



MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

Přírodou inspirované prohledávací algoritmy

Roman Neruda, Martin Pilát

15. LEDNA 2018

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 slití 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 *Zkříž* 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

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

Ohodnot' jedince v populaci P_t

while neplatí kritérium ukončení **do**

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

 k rodiči \vec{x}_i^t vygenruj 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

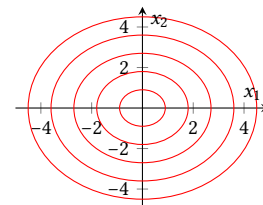
Pro spojitou optimalizaci můžeme samozřejmě použít stejné operátory jako pro každé jiné vektorové kódování jedince, tedy například n -bodové, případně uniformní křížení. Nicméně častěji se používají operátory specializované, které přímo využívají toho, že jedinci jsou vektory čísel.

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

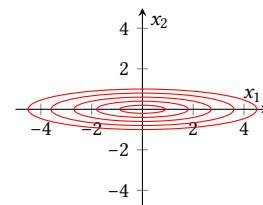
$$o_1 = w * p_1 + (1 - w) * p_2,$$

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

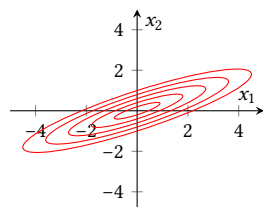
kde p_1 a p_2 jsou rodiče a $w \in (0, 1)$ je (případně náhodně) zvolená váha. Ačkoliv se takové křížení relativně často používá hlavně v jednodušších aplikacích, jeho velkou nevýhodou je, že výslední potomci se nikdy nemohou dostat z konvexního obalu počáteční populace.



(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

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

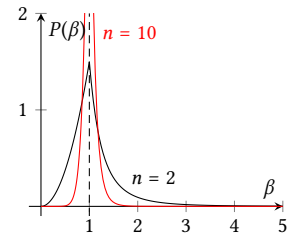
Proměnná n má v tomto případě podobný význam jako u SBX a určuje tvar distribuce. Opět větší hodnota n vede k větší pravděpodobnosti menších změn.

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.

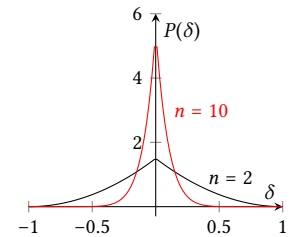
V diferenciální evoluci se jedinec vytváří pomocí jediného operátoru, který v sobě kombinuje jak mutaci, tak křížení. Jeho vstupem jsou hned čtyři jedinci z populace rodičů. Jako rodič p_4 se většinou volí postupně všichni

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 evolutionstrategie*. 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

$$\sigma = \sigma \exp^{1/\sqrt{d+1}}(f(\vec{x}_1) - f(\vec{x}) - 1/5),$$

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**

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

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

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

- 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

Vícekriteriální optimalizace

Zatím jsme se zabývali jen řešením problémů, které obsahují jen jedno kritérium. řešení takových problémů pomocí evoluce je často přímočaré - to jediné kritérium se použije přímo jako fitness, případně se tato fitness může nějakým způsobem škálovat. Nicméně prakticky se často vyskytují problémy, kde je potřeba optimalizovat zároveň kritérií více. Typickým příkladem může třeba být optimalizace tvaru křídla u letadla, od kterého chceme co největší vztlak při co nejmenším čelním odporu. Takové problémy se řeší v oblasti vícekriteriální optimalizace.

Obecně máme tedy optimalizační problém typu

$$\begin{array}{ll} \min_x & f_1(x), \dots, f_n(x) \\ \text{za podmínek} & g_i(x) \leq 0 \\ & h_i(x) = 0, \end{array}$$

kde $f_i : D \rightarrow \mathbb{R}$ jsou kritéria a $g_i, h_i : D \rightarrow \mathbb{R}$ jsou omezující podmínky, přičemž D je nějaký prostor parametrů všech těchto funkcí. Například ve spojitě optimalizaci to je nějaká podmnožina \mathbb{R}^n , v kombinatorické optimalizaci například množina všech permutací na $\{1, \dots, n\}$.

