reálné

hodnoty, kterou realizuje výběrem z normálního rozdělení. Ukázalo se, že hodnoty rozptylu normálního rozdělení jednotlivých parametrů jsou podstatné pro dobré fungování algoritmu. Jedno z navržených řešení je začlenit tyto rozptyly do každého jedince a upravovat je také v průběhu algoritmu. Jelikož tak vlastně upravujeme parametry, které zároveň používáme při vlastní mutaci, hovoříme o meta-evoluci.

Význam křížení pro EA

Spojité optimalizace

Mnoho optimalizačních problémů z běžného života se dá definovat jako optimalizace funkce

$$f : D \rightarrow \mathbb{R},$$

kde $D \subseteq \mathbb{R}^d$. Je proto přirozené, že mnoho výzkumníků se zabývá právě evolučními algoritmy, kterou jsou schopné optimalizovat takové funkce. Problém optimalizace takových funkcí se v literatuře objevuje pod pojmem *spojitá optimalizace*¹⁰. Je důležité si uvědomit, že název vyjadřuje pouze to, že prostor, ve kterém se hledají řešení je spojitý (\mathbb{R}^n), samotná optimalizovaná funkce f být spojitá nemusí.

¹⁰ anglicky *continuous optimization*

Velmi často je definiční obor funkce D d -rozměrný interval

$$D = [l_1, u_1] \times \cdots \times [l_d, u_d],$$

kde l_i a u_i jsou dolní a horní meze pro i -tou proměnnou. Objevují se ale i obecnější problémy, kde je D dána pomocí množiny podmínek tvaru $g(\vec{x}) \leq 0$ a $h(\vec{x}) = 0$, pro $g, h : \mathbb{R}^d \rightarrow \mathbb{R}$.

V této kapitole napřed budeme uvažovat optimalizační problém tvaru

$$\begin{array}{ll} \min_{\vec{x}} & f(\vec{x}) \\ \text{za podmíněk} & \vec{x} \in [l_1, u_1] \times \cdots \times [l_d, u_d]. \end{array}$$

Vlastnosti funkcí

Je zřejmé, že některé typy funkcí budou pro evoluční algoritmy lehčí, než jiné. Velký vliv na efektivitu evolučního algoritmu mají především vlastnosti jako multi-modalita, separabilita a podmíněnost.

Funkce je *multi-modální*, pokud má velké množství lokálních optim. Je zřejmé, že v takovém případě může mít algoritmus problém s uváznutím v lokálním optimu a je potřeba tomu přizpůsobit operátory. Existuje i oblast multi-modální optimalizace, kde je cílem najít co nejvíce různorodých lokálních optim.

Separabilní funkce jsou naopak pro optimalizaci jednodušší. Jsou to takové funkce, které se dají zapsat pomocí funkcí jedné proměnné. Formálně, funkce $f(x_1, \dots, x_n) : \mathbb{R}^n \rightarrow \mathbb{R}$ je *aditivně separabilní*, pokud se dá zapsat jako součet funkcí $f_1(x_1), \dots, f_n(x_n)$, tj. $f(x_1, \dots, x_n) = \sum_{i=1}^n f_i(x_i)$. Obdobně můžeme zadefinovat i funkci multiplikativně separabilní. Z hlediska optimalizace je velkou výhodou separabilních funkcí, že se dají optimalizovat po jednotlivých složkách vektoru, tj. optimum můžeme najít tak, že vždy zafixujeme hodnoty $n - 1$ parametrů a optimalizujeme jen podle jednoho.

Další vlastností, které výrazně ovlivňuje evoluci je podmíněnost funkce. Ta vyjadřuje, jak moc se liší vliv jednotlivých proměnných na hodnotu funkce. Pro kvadratické funkce (u kterých vrstevnice vypadají jako elipsoidy), je jejich podmíněnost druhá odmocnina poměru mezi délkou nejdelší a nejkratší osy elipsoidu. Pokud je podmíněnost funkce velká, říká se, že je funkce špatně podmíněná. Z hlediska evolučního algoritmu je důležité, že by s různým vlivem proměnných na hodnotu funkce měly počítat operátory.

Příklady posledních dvou vlastností vidíme na obrázku 1, který ukazuje vrstevnice funkcí dvou proměnných. Horní funkce je jednoduchý dvoudimenzionální paraboloid, který je separabilní a dobře podmíněný. Na prostředním obrázku je paraboloid, který má jednu osu delší než druhou a znázorňuje tedy špatnou podmíněnost¹¹. Poslední funkce potom kombinuje špatnou podmíněnost s neseparabilitou.

Výše uvedené vlastnosti jsou ty, které nejvíce ovlivňují efektivitu evolučních algoritmů při spojitě optimalizaci, nejsou to ale všechny. Některé algoritmy například mohou využívat různé symetrie dané funkce, naopak funkce, které mají své globální optimum jen ve velmi malé oblasti prohledávaného prostoru a jinak jsou konstantní jsou pro evoluci velmi těžké obecně.

Kódování pro spojitou optimalizaci

Jedno z prvních rozhodnutí, které je potřeba udělat při návrhu evolučního algoritmu je výběr kódování jedince. Vzhledem k tomu, že ve spojitě optimalizaci pracujeme s vektory reálných čísel, je otázka výběru kódování relativně snadná a jedinci jsou v naprosté většině případů kódování jako vektor typu float nebo double.

??? Existuje pro tohle ↓ nějaká reference?

Dalo by se uvažovat i o kódování jedince přímo po vzoru Hollandova genetického algoritmu, tj. binárním vektorem, a používat jednoduchá n -bodová křížení a bit-flip mutace, ale taková reprezentace trpí tím, že změna různých bitů v číslech vede k výrazně různým změnám hodnoty (např. změna bitu na konci mantisy vs. změna bitu na začátku exponentu).

Operátory pro spojitou optimalizaci

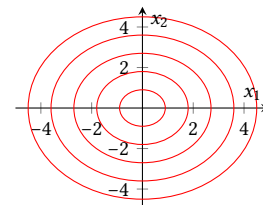
Pro spojitou optimalizaci můžeme samozřejmě použít stejné operátory jako pro každé jiné vektorové kódování jedince, tedy například n -bodové, případně uniformní křížení. Nicméně častěji se používají operátory specializované, které přímo využívají toho, že jedinci jsou vektory čísel.

Tzv. aritmetické křížení vektorů počítá vážený průměr dvou rodičů tak, aby vytvořilo potomka, potomci se tedy spočítají jako

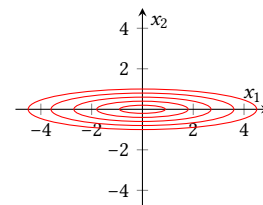
$$o_1 = w * p_1 + (1 - w) * p_2,$$

$$o_2 = (1 - w) * p_1 + w * p_2,$$

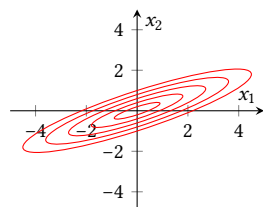
kde p_1 a p_2 jsou rodiče a $w \in (0, 1)$ je (případně náhodně) zvolená váha. Ačkoliv se takové křížení relativně často používá hlavně v jednodušších aplikacích, jeho velkou nevýhodou je, že výslední potomci se nikdy nemohou dostat z konvexního obalu počáteční populace.