Prvním z problémů ve vícekriteriální evoluci je, že kritéria si typicky navzájem odporují, tedy dobrá hodnota jednoho kritéria typicky znamená špatnou hodnotu jiného. Řešením problému vícekriteriální evoluce tedy není jeden bod, ale je to celá množina bodů, tzv. Pareto optimální množina. Ta je definována na základě tzv. Pareto dominance. Říkáme, že bod x dominuje bod y ($x \preceq y$) pokud $f_i(x) \leq f_i(y)$ pro všechna i a zároveň $f_j(x) < f_j(y)$ pro alespoň jedno j . Pareto optimální množina je potom definována jako množina bodů z D , které nejsou dominované žádným jiným bodem z D . Občas se definuje i tzv. Pareto optimální fronta, která se skládá z funkčních hodnot bodů v Pareto optimální množině, tedy pro Pareto optimální množinu P a kritéria f_1, \dots, f_n , Pareto optimální fronta je množina $\{(f_1(x), \dots, f_n(x)) | x \in P\}$.

Pareto optimální množina je v typickém případě velká, často dokonce nekonečná (obzvlášť ve spojitě optimalizaci), není tedy možné najít jí celou a algoritmy vrací jen nějakou její konečnou aproximaci. Po takové aproximaci typicky chceme, aby body v ní byly blízko skutečně Pareto optimálním bodům (tj. takovým, že je žádný bod nedominuje) a zároveň by měly být víceméně rovnoměrně rozprostřeny po celé Pareto optimální frontě. Těmto dvěma požadavkům se říká konvergence a rozprostření.

Metriky pro porovnání algoritmů

Problém s porovnáním dvou řešení se opakuje znovu na vyšší úrovni, pokud chceme porovnat dvě různé aproximace Pareto optimální množiny, např. v případě, že chceme sledovat konvergenci algoritmu, nebo chceme porovnat dva různé algoritmy pro vícekritériální evoluci. Existuje několik různých metrik, které nějakým způsobem porovnávají dvě aproximace Pareto optimální množiny (většinou pracují s Pareto optimální frontou).

Jednou z prvních bylo tzv. pokrytí (cover), které je pro dvě stejně velké aproximace Pareto fronty A, B definované jako

$$C(A, B) = \frac{\text{počet bodů z množiny } B \text{ dominovaných body z množiny } A}{\text{počet bodů v množině } A}.$$

Nevýhodou tohoto kritéria je, že je binární, tj. pokud chceme porovnávat více aproximací, musíme porovnání udělat pro každou dvojici. Navíc kritérium vyjadřuje hlavně konvergenci k Pareto optimální frontě.

Častěji se používají tzv. unární indikátory, které vyjadřují kvalitu nalezené aproximace jako jedno číslo. Příkladem může být například tzv. generační vzdálenost (generational distance, GD), která je pro aproximaci A Pareto optimální fronty P definována jako průměrná vzdálenost bodu z A k bodu z P . Lepší aproximace mají menší hodnoty GD. Zásadní problém GD je, že vyjadřuje jen konvergenci, pokud budou všechny body z A těsně u sebe a blízko k Pareto optimální frontě, hodnota GD bude nízká. Tento problém řeší inverzní GD (IGD), která místo toho počítá vzdálenost od každého bodu z P k nejbližšímu bodu z A . Tímto postupem se započítá jako konvergence tak rozprostření.

V současnosti se nejčastěji jako indikátor kvality nalezeného řešení používá hyperobjem prostoru dominovaného danou aproximací Pareto optimální fronty. Formálně se dá pro aproximaci Pareto optimální fronty A a referenční bod $r \in \mathbb{R}^n$ jako

$$H(A, r) = \int_{\mathbb{R}^n} \text{Char}(\{x \in \mathbb{R}^n \mid \exists a \in A \ a \leq x \leq r\})(z) \, dz,$$

kde $\text{Char}(X)$ je charakteristická funkce množiny X . Jako referenční bod r se často volí vektor $(1, \dots, 1)$ po té, co se celá aproximace naškáluje na interval $[0, 1]$. Výhodou hyperbojemu je, že (podobně jako IGD) kombinuje jak konvergenci tak rozprostření. Jeho nevýhodou naopak je, že jeho výpočet roste s počtem kritérií a problém jeho výpočtu je ve skutečnosti #P-úplný.

Evoluční algoritmy pro vícekritériální evoluci se od jiných evolučních algoritmů liší především ve způsobu, jakým provádějí selekci. Použité genetické operátory jsou závislé na typu řešeného problému a kódování jedince a nijak se neliší od postupů používaných v jednokritériální evoluci.

Jedním z prvních evolučních algoritmů, který se snažil řešit problém vícekritériální evoluce byl algoritmus VEGA (Vector Evaluated Genetic Algorithm) navržený už v 80. letech. V tomto algoritmu se různé části populace vybíraly pomocí různých kritérií. Takový algoritmus ale z dnešního pohledu moc dobře nefunguje, neboť často konverguje k optimům jednotlivých kritérií a kompromisní řešení se moc často neobjevují. Moderní vícekritériální algoritmy pak jde rozdělit do dvou skupin – na algoritmy založené přímo

na Pareto dominanci (např. NSGA-II a MO-CMA-ES popsané níže) a na algoritmy založené na dekompozici vícekritériálního problému na množinu jednokritériálních (např. MOEA/D popsaný na konci této kapitoly).

Evoluční algoritmus s nedominovaným tříděním

Přímo na základě Pareto dominování lze definovat fitness funkci v evolučním algoritmu. Takový postup používá mnoho vícekritériálních algoritmů, jako příklad si ukážeme populární algoritmus NSGA-II (Non-dominated Sorting Genetic ALgorithm II). Myšlenkou nedominovaného třídění je, že pokud vezmeme dva jedince z populace a jeden z nich dominuje druhého, je určitě lepší. Při selekci se tedy jedinci rozdělí do tzv. nedominovaných front – v první frontě jsou jedinci, kteří nejsou dominováni žádnými jinými jedinci v populaci, v druhé frontě jsou jedinci, kteří jsou dominováni jen jedinci z první fronty atd. Tímto způsobem se celé populace rozdělí do několika front přičemž jedinci ve frontě s menším číslem se považují za lepší, než jedinci ve frontě s vyšším číslem. Pro porovnání jedinců ve stejné frontě potřebuje nějaké druhotné kritérium. Zde původní verze NSGA-II uvažuje crowding distance, v zásadě jde o vzdálenost jedince od nejbližších jedinců ve stejné Pareto optimální frontě. Jedinci s vyšší hodnotou této vzdálenosti jsou lepší, než jedinci s menší hodnotou, protože jsou v částech prostoru, které jsou méně obsazeny.

Výše popsaná selekce sama o sobě nezajišťuje žádnou formu elitismu. Ten se do algoritmu dostane tak, že se před environmentální selekcí populace rodičů a potomků sloučí do jedné, ze které se potom výše popsaným způsobem vybere správný počet nejlepších jedinců. Selektce je tedy deterministická.

Algoritmus NSGA-II funguje relativně dobře pro problémy s dvěma nebo třemi kritérii, ale právě kvůli nedominovanému třídění má problém, pokud je počet kritérií větší. V takovém případě je totiž mnohem větší šance, že dva jedinci budou navzájem nedominováni. To potom vede k tomu, že většina jedinců v populaci patří do první nedominované fronty a algoritmus ztrácí jakýkoliv tlak na konvergenci k Pareto optimální frontě (optimalizují se jen vzdálenosti mezi řešeními). Tento problém se dá částečně obejít tím, že se druhotné třídící kritérium nahradí za nějaké jiné, např. za příspěvek daného jedince k hyperobjemu celé nedominované fronty. Ačkoliv to situaci trochu zlepšuje, NSGA-II typicky nefunguje moc dobře pro problémy s více než třemi kritérii.