(a) Dobře podmíněná separabilní funkce



(b) Špatně podmíněná funkce



(c) Špatně podmíněná a neseparabilní funkce

Obrázek 1: Příklady různých vlastností funkcí

¹¹ podmíněnost zobrazené funkce jen cca 4.8, nedá se tedy považovat za špatně podmíněnou, objevují se i funkce s podmíněností 10^6

Mutace pro spojitou optimalizaci se občas rozdělují na dva typy – ovlivněné a neovlivněné. Neovlivněná mutace generuje novou hodnotu pro složku vektoru nezávisle na aktuální hodnotě, ovlivněná mutace naopak aktuální hodnotu používá. Typickým příkladem neovlivněné mutace je vygenerování nového čísla z daného rozsahu. Ovlivněná mutace typicky přičítá k dané složce vektoru číslo z normálního rozdělení $\mathcal{N}(\mu, \sigma^2)$.

Jednou z nevýhod výše uvedeného aritmetického křížení je, že potomci jsou velmi často relativně daleko od svých rodičů a nejsou jim tedy moc podobní. Právě z toho důvodu Deb a Agrawal navrhli simulované binární křížení (SBX)¹². Hlavní inspirací pro vytvoření SBX bylo jednobodové křížení binárních řetězců, které reprezentují čísla ve dvojkové soustavě. Při náhodné volbě bodu pro křížení velká část nových potomků je relativně blízko k jednomu z rodičů. Jednobodové křížení má navíc další pěknou vlastnost – oba potomci jsou stejně daleko od průměru rodičů.

Cílem SBX je právě simulovat velikosti změn, které se dějí při jednobodovém křížení binárních řetězců. Noví potomci se v SBX křížení spočítají jako

$$o_{1,2} = \frac{1}{2}(p_1 + p_2) \pm \frac{1}{2}\beta(p_2 - p_1),$$

kde p_1 a p_2 jsou rodiče a β je náhodné číslo s pravděpodobnostním rozdělením

$$P(\beta) = \begin{cases} \frac{1}{2}(n+1)\beta^n & \text{pro } \beta \leq 1 \\ \frac{1}{2}(n+1)\frac{1}{\beta^{n+2}} & \text{pro } \beta > 1, \end{cases}$$

kde n je parametr rozdělení, který určuje, jak často budou hodnoty β blízko 1. Pro vyšší hodnoty n je tato pravděpodobnost vyšší (viz Obrázek 2).

Podobnou motivaci jako SBX má i tzv. *polynomiální mutace* (PM)¹³. V tomto případě se simuluje velikost změn v bit-flip mutaci binárně kódovaných čísel. To znamená, že nově vytvořený jedinec je s velkou pravděpodobností blízko svému rodiči. Gaussovská mutace se chová podobně, ale u PM je pravděpodobnost malých změn mnohem větší. Nový jedinec se v polynomiální mutaci vytvoří jako

$$o = p + \delta\Delta_{max}$$

kde Δ_{max} je maximální velikost mutace a δ je náhodné číslo z rozdělení

$$P(\delta) = \frac{1}{2}(n+1)(1-|\delta|)^n, \quad \delta \in (-1, 1).$$

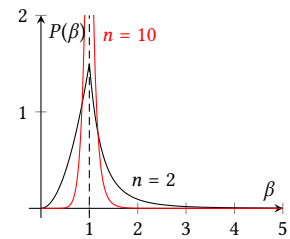
Proměnná n má v tomto případě podobný význam jako u SBX a určuje tvar distribuce. Opět větší hodnota n vede k větší pravděpodobnosti menších změn.

Diferenciální evoluce

Všechny výše popsané operátory mají jednu zásadní nevýhodu a tou je, že operují s jedinci po složkách, jsou tedy vhodné především pro separabilní funkce. *Diferenciální evoluce*¹⁴ přináší jiný přístup, který tímto problémem netrpí.

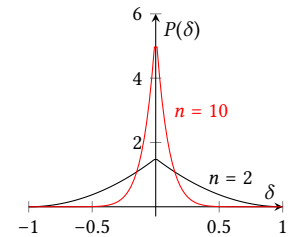
V diferenciální evoluci se jedinec vytváří pomocí jediného operátoru, který v sobě kombinuje jak mutaci, tak křížení. Jeho vstupem jsou hned čtyři jedinci z populace rodičů. Jako rodič p_4 se většinou volí postupně všichni

¹² Kalyanmoy Deb and Ram Bhushan Agrawal. Simulated binary crossover for continuous search space. *Complex systems*, 9(2):115–148, 1995



Obrázek 2: Distribuce β v SBX křížení pro $n = 2$ a $n = 10$.

¹³ Kalyanmoy Deb and Mayank Goyal. A combined genetic adaptive search (GeneAS) for engineering design. *Computer Science and Informatics*, 26:30–45, 1996



Obrázek 3: Pravděpodobnostní rozdělení δ v polynomiální mutaci pro $n = 2$ a $n = 10$.

¹⁴ Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, Dec 1997

jedinci z populace (není tedy volen náhodně), další tři rodiče jsou zvoleny náhodně. Nový jedinec vzniká tak, že se k rodiči p_3 přičte rozdíl rodičů p_2 a p_3 a potom se provede křížení s rodičem p_4 . Postup se dá zapsat jako

$$o^i = \begin{cases} p_3^i + F(p_1^i - p_2^i) & \text{pro } r^i < CR \text{ nebo } i = R \\ p_4^i & \text{jinak,} \end{cases}$$

kde F a CR jsou parametry mutace a křížení, r_i je náhodné číslo z rovnoměrného rozdělení $\mathcal{U}(0, 1)$, R je náhodné číslo z množiny $\{1, \dots, d\}$, které zajišťuje, že v každém jedinci alespoň jedna pozice vznikne pomocí první řádky, a o^i a p_j^i značí i -tou složku potomka resp. j -tého rodiče.

Právě první část definice zajišťuje, že diferenciální evoluce je invariantní vůči rotacím, posunům a škálování prohledávaného prostoru a tedy funguje lépe pro neseparabilní a špatně podmíněné funkce. Pro toto chování je velmi důležitá volba parametru CR . Pro vysoké hodnoty CR větší část jedince vzniká právě pomocí rozdílu dvou rodičů, pro nízké hodnoty CR naopak je většina jedince tvořena rodičem p_4 . Parametr CR se doporučuje nastavovat na nižší hodnoty ($CR = 0.2$) pro separabilní funkce a na vyšší hodnoty pro neseparabilní funkce ($CR = 0.9$).