Vícekritériální evoluční strategie

V případě, že chceme řešit vícekritériální problémy ve spojitě optimalizaci, může se nám hodit vícekritériální verze evoluční strategie CMA-ES, tedy algoritmus MO-CMA-ES. Ten je zajímavý i tím, že každý jedinec v jeho populaci je vlastně instance $(1 + 1) - \text{CMA} - \text{ES}$, v populaci μ jedinců tak máme vlastně μ nezávislých instancí CMA-ES algoritmu.

V každé generaci MO-CMA-ES každý jedinec vygeneruje nové řešení pomocí své instance $(1 + 1)$ -CMA-ES. Toto řešení se porovná s rodičem a pokud je lepší (tj. dominuje ho, nebo jsou obě řešení neporovnatelná a potomek je lepší v druhotném kritériu) aktualizují se parametry jeho

instance CMA-ES. Na konci generace se použije selekce z NSGA-II algoritmu, která rozhodne, kteří jedinci přežijí do další generace.

Nevýhodou toho, že algoritmus obsahuje nezávislé instance CMA-ES je to, že se endogenní parametry těchto strategií adaptují velmi pomalu. Byla proto navržena i verze se sdílenými parametry, která tento problém řeší. I MO-CMA-ES, kvůli tomu, že používá stejnou selekci jako NSGA-II, má problém s větším množstvím kritérií.

Vícekritériální evoluce založená na dekompozici

Problém vícekritériální optimalizace se dá řešit pomocí jeho transformace na jednokritériální optimalizaci. Většinou se tím vícekritériální problém převede na několik jednokritériálních problémů (proto se tomuto přístupu také říká „dekompozice“). Takový postup je relativně jednoduchý a umožňuje přímo použít existující evoluční algoritmy, ale na druhou stranu při naivní implementaci potřebuje pro nalezení každého řešení spustit nový běh algoritmu.

Jednou z možností jak takovou transformaci udělat je použít vážený součet kritérií jako fitness funkci, tj.

$$f(x) = \sum_{i=1}^n w_i f_i(x),$$

kde w_i jsou váhy mezi 0 a 1. Tento způsob ale má zásadní nevýhodu – funguje jen v případě, že Pareto optimální fronta je konvexní. Z tohoto pohledu je lepší tzv. Čebyševova dekompozice, která funkci definuje jako

$$f(x) = \lambda_i |f_i(x) - f_i^*|,$$

kde λ_i jsou opět váhy mezi 0 a 1 a $f_i^* = \min_x f_i(x)$. Takový způsob dekompozice funguje i pro nekonvexní Pareto optimální fronty.

Již jsme zmínili, že tyto dekompozice se typicky nepoužívají přímo, algoritmus MOEA/D (Multiobjective Evolutionary Algorithm Based on Decomposition) je ale právě na těchto dekompozicích založený. Místo toho, aby řešil postupně několik jednokritériálních problémů, řeší je všechny najednou a využívá navíc předpokladu, že podobné problémy (s podobnými vahami w_i , nebo λ_i) budou nejspíš mít podobná řešení. V MOEA/D odpovídá každý jedinec jednomu nastavení vah a předpokladu se využívá tak, že se definují okolí jedinců jako množiny řešení, které mají nejpodobnější váhy. Genetické operátory se potom s velkou pravděpodobností (typicky kolem 90 %) pouští jen na jedincích, kteří jsou v okolí. V rámci mutace je navíc možnost jedince vylepšit problémově-specifickou heuristikou. Po aplikaci genetických operátorů se jedinec porovná se svým rodičem, a pokud je lepší, nahradí ho. Zároveň se porovná i s ostatními jedinci v okolí a také je nahradí, pokud je lepší pro dekompozici, která jim odpovídá. V zájmu zachování diverzity v populaci se většinou počet jedinců nahrazených v okolí omezuje – typická hodnota jsou maximálně dva nahrazení jedinci.

Díky svému způsobu selekce a dekompozici problému na jednokritériální problémy funguje algoritmus MOEA/D dobře i pro problémy s větším množstvím kritérií.

Kombinatorická optimalizace

Problém obchodního cestujícího

Problém obchodního cestujícího (POC) je jedním z nejznámějších NP-úplných problémů. Jde o nalezení nejkratší cesty mezi n městy, která navštíví všechna města právě jednou a skončí v počátečním městě ²⁸.

Evoluční řešení problému obchodního cestujícího má zřejmou fitness — délku cesty — zatímco reprezentací řešení existuje několik a genetické operátory (hlavně operátor křížení) jsou na těchto reprezentacích velmi závislé. V následujícím ukážeme nejprve nejčastější reprezentaci pomocí cesty (permutace) ²⁹, které je sice velmi přímočará, ale vyžaduje specifické operátory křížení. O operátorech mutace a inicializace se zmíníme poté.

Reprezentace cestou

Reprezentace cestou je velmi názorná, jde o posloupnost celočíselných hodnot označujících pořadí měst tak, jak je navštívíme. Jedinci evolučního algoritmu tedy permutaci o n prvcích. (412876935) tedy říká, že řešení reprezentuje cestu městy v pořadí 4 – 1 – 2 – 8 – 7 – 6 – 9 – 3 – 5. Zjevnou nevýhodou této reprezentace je, že jednobodové křížení dvou permutací typicky nevyrobí potomky jako validní permutace.

Jednotlivé návrhy křížení proto pracují tak, aby opravily nebo doplnily jedince, který od svých rodičů zdědí vlastnosti, jež autoři považují za důležité. V různých kříženích se proto (kromě zachování permutace) zrcadlí heuristický přístup k tomu, co by měla rekombinace dvou cest v problému POC zohlednit.

Prvním návrhem křížení je PMX — křížení částečné shody nebo také částečného zobrazení. ³⁰ Goldberga a Linglea. Snahou tohoto křížení je zachovat co nejvíce měst na svých původních pozicích. Jde o dvoubodové křížení, které dvěma náhodnými řezy rozdělí každého rodiče na tři části, přičemž střední část použije k definici zobrazení, která města na stejných pozicích spolu korespondují. V příkladu na obr. definujeme tedy zobrazení měst: 1 ↔ 4, 8 ↔ 5, 7 ↔ 6, 6 ↔ 7. Křížení PMX vymění středy jedinců a dále doplní z původních cest ta města, která nezpůsobí konflikt, t.j. nejsou ve vyměněných středních částech. Nakonec se pro doplnění zbylých měst použije korespondence dané zobrazením dle střední části.

Davis navrhl jiný přístup ke křížení, tzv. křížení zachovávající pořadí (OX) ³¹, které se snaží co nejvíce zachovat relativní pořadí jednotlivých měst v cestě. Obdobně jako v případě PMX jde o dvoubodové křížení, které nejprve vymění středy jedinců. Dále přeuspořádáme cestu druhého rodiče

²⁸ V daném ohodnoceném úplném grafu máme najít nejkratší hamiltonovskou kružnici.

²⁹ Permutační zakódování jedinců je přirozené i u jiných problémů, zapamatujme si jej.

³⁰ Partially mapped/matched crossover

³¹ Order crossover

(123|4567|89) PMX (452|1876|93) :
 (...|1876|...) (...|4567|...)
 a zobrazení: 1-4 8-5 7-6 6-7
 lze doplnit (.23|1876|.9) (...2|4567|93)
 a dle zobr. (423|1876|59) (182|4567|93)

Obrázek 4: Příklad křížení PMX.

od druhého bodu křížení a popořadě doplňujeme prvního potomka tak, že vynecháváme města, která již jsou v nové střední části. Druhý potomek vznikne stejným způsobem z přeuspořádané cesty prvního rodiče, viz obr. .