Parametr F určuje, jak velká část rozdílu dvou jedinců se použije a tedy i to, jak velké se při mutaci dělají kroky. Nejčastěji se doporučuje hodnota okolo $F = 0.8$, ale vyskytují se nastavení mezi 0.5 a 2.0. Někdy se dokonce F bere jako náhodná hodnota z intervalu $[0.5, 1.0]$.

Kromě výše uvedeného operátoru se v diferenciální evoluci liší i selekce. Nový potomek se porovnává s rodičem p_4 a v populaci se nechá jen lepší z obou. To je i důvod, proč se často rodič p_4 nevolí náhodně.

V průběhu let vznikl systém pro klasifikaci různých variant diferenciální evoluce, popsaná verze se dnes nazývá *DE/rand/1/bin*. Označení vyjadřuje, že se jedná o diferenciální evoluci, kde jsou rodič p_3 pro mutaci je volen náhodně, vybírá se jedna dvojice a používá se binomiální křížení. Existuje ale mnoho dalších variant¹⁵. Například se místo náhodně zvoleného jedince p_3 bere vždy nejlepší jedinec z populace (*DE/best/. / .*) nebo se bere více dvojic při mutaci. Pro k dvojic potom vypadá operátor v diferenciální evoluci typu *DE/. / k/. / .* jako

$$o^i = \begin{cases} p_3^i + F \sum_{k=1}^k (p_{1,k}^i - p_{2,k}^i) & \text{pro } r^i < CR \text{ nebo } i = R \\ p_4^i & \text{jinak,} \end{cases}$$

kde značení je stejné jako výše.

Místo binomiálního křížení se občas používá tzv. *exponenciální křížení*. To používá stejný parametr CR , ale jiným způsobem. Jedná se vlastně o obdobu jedno- nebo dvou-bodového křížení. Začne se na náhodné pozici v jedinci a kopírují se parametry z druhého jedince, dokud je náhodně zvolené číslo z $\mathcal{U}(0, 1)$ menší než CR . Pokud se narazí dříve na konec vektoru, pokračuje se znovu od začátku. Varianta s tímto křížením se nazývá *DE/. / . / exp* a ačkoliv je velmi oblíbená, vysloužila si v poslední době kritiku¹⁶ kvůli tomu, že v křížení je větší pravděpodobnost, že se překopírují hodnoty, které jsou na sousedních pozicích, a kvůli tomu je algoritmus závislý na pořadí proměnných.

¹⁵ Efrén Mezura-Montes, Jesús Velázquez-Reyes, and Carlos A. Coello Coello. A comparative study of differential evolution variants for global optimization. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO '06*, pages 485–492, New York, NY, USA, 2006. ACM

¹⁶ Ryoji Tanabe and Alex Fukunaga. Reevaluating exponential crossover in differential evolution. In Thomas Bartz-Beielstein, Jürgen Branke, Bogdan Filipič, and Jim Smith, editors, *Parallel Problem Solving from Nature – PPSN XIII: 13th International Conference, Ljubljana, Slovenia, September 13–17, 2014. Proceedings*, pages 201–210, Cham, 2014. Springer International Publishing

Evoluční strategie

*Evoluční strategie*¹⁷ jsou z historického hlediska jistou alternativou Hollandova genetického algoritmu. Jsou o něco starší a je na nich zajímavé, že jsou o něco komplikovanější. V současnosti se používají především pro řešení problémů spojitě optimalizace, ale myšlenky, které se v této oblasti vyskytují, se dají použít i jinde. V této kapitole představíme základní myšlenky evolučních strategií. Text je inspirován pěkným shrnutím moderních evolučních strategií od Nikolause Hansena *a spol.*¹⁸, které rozhodně doporučujeme těm, kdo se o evolučních strategiích chtějí dozvědět více, případně je zajímají i teoretické výsledky týkající se evolučních strategií.

Evoluční strategie se dělí do dvou skupin podle způsobu, jakým pracují s populacemi rodičů a potomků. V obou případech jsou důležité parametry μ a λ , které označují počet rodičů a počet potomků, kteří z nich vznikají. Pro druhy evolučních strategií potom existuje ustálené značení (μ, λ) -ES a $(\mu + \lambda)$ -ES. V prvním případě (“čárková selekce”) máme populaci μ rodičů, ze kterých vytvoříme λ potomků ($\lambda > \mu$), z těch potom vybereme nejlepších μ jako rodiče do další generace. Ve druhém případě (“plus selekce”) z μ rodičů vytvoříme opět λ potomků, ale před selekcí napřed sloučíme rodiče a potomky do jedné populace velikosti $\mu + \lambda$. Z té se potom opět vybere μ nejlepších jedinců jako rodiče do další generace.

Jednou ze základních vlastností evolučních strategií, které je odlišují od jiných typů evolučních algoritmů je to, že obsahují nějakou formu samo-adaptace parametrů. V případě spojitě optimalizace se tedy kromě samotných hodnot vektoru vyvíjí například i parametry pro mutaci. Technicky se tedy potom jedinec skládá ze dvou částí – samotného zakódovaného vektoru čísel \vec{x} a vektoru tzv. *endogenních parametrů* \vec{s} , které právě obsahují všechny parametry, které ovlivňují chování operátorů¹⁹. Je důležité si uvědomit, že endogenní parametry nijak přímo neovlivňují fitness jedince, jejich hodnoty se vyvíjí jen díky tomu, že jedinci s lepší hodnotou endogenních parametrů mají po aplikaci genetických operátorů častěji lepší fitness. Pro samotnou evoluci endogenních parametrů se mohou používat stejné operátory jako pro samotného jedince s fixně nastavenými parametry, nicméně moderní evoluční strategie častěji používají deterministický způsob nastavení těchto parametrů.

Důležitou součástí evolučních strategií je rekombinace. Ta v zásadě odpovídá křížení v genetických algoritmech a jejím cílem je vytvořit nového jedince kombinací jedinců z populace. V evolučních strategiích se ale velmi často používají rekombinace, které kombinují všechny jedince. Častá je tzv. *interpolační rekombinace*, která jednoduše spočítá průměr všech jedinců v populaci – nový jedinec je vlastně centroid celé populace. Používá se i její

¹⁷ I. Rechenberg. *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Stuttgart, GER, 1973; and Hans-Paul Schwefel. *Numerische optimierung von computer-modellen mittels der evolutionsstrategie*. PhD thesis, 1977

¹⁸ Nikolaus Hansen, Dirk V Arnold, and Anne Auger. Evolution strategies. In *Springer handbook of computational intelligence*, pages 871–898. Springer, 2015

¹⁹ Vedle pojmu “endogenní parametry” se ještě občas objevuje pojem “exogenní parametry”, který označuje parametry algoritmu, které se nemění, jako např. velikost populace.

varianta, která používá vážený součet, kde lepší jedinci v populaci mají větší váhu při výpočtu průměru (nejhorší jedinci se dokonce mohou úplně ignorovat). Kromě těchto dvou nejčastěji používaných rekombinací se dají používat i křížení známá z genetických algoritmů, není to ale moc obvyklé. Existuje i uniformní rekombinace, která každou složku vektoru jedince vybírá z náhodného jedince v populaci.