(123|4567|89) OX (452|1876|93) :
 (...|1876|...) (...|4567|...)
 a přeuspořádáme cestu 2 od druhého bodu
 křížení: 9-3-4-5-2-1-8-7-6
 vyhodíme překřížená města z 1, zbyde: 9-3-2-1-8
 doplníme potomka 1: (218|4567|93)
 obdobně potomek 2: (345|1876|92)

Obrázek 5: Příklad křížení OX.

Na základě křížení OX vznikl algoritmus křížení modifikovaného křížení (ModX)³², které se ukázalo jako úspěšné v řešení problémů rozvrhování. ModX je vlastně OX jen s jedním křížícím bodem, kde oba rodiče přeuspořádáme tak, aby cesta začínala za křížícím bodem a pak křížíme dle principu OX od začátku obou jedinců, viz obr. .

³² Modified crossover

(0845|671239) ModX (6712|483590)

potomek 1: (6712084539)
 potomek 2: (0845671239)

Obrázek 6: Příklad křížení ModX.

Posledním z klasických křížení permutačních řešení POC, které si uvedeme, je cyklické křížení (CX)³³ navržené Davisem a Hollandem. Snahou autorů bylo, aby každá pozice v potomkovi odpovídala, pokud možno, jednomu s rodičů. Za tímto účelem se při tvorbě potomka pohybujeme v cyklech po permutaci podle odpovídajících měst. První pozici si zvolíme náhodně z jednoho z rodičů, pak se snažíme vzít to město, které je na stejné pozici u druhého z rodičů, ale bereme z pozice v prvním rodiči, atd., dokud nedokončíme cyklus. Pak doplníme města a pozicemi z druhého rodiče, viz obr. .

³³ Cycle crossover

Křížení rekombinací hran

Hlavní myšlenkou křížení rekombinací hran (ER)³⁴ je přesun pozornosti od uzlů grafu (městům) k hranám (cestám). Autoři Whitley s kolegy zjistili, že výše zmíněné operátory ve skutečnosti nezachovávají více jak 60% hran z obou rodičů. Algoritmus ER (a jeho vylepšení ER2) se tedy snaží pracovat tak, že v potomcích zachová co nejvíce již existujících hran. Pracuje ve dvou

³⁴ Edge recombination crossover

(123456789) CX (412876935)
 první pozice, náhodně třeba z prvního $P1=(1.....)$,
 teď musíme vzít 4, $P1=(1..4....)$, pak 8, 3 a 2
 $P1=(1234...8.)$, dál už nelze, doplníme z druhého
 $P1=(123476985)$
 Obdobně $P2=(412856739)$

Obrázek 7: Příklad křížení CX.

fázích — nejprve si vybuduje seznam sousedů všech vrcholů, a potom tvoří nové jedince s pomocí tohoto seznamu, viz alg. 7.

procedure ER

$K \leftarrow []$ prázdný seznam

$N \leftarrow$ první uzel náhodného rodiče

while Length(K) < Length(Parent) **do**

$K \leftarrow [K, N]$

Vyhoď N ze všech seznamů sousedů

if seznam sousedů $N \neq \emptyset$ **then**

$N^* \leftarrow$ soused s nejmeně sousedy v seznamu (nebo náhodný z více takových)

else

$N^* \leftarrow$ náhodně zvolený uzel $\notin K$

end if

$N \leftarrow N^*$

end while

end procedure

Algoritmus 7: Křížení rekombinací hran

Obr. a ukazují práci původní varianty algoritmu a jejího vylepšení, které dává přednost výběru měst, která jsou v seznamu dvakrát (označené #). Dodejme, že algoritmus se může zastavit ještě před sestavením kompletní permutace, ale to se dle praktických zkušeností děje jen zřídka (v 1–1,5% případů). Algoritmy křížení ER jsou v praxi velmi úspěšné.

Další operátory

Zatím jsme hovořili jen o kříženích, zastavme se nyní u dalších operátorů pro problém POC.