Použitá rekombinace se občas objevuje i v notaci evolučních strategií, v takovém případě se píše jako $(\lambda/\rho_R + \mu)$, kde právě ρ označuje počet jedinců použitých pro rekombinaci a R označuje konkrétní typ rekombinace (I pro interpolační, W pro váženou interpolační rekombinaci).

Základním operátorem v evolučních strategiích ale je mutace. Typicky jde o gaussovskou mutaci s určitým rozptylem. Zde se objevuje několik možností, kde nejjednodušší je sférická mutace, která přičítá k jedinci vektor z normálního rozdělení $\sigma\mathcal{N}(0, I)$, kde I je jednotková matice. O něco složitější varianta potom používá jiný rozptyl pro každou souřadnici vektoru, tj. k jedinci se přičítá hodnota z normálního rozdělení $\mathcal{N}(0, \text{diag}(\vec{\sigma}))$, kde $\text{diag}(\sigma)$ značí diagonální matici s vektorem $\vec{\sigma}$ na hlavní diagonále. Konečně v nejkomplikovanějším případě se jedinci generují s normálního rozdělení s plnou kovarianční maticí $\mathcal{N}(0, \Sigma)$. Ačkoliv první dva způsoby vyžadují mnohem nižší množství endogenních parametrů (1 respektive d), jejich nevýhoda spočívá v tom, že nejsou schopny zachytit závislosti mezi parametry a hodí se tak především pro separabilní problémy. Na druhou stranu způsob s plnou kovarianční maticí je invariantní vůči rotacím a škálování prohledávaného prostoru (a díky dalším vlastnostem evolučních strategií i k monotónnímu škálování fitness funkce).

Vzhledem k relativně velkému množství endogenních parametrů v evolučních strategiích a k tomu, že vhodné nastavení parametrů je různé pro různé optimalizační problémy a dokonce i v různých fázích evoluce, vznikla potřeba parametry nastavovat adaptivně. Při pevném nastavení velikosti mutace typicky pozorujeme tři fáze evoluce. V první fázi evoluce je konvergence pomalá, protože změny dělané mutací jsou příliš malé. Následuje fáze rychlé konvergence, kde jsou velikosti změn optimální pro danou fázi výpočtu, a v poslední fázi se konvergence zase zpomalí, protože jsou změny moc velké (algoritmus “skáče” kolem optima).

Pravidlo jedné pětiny

První metoda adaptivního nastavování vzešla z Rechenbergových experimentů,²⁰ které zkoumaly vliv rozptylu Gaussovske mutace v $(1 + 1)$ -ES při optimalizaci dvou funkcí. Ukázalo se, že pro obě funkce algoritmus nejrychleji konverguje, pokud se velikost mutace zmenší, když je pravděpodobnost, že je potomek lepší než rodič menší než cca $\frac{1}{5}$, a zvětší, když je tato pravděpodobnost větší než cca $\frac{1}{5}$. Z tohoto pozorování vzniklo pravidlo $\frac{1}{5}$.²¹

Proč takové pravidlo funguje? Představme si optimalizaci lineární funkce, v takovém případě je pravděpodobnost, že potomek je lepší než rodič, přesně 50%. Obecnou funkci můžeme v každém bodě podle Taylorova pravidla aproximovat pomocí lineární funkce, tato aproximace je tím přesnější, čím blíže jsme bodu, ve kterém jsme funkci aproximovali, tedy, při malé velikosti mutace bude většina potomků blízko a pravděpodobnost, že jsou lepší než

²⁰ I. Rechenberg. *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Stuttgart, GER, 1973

²¹ V angličtině “one fifth rule”

rodič, je kolem 50%. Naopak, při nekonečném rozptylu mutace pravděpodobnost zlepšení odpovídá části prostoru, kde má funkce lepší hodnotu, než v daném bodě. Ve chvíli, kdy se algoritmu přiblíží optimu je tato pravděpodobnost většinou blízko 0. Ve skutečnosti se dokonce ukazuje, že pro většinu funkcí pravděpodobnost zlepšení monotónně roste s klesajícím rozptylem mutace. Přesná hodnota pro optimální pravděpodobnost zlepšení závisí na optimalizované funkci, nicméně právě $\frac{1}{2}$ je často doporučovaná.

Implementace pravidla $\frac{1}{2}$ je velmi jednoduchá. Může vypadat například tak, že se rozptyl upravuje podle vztahu

$$\sigma = \sigma \exp^{1/\sqrt{d+1}}(f(\vec{x}_1) - f(\vec{x}) - \frac{1}{2}),$$

kde je charakteristická funkce (tj. $(\varphi) = 1$, pokud φ je pravdivý výraz a 0 jinak), \vec{x}_1 je potomek rodiče \vec{x} a d je počet proměnných daného optimalizačního problému.

Sebe-adaptivní evoluční strategie

Jiný přístup k sebe-adaptaci endogenních parametrů zvolil Schwefel²², ten pro jejich ladění použil podobné genetické operátory, jako se používají pro samotné zakódované řešení. Základní myšlenka je, že se každý endogenní parametr vynásobí číslem $\exp(\mathcal{N}(0, I))$. Hlavním důvodem pro použití takové “multiplikativní” mutace je to, že relativní velikost změn oběma směry je stejná a navíc se vyvíjí vyvíjí rozptyly pro mutaci, které musí být vždy kladné. Aditivní mutace se v takovém případě moc nehodí.

Celkově se dá tato evoluční strategie (nazývaná $(\mu/\mu, \lambda) - \sigma$ SA-ES zapsat jako v algoritmu 4 níže. Můžeme si všimnout hned několika věcí – mutace pro jedince i pro parametry je podobná, až na to, že mutace parametrů je multiplikativní a mutace jedince aditivní. Rekombinace se chová opět stejně pro jedince i pro jim příslušné parametry mutace – počítá se průměr přes všechny jedince v populaci. Díky tomu, že je rekombinace deterministická, stačí ji dělat jen jednou za iteraci, proto se dělá mimo for cyklus.

²² Hans-Paul Schwefel. *Numerische optimierung von computer-modellen mittels der evolutionsstrategie*. PhD thesis, 1977

Require: $d \in \mathbb{N}_+, \lambda \geq 5d, \mu \approx \lambda/4, \tau \approx 1/\sqrt{d}, \tau_i \approx 1/d^{1/4}$

```

1: procedure  $(\mu/\mu, \lambda) - \sigma$ SA-ES
2:   inicializuj  $\vec{x} \in \mathbb{R}^n, \vec{\sigma} \in \mathbb{R}_+^n$ 
3:   while není konec do
4:     for  $k \in 1, \dots, \lambda$  do
5:        $\xi_k = \tau \mathcal{N}(0, 1)$  ▷ globální rozptyl
6:        $\vec{\xi}_k = \tau_i \mathcal{N}(0, I)$  ▷ rozptyl pro souřadnice
7:        $\vec{z}_k = \mathcal{N}(0, I)$  ▷ změna řešení  $\vec{x}$ 
8:        $\vec{\sigma}_k = \sigma \circ \exp(\vec{\xi}_k) \exp(\xi_k)$ 
9:        $\vec{x}_k = \vec{x} + \vec{\sigma}_k \circ \vec{z}_k$ 
10:    end for
11:     $\mathcal{P} =$  vyber  $\mu$  nejlepších z  $\{(\vec{x}_i, \vec{\sigma}_i) \mid i \in \{1, \dots, k\}\}$ 
12:     $\vec{\sigma} = \frac{1}{\mu} \sum_{\vec{\sigma}_k \in \mathcal{P}} \vec{\sigma}_k$ 
13:     $\vec{x} = \frac{1}{\mu} \sum_{\vec{x}_k \in \mathcal{P}} \vec{x}_k$ 
14:  end while
15: end procedure
```

Algoritmus 4: Evoluční strategie s adaptací rozptylů mutace, d značí počet proměnných problému, \circ je operace násobení vektoru po složkách

Mezi Rechenbergovým a Schwefelovým přístupem je rozdíl i v tom, že ten druhý nastavuje různé rozptyly mutace pro různé souřadnice, původní Rechenbergův přístup pracuje jen s jedním rozptylem a tedy se sférickou mutací.

Přestože jsou relativní změny rozptylů nestranné, ukazuje se, že algoritmus má v případě neutrální selekce²³ tendenci velikost rozptylu zvyšovat. Tomu se dá předejít použitím větší velikosti populace. Nicméně moderní evoluční strategie z tohoto důvodu přistupují k deterministickým (někdy se používá termín “derandomizovaným”) metodám nastavení endogenních parametrů.

Kumulativní adaptace velikosti kroku

Jedním ze způsobů odstranění náhody z adaptace evolučních strategií je použití tzv. *evoluční cesty*, je to vlastně vektor, ve kterém se ukládají průměrné změny v každém směru u jedinců, kteří projdou selekcí. Z velikosti těchto změn se potom spočítá nový vektor rozptylů. Právě takovou techniku používá algoritmus založený na kumulativní změně velikosti kroku (Cumulative Step-Size Adaptation – CSA)²⁴ vytvořený Ostermeierem a spol.²⁵ Přesný popis CSA je uveden jako Algoritmus 6.

Základ algoritmu je podobný výše uvedenému algoritmu z adaptací rozptylů, ale vidíme, že adaptace nyní není založena na náhodě, ale počítá se právě z evoluční cesty \vec{s}_σ . Aby se zabránilo přílišným oscilacím evoluční cesty způsobeným náhodným samplováním, vektor se průměruje s vektorem z předcházejícího kroku. Relativně složité koeficienty, které se při výpočtu průměru používají, jsou zvoleny tak, aby se očekávaná velikost kroku neměnila při neutrální selekci.

Require: $d \in \mathbb{N}_+, \lambda \in \mathbb{N}, \mu \approx \lambda/4, c_\sigma \approx \sqrt{\mu/(d + \mu)}, f \approx 1 + \sqrt{\mu/d}, f_i \approx 3d$

- 1: **procedure** $(\mu/\mu, \lambda)$ -ES s EVOLUČNÍ CESTOU
- 2: Inicializuj $\vec{x} \in \mathbb{R}^d, \vec{\sigma} \in \mathbb{R}_+^n$
- 3: **while** není konec **do**
- 4: **for** $k \in \{1, \dots, \lambda\}$ **do**
- 5: $\vec{z}_k = \mathcal{N}(0, \mathbf{I})$
- 6: $\vec{x}_k = \vec{x} + \vec{\sigma} \circ \vec{z}_k$ ▷ mutace
- 7: **end for**
- 8: \mathcal{P} = vyber μ nejlepších jedinců z $\{(\vec{x}_k, \vec{z}_k) | 1 \leq k \leq \lambda\}$
- 9: $\vec{s}_\sigma = (1 - c_\sigma)\vec{s}_\sigma + \sqrt{c_\sigma(2 - c_\sigma)} \frac{\sqrt{\mu}}{\mu} \sum_{\vec{z}_k \in \mathcal{P}} \vec{z}_k$ ▷ evoluční cesta
- 10: $\vec{\sigma} = \vec{\sigma} \exp^{1/f_i} \left(\frac{\|\vec{s}_\sigma\|}{\mathbb{E}[\|\mathcal{N}(0, \mathbf{I})\|]} \right) \exp^{c_\sigma/f} \left(\frac{\|\vec{s}_\sigma\|}{\mathbb{E}[\|\mathcal{N}(0, \mathbf{I})\|]} \right)$
- 11: $\vec{x} = \frac{1}{\mu} \sum_{\vec{x}_k \in \mathcal{P}} \vec{x}_k$ ▷ rekombinace
- 12: **end while**
- 13: **end procedure**

Adaptace kovarianční matice

Všechny výše uvedené algoritmy adaptovaly jen různé rozptyly pro různé složky vektoru, tedy nebyly schopny postihnout závislosti mezi proměnnými, tj. byly vhodné především pro separabilní problémy. V této části

²³ Neutrální selekce je nezávislá na fitness jedince, dá se implementovat např. tak, že je fitness konstantní a nejlepší jedinci se vybírají náhodně. Občas se používá při analýze chování algoritmů.

²⁴ Rozptyl gaussovské mutace se v evolučních strategiích také označuje jako velikost kroku – anglicky “step-size”

²⁵ Andreas Ostermeier, Andreas Gawelczyk, and Nikolaus Hansen. *Step-size adaptation based on non-local use of selection information*, pages 189–198. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994

Algoritmus 5: Evoluční strategie s koordinovanou adaptací velikosti kroku, d značí počet proměnných problému, \circ je operace násobení vektoru po složkách

si představíme evoluční strategii, která adaptuje celou kovarianční matici (CMA-ES)²⁶ a změny v jedincích tedy sampuluje z vícerozměrného normálního rozdělení. Díky tomu se podobně jako diferenciální evoluce stává invariantní vůči rotacím a škálování prohledávaného prostoru a díky tomu, že používá jen porovnávání hodnot fitness, je nezávislá i na monotónních škálováních fitness funkce.

Algoritmus CMA-ES²⁷ je vlastně zobecněním předchozího algoritmu, k evoluční cestě pro σ se přidává evoluční cesta i pro kovarianční matici C . Níže uvedený pseudokód popisuje $(\mu/\mu_w, \lambda)$ -CMA-ES, tj. verzi, kde se používá vážený součet jedinců při rekombinaci (viz řádka 13). Jedná se o jednu z nejobecnějších verzí algoritmu CMA-ES. Ačkoliv vypadá komplikovaně, neliší se moc od předchozího algoritmu. Pořád obsahuje evoluční cestu pro σ (\vec{s}_σ), která určuje globální velikost kroku σ (řádky 9 a 11), navíc ale obsahuje jen evoluční cestu pro kovarianční matici C (\vec{s}_c), která se používá při její aktualizaci. Myšlenka aktualizace je podobná jako u aktualizace σ , tj. počítá se vážený součet předcházející hodnoty C , aktuální evoluční cesty pro C a kovariance aktuálních změn. Cílem opět je vyhnout se prudkým změnám C .

Require: $d \in \mathbb{N}_+, \lambda \geq 5 \in \mathbb{N}, \mu \approx \lambda/2, w_k = w'(k)/\sum_k w'(k), w'(k) = \log(\lambda/2 + 1/2) - \log \text{rank}(f(\vec{x}_k)), \mu_w = 1/\sum_k w_k^2, c_\sigma \approx \mu_w/(d + \mu_w), f \approx 1 + \sqrt{\mu_w/d}, c_c \approx (4 + \mu_w/d)/(d + 4 + 2\mu_w/d), c_1 = 2/(d^2 + \mu_w), c_\mu \approx \mu_w/(d^2 + \mu_w), c_m = 1, h_\sigma = 1_{\|\vec{s}_\sigma\|^2/d < 4/(d+1)}, c_h = c_1(1 - h_\sigma^2)c_c(2 - c_c), C^{1/2}C^{1/2} = C$

- 1: **procedure** $(\mu/\mu_w, \lambda)$ -CMA-ES
- 2: Inicializuj $\vec{x} \in \mathbb{R}^d, \vec{\sigma} \in \mathbb{R}_+^n, C = I, \vec{s}_c = 0, \vec{s}_\sigma = 0$
- 3: **while** není konec **do**
- 4: **for** $k \in \{1, \dots, \lambda\}$ **do**
- 5: $\vec{z}_k = \mathcal{N}(0, I)$
- 6: $\vec{x}_k = \vec{x} + \sigma C^{1/2} \vec{z}_k$ ▷ mutace
- 7: **end for**
- 8: $\mathcal{P} =$ vyber μ nejlepších jedinců z $\{(\vec{x}_k, \vec{z}_k) | 1 \leq k \leq \lambda\}$
- 9: $\vec{s}_\sigma = (1 - c_\sigma)\vec{s}_\sigma + \sqrt{c_\sigma(2 - c_\sigma)}\sqrt{\mu_w} \sum_{\vec{z}_k \in \mathcal{P}} w_k \vec{z}_k$ ▷ evoluční cesta
- 10: $\vec{s}_c = (1 - c_c)\vec{s}_c + h_\sigma \sqrt{c_c(2 - c_c)}\sqrt{\mu_w} \sum_{\vec{z}_k \in \mathcal{P}} w_k C^{1/2} \vec{z}_k$
- 11: $\vec{\sigma} = \vec{\sigma} \exp^{c_\sigma/f} \left(\frac{\|\vec{s}_\sigma\|}{\mathbb{E}\|\mathcal{N}(0, I)\|} - 1 \right)$
- 12: $C = (1 - c_1 + c_h - c_\mu)C + c_1 \vec{s}_c \vec{s}_c^T + c_\mu \sum_{\vec{z}_k \in \mathcal{P}} w_k C^{1/2} \vec{z}_k (C^{1/2} \vec{z}_k)^T$
- 13: $\vec{x} = \vec{x} + c_m \sigma C^{1/2} \sum_{\vec{z}_k \in \mathcal{P}} w_k \vec{z}_k$ ▷ rekombinace
- 14: **end while**
- 15: **end procedure**

Velké množství relativně složitých výrazů v kódu opět zajišťuje nestranost algoritmu při neutrální selekci.

Algoritmus CMA-ES aktuálně patří mezi nejlepší evoluční algoritmy pro spojitou optimalizaci, práce na něm stále pokračují a algoritmus se dále vyvíjí. Hledají se například jiné cesty pro adaptaci velikosti kroku v algoritmu.

²⁶ V angličtině “Covariance Matrix Adaptation Evolution Strategy”

²⁷ Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9(2):159–195, June 2001

Algoritmus 6: Evoluční strategie s koordinovanou adaptací velikosti kroku, d značí počet proměnných problému, \circ je operace násobení vektoru po složkách

Kombinatorická optimalizace

Evoluce pro strojové učení

Rojové algoritmy

Rojové algoritmy mají některé společné rysy s evolučními algoritmy, jako je populační prohledávání, ale jsou charakteristické tím, že kladou důraz na modelování pohybu populace v prostoru parametrů inspirovaném různými druhy organizace sociálního hmyzu nebo jiných živočichů v hejnech. Počáteční výzkum Reynoldse vedl k návrhu jednoduchého lokálního algoritmu tzv. boidů, kteří velice jednoduše a překvapivě dobře modelují organizaci roje a koordinovaný pohyb v něm. Teprve algoritmus optimalizace rojem částic (PSO) ale přinesl využití těchto principů do prohledávacích algoritmů. V současnosti existuje celá řada dalších příbuzných algoritmů od ant colony optimization přes umělé imunitní systémy až ke včelím algoritmům ²⁸.

28

Principem fungování rojové optimalizace je organizovaný pohyb roje částic (jedinců) po prostoru parametrů. Každá částice je charakterizována svou polohou a vektorem rychlosti. Každá částice si také pamatuje ze své historie nejlepší místo (ve smyslu hodnoty účelové funkce), které ona sama v historii navštívila a globální dosud nejlepší místo nalezené celým rojem. V každém kroku pak částice upraví svou rychlost jako součet tří vektorů: současné rychlosti, směru k lokálnímu extrému částice a směru ke globálnímu extrému celého roje. Tato suma je vážena dílem vstupními parametry algoritmu nastavovanými experimentátorem (jde o nelehký problém závisející na typu dat) a dílem náhodně vygenreovanými koeficienty pro každý krok.

procedure OPTIMALIZACE ROJEM ČÁSTIC

 $\vec{g} \leftarrow \arg \max_{\vec{y}} f(\vec{y})$ \triangleright inicializace globální nejlepší pozice**for** $i \leftarrow 1, \dots, l$ **do**Inicializuj pozici částice \vec{x}_i náhodně z rovnoměrného rozdělení $\vec{p}_i \leftarrow \vec{x}_i$ \triangleright inicializace nejlepší známé pozice částice**if** $f(\vec{p}_i) < f(\vec{g})$ **then** $\vec{g} \leftarrow \vec{p}_i$ \triangleright update globální nejlepší pozice**end if**Inicializuj rychlost částice \vec{v}_i náhodně z rovnoměrného rozdělení**end for****while** $(f(\vec{x}^t) > \text{kritérium ukončení})$ **do****for** $i \leftarrow 1, \dots, l$ **do**zvol r_g a r_p z rovnoměrného rozdělení $U(0, 1)$ **for** $d \leftarrow 1, \dots, n$ **do** \triangleright update rychlosti částice $v_{i,d} \leftarrow \omega v_{i,d} + \varphi_p r_p (p_{i,d} - x_{i,d}) + \varphi_g r_g (g_{i,d} - x_{i,d})$ **end for** $\vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i$ \triangleright update pozice částice**if** $f(\vec{x}_i) < f(\vec{p}_i)$ **then** $\vec{p}_i \leftarrow \vec{x}_i$ \triangleright update nejlepší pozice částice**if** $f(\vec{p}_i) < f(\vec{g})$ **then** $\vec{g} \leftarrow \vec{p}_i$ \triangleright update nejlepší globální pozice**end if****end if****end for****end while****end procedure**

Algoritmus 7: Schéma optimalizace rojem částic minimalizujících $f : \mathbb{R}^n \rightarrow \mathbb{R}$

Genetické programování

Genetické programování je zajímavé tím, že přináší práci s jedinci odlišného typu — jde o stromy reprezentující programy²⁹. Strom je sestaven z množiny neterminálních symbolů reprezentujících funkce a množiny terminálních symbolů (listů) reprezentujících proměnné a konstanty. Proměnné jsou charakterizovány úlohou k řešení, neterminální symboly určují programovací jazyk, ve kterém budou programy vznikat. Stromy jsou vhodné jednak proto, že představují kompaktní reprezentaci programů, a také proto, že na nich lze elegantně provádět křížení realizované jako výměna náhodně zvolených podstromů mezi dvěma jedinci. Mutace je realizována jako nahrazení podstromu náhodně vygenerovaným podstromem. Je zajímavé, že v kontrastu s jinými oblastmi evolučních algoritmů, nehraje mutace v genetickém programování tak významnou roli a obvykle mívá malou pravděpodobnost. Je to pravděpodobně způsobeno tím, že křížení představuje ve stromu programu velké změny, které tak supluje i účinek mutace.

Náhodné generování stromů je důležitou operací jak pro inicializaci populace tak pro mutace. Byly navrženy dvě metody, které se snaží buď náhodně generovat kompletní strom dané hloubky, anebo náhodně rostou větve stromů. Druhá metoda generuje tedy řidší stromy s nesterjně dlouhými větvemi. Pro inicializaci se doporučuje kombinace těchto metod v poměru 1:1 (ramped half-and-half).

procedure GENETICKÉ PROGRAMOVÁNÍ

 $t \leftarrow 0$ *Inicializuj* populaci P_t metodou *ramped half-and-half**Ohodnoť* jedince v populaci P_t **while** neplatí *kritérium ukončení* **do** $i \leftarrow 0$ **repeat**vyber pravděpodobnostně variační operaci $o \in \{C, M\}$ **if** $o=C$ **then** ▷ Kříženívyber z P_t 2 rodiče*Zkříž* rodičevlož potomky do P_{t+1} $i \leftarrow i + 2$ **else** ▷ Mutacevyber z P_t 1 rodiče*Mutuj* rodičevlož potomka do P_{t+1} $i \leftarrow i + 1$ **end if****until** $i \geq n$ *Ohodnoť* jedince v populaci P_{t+1} *Zahoď* P_t $t \leftarrow t + 1$ **end while****end procedure**

Algoritmus 8: Schéma Kozova algoritmu genetického programování nad syntaktickými stromy

Neuroevoluce

Použití evolučních algoritmů pro učení neuronových sítí je oblast zkoumaná od přelomu osmdesátých a devadesátých let. První pokusy se týkaly učení vah sítí s předem danou strukturou, jde tedy o využití evolučního algoritmu jako alternativy ke standardnímu učení neuronových sítí založených typicky na lokálních metodách gradientní optimalizace³⁰. Tato úloha není příliš složitá, parametry sítě zakódujeme do reálného vektoru a na něj použijeme jeden z výše popsaných algoritmů. Výpočet fitness pak znamená spočítat chybu neuronové sítě na množině trénovacích dat.

30

Dnes z experimentů víme, že evoluční algoritmus těžko soupeří v rychlosti s nejlepšími gradientními algoritmy, nicméně jeho výhody pro učení parametrů sítě jsou robustnost proti uvážnutí v lokálních extrémech a snadná paralelizace. Jednou z oblastí, kde je evoluční algoritmus pro učení neuronových sítí nezastupitelný, jsou případy tzv. posilovaného učení, kde není k dispozici informace o chybě sítě pro každý vstup. Tyto případy jsou velmi časté v robotice, kdy teprve z chování robota v delším časovém úseku můžeme odvodit jeho úspěšnost. V této oblasti je evoluční učení neuronových sítí takřka nezastupitelné, i proto se často hovoří o oboru evoluční robotiky.

Schopnost evolučních algoritmů optimalizovat i složitěji reprezentované struktury vedla ke snahám použít je pro optimalizaci velikosti a propojení jednotek uvnitř sítě, mluvíme o strukturálním učení. Toto učení lze rozdělit podle toho, zda reprezentace sítě je zakódovaná do jedince evolučního algoritmu přímo či nepřímo. Další dělení je, zda se struktura učí najednou společně s parametry sítě (takže kódujeme obě komponenty v jednom genomu) anebo se struktura učí zvlášť a učení parametrů sítě je vlastně záležitostí výpočtu fitness jedince. Při tomto rozdělení úlohy na učení struktury a parametrů je zajímavé si uvědomit, že učit parametry lze libovolným algoritmem, třeba další evolucí.

Při přímých metodách učení struktury sítě se většinou pracuje s reprezentací pomocí matice propojení neuronů v síti. Tato matice může být booleovská (v případě reprezentace struktury) nebo reálná (pro reprezentaci struktury i hodnot vah spojů v síti). Jelikož matice propojení může být poměrně velká, vznikly snahy o kompaktní vyjádření struktury sítě, které je často nepřímé. Kitano navrhl gramatické kódování booleovské matice spojů, které uvažuje dvojrozměrnou gramatiku schopnou v logaritmickém prostoru popsat matice navíc s možností zachytit symetrie struktury. Gruau navrhl jinou metodu založenou na principu genetického programování, kde program je vlastně návod na sestavení sítě metodou růstu z minimální konfigurace³¹.

31

Jednou z dnes nejúspěšnějších neuroevolučních metod je Stanleyho Neat

(neuroevolution of augmenting topologies)³². Jde o přímou metodu vyvíjející zároveň strukturu i parametry sítě opět postupem od minimální konfigurace ke složitějším. Autor elegantně vyřešil problém, jak má vlastně vypadat křížení dvou sítí s různou topologií. Používá k tomu globální paměť evoluční historie tvaru sítě jednoduše realizovatelnou pomocí tzv. rodného čísla spoje, díky kterému lze určit, které hrany v grafu sítě si evolučně odpovídají, a ty se pak mohou křížit. Důležitým problémem při současném učení struktury a parametrů se ukázal krátkodobý negativní vliv strukturálních změn na kvalitu sítě a z toho vyplývající nutnost ochrany strukturálně nových jedinců, kteří potřebují čas na doladění svých parametrů. To je v Neatu opět řešeno pomocí využití rodných čísel hran pro určení podobných sítí a relativizací fitness uvnitř množin podobných sítí.

Obsah

| | |
|------|---|
| Úvod | 3 |
|------|---|

| | |
|---|---|
| <i>Evoluční algoritmy</i> | 5 |
| <i>Evoluce, geny a DNA</i> | 5 |
| <i>Obecné schéma evolučního algoritmu</i> | 5 |

| | |
|----------------------------|---|
| <i>Genetické algoritmy</i> | 7 |
| <i>Genetické algoritmy</i> | 7 |
| <i>Operátory</i> | 8 |
| <i>GA na číslech</i> | 8 |
| <i>GA na permutacích</i> | 8 |

| | |
|------------------------------|----|
| <i>Evoluční programování</i> | 9 |
| <i>EP na KA</i> | 9 |
| <i>EP na číslech</i> | 9 |
| <i>Význam křížení pro EA</i> | 10 |

| | |
|--|----|
| <i>Spojitá optimalizace</i> | 11 |
| <i>Vlastnosti funkcí</i> | 11 |
| <i>Kódování pro spojitou optimalizaci</i> | 12 |
| <i>Operátory pro spojitou optimalizaci</i> | 12 |
| <i>Diferenciální evoluce</i> | 13 |

| | |
|------------------------------|----|
| <i>Evoluční strategie</i> | 15 |
| <i>Pravidlo jedné pětiny</i> | 16 |

| | |
|---|--------|
| <i>Sebe-adaptivní evoluční strategie</i> | 17 |
| <i>Kumulativní adaptace velikosti kroku</i> | 18 |
| <i>Adaptace kovarianční matice</i> | 18 |
| <i>Kombinatorická optimalizace</i> | 21 |
| <i>Evoluce pro strojové učení</i> | 23 |
| <i>Rojové algoritmy</i> | 25 |
| <i>Genetické programování</i> | 27 |
| <i>Neuroevoluce</i> | 29 |
| <i>Rejstřík</i> | 39 |
| <i>Literatura</i> | 43 |

Seznam obrázků

- 1 Příklady různých vlastností funkcí 12
- 2 Distribuce β v SBX křížení pro $n = 2$ a $n = 10$. 13
- 3 Pravděpodobnostní rozdělení δ v polynomiální mutaci pro $n = 2$ a
 $n = 10$. 13

Seznam tabulek

Seznam algoritmů

| | | |
|---|---|----|
| 1 | Schéma evolučního algoritmu | 6 |
| 2 | Schéma Hollandova genetického algoritmu | 8 |
| 3 | Schéma meta-evolučního programování nad vektorem reálných čísel | 9 |
| 4 | Evoluční strategie s adaptací rozptylů mutace, d značí počet proměnných problému, \circ je operace násobení vektoru po složkách | 17 |
| 5 | Evoluční strategie s koordinovanou adaptací velikosti kroku, d značí počet proměnných problému, \circ je operace násobení vektoru po složkách | 18 |
| 6 | Evoluční strategie s koordinovanou adaptací velikosti kroku, d značí počet proměnných problému, \circ je operace násobení vektoru po složkách | 19 |
| 7 | Schéma optimalizace rojem částic minimalizujících $f : \mathbb{R}^n \rightarrow \mathbb{R}$ | 26 |
| 8 | Schéma Kozova algoritmu genetického programování nad syntaktickými stromy | 28 |





Rejstřík

Evoluční algoritmus, 5
Evoluční programování, 5

Evoluční strategie, 5

Genetický algoritmus, 5, 7

Todo list

| | | |
|---|--|----|
|  | <!-- update konec uvodu | 3 |
|  | Missing ref. | 5 |
|  | Missing ref. | 5 |
|  | ??? Existuje pro tohle ↓ nějaká reference? | 12 |

Literatura

- [1] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies – a comprehensive introduction. 1(1):3–52, May 2002.
- [2] Kalyanmoy Deb and Ram Bhushan Agrawal. Simulated binary crossover for continuous search space. *Complex systems*, 9(2):115–148, 1995.
- [3] Kalyanmoy Deb and Mayank Goyal. A combined genetic adaptive search (GeneAS) for engineering design. *Computer Science and Informatics*, 26:30–45, 1996.
- [4] Agoston E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, 2003.
- [5] David B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, USA, 1995.
- [6] Frédéric Gruau. *Neural Network Synthesis Using Cellular Encoding And The Genetic Algorithm*. PhD thesis, L’universite Claude Bernard-lyon I, 1994.
- [7] Nikolaus Hansen, Dirk V Arnold, and Anne Auger. Evolution strategies. In *Springer handbook of computational intelligence*, pages 871–898. Springer, 2015.
- [8] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9(2):159–195, June 2001.
- [9] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.
- [10] James Kennedy, James F Kennedy, Russell C Eberhart, and Yuhui Shi. *Swarm intelligence*. Morgan Kaufmann, 2001.
- [11] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
- [12] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA, USA, 1994.
- [13] John R. Koza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

- [14] John R. Koza, David Andre, Forrest H. Bennett, and Martin A. Keane. *Genetic Programming III: Darwinian Invention & Problem Solving*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1999.
- [15] Efrén Mezura-Montes, Jesús Velázquez-Reyes, and Carlos A. Coello Coello. A comparative study of differential evolution variants for global optimization. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO '06*, pages 485–492, New York, NY, USA, 2006. ACM.
- [16] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)*. Springer-Verlag, London, UK, UK, 1996.
- [17] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1996.
- [18] Andreas Ostermeier, Andreas Gawelczyk, and Nikolaus Hansen. *Step-size adaptation based on non-local use of selection information*, pages 189–198. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.
- [19] I. Rechenberg. *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Stuttgart, GER, 1973.
- [20] Hans-Paul Schwefel. *Numerische optimierung von computer-modellen mittels der evolutionsstrategie*. PhD thesis, 1977.
- [21] Gene I. Sher. *Handbook of Neuroevolution Through Erlang*. Springer Publishing Company, Incorporated, 2012.
- [22] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.
- [23] Kenneth O Stanley and Risto Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, 2003.
- [24] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, Dec 1997.
- [25] Ryoji Tanabe and Alex Fukunaga. Reevaluating exponential crossover in differential evolution. In Thomas Bartz-Beielstein, Jürgen Branke, Bogdan Filipič, and Jim Smith, editors, *Parallel Problem Solving from Nature – PPSN XIII: 13th International Conference, Ljubljana, Slovenia, September 13–17, 2014. Proceedings*, pages 201–210, Cham, 2014. Springer International Publishing.