Inicializaci jedinců na začátku algoritmu můžeme provést generováním náhodných permutací, ale lze využít i heuristik pro lepší výchozí řešení. Jednou z nich je jednoduchý algoritmus hladového typu, který inicializuje tak, začne náhodným městem a pak postupně vybírá nejbližšího souseda k již určenému městu. Jinou možností inicializace je algoritmus vkládání hran viz alg. 8.

Mutačních operátorů vhodných pro POC je celá řada, i když nejjednodušší (změna jednoho města) naopak použít nelze. Oblíbené jsou ale mutace Posun podcesty, Záměna 2 měst, Záměna podcesty nebo heuristiky jako je 2-opt viz alg. 9³⁵. Zajímavé na permutační reprezentaci také je, že u ní funguje inverze.

³⁵ G. A. CROES (1958). A method for solving traveling salesman problems. *Operations Res.* 6 (1958), pp., 791-812.

(123456789) ER (412876935)

```
1:  9 2 4
2:  1 3 8
3:  2 4 9 5
4:  3 5 1
5:  4 6 3
6:  5 7 9
7:  6 8
8:  7 9 2
9:  8 1 6 3
```

Obrázek 8: Příklad křížení ER.

Začnu v 1, následníci jsou 9, 2, 4
 9 vypadává, má 4 násl., z 2 a 4 náhodně vyberu 4
 násl. 4 jsou 3 a 5, беру 5,
 neboť 5 má 3 násl, zatímco 3 má 4 násl.
 Ted' máme (145.....), podobně pokračujeme
 ... (145678239)

(123456789) ER (412876935)

```
1:  9 #2 4
2:  #1 3 8
3:  2 4 9 5
4:  3 #5 1
5:  #4 6 3
6:  5 #7 9
7:  #6 #8
8:  #7 9 2
9:  8 1 6 3
```

Obrázek 9: Příklad křížení ER2.

(12.....), (128.....), (1287.....), (12876....)
 náhodně např. (128765...), (1287654..), (12876543.)
 (128765439)

procedure VKLÁDÁNÍ HRAN

cesta $T \leftarrow$ náhodná hrana

while Length(K) < n **do**

vyber nejbližší město c k cestě T

najdi hranu $(k - j) \in T$ tak, že minimalizuje rozdíl mezi $(k - c - j)$

a $(k - j)$

Vyhoď $(k - j)$ z T

Vlož $(k - c)$ do T

Vlož $(c - j)$ do T

end while

end procedure

Algoritmus 8: Inicializace vkládáním hran

```

procedure 2-OPT(cesta, i, k)
  new1 ← cesta[1] .. cesta[i-1]
  new2 ← cesta[i] .. cesta[k] v převráceném pořadí
  new3 ← cesta[k+1] .. cesta[n] return new1+new2+new3
end procedure

```

Algoritmus 9: Heuristika 2-opt pro lokální zlepšení cesty

Jiné reprezentace

Přesto, že permutační zakódování jedinců POC je nejčastější, zkoumají se i jiné možnosti reprezentace cest.

Jednou z nich je reprezentace sousednosti³⁶, která cestu reprezentuje také jako seznam měst, ale platí že město j je na pozici i , vede-li cesta z i do j . Takže např. (248397156) odpovídá cestě 1-2-4-3-8-5-9-6-7. Každá cesta má jen 1 reprezentaci, ale některé seznamy negenerují přípustnou cestu. Klasické křížení v této reprezentaci také nefunguje (a navrhuji se různé opravy), ale hlavní motivací pro její zavedení bylo, že v ní mají dobrý význam klasická schémata. Např. (*3*...) značí všechny cesty s hranou 2-3.

³⁶ Adjacency representation

Problém fungování klasického jednobodového křížení se snaží řešit ordinální reprezentace³⁷. Ta je založena na myšlence, že číslo reprezentující jedno město nebude označovat jeho absolutní název ale jen relativní pozici (pořadí) v bufferu všech měst, ze kterého při konstrukci jedince použita města mažeme. Takže například máme-li buffer (123456789) a cestu 1-2-4-3-8-5-9-6-7, její reprezentace je (112141311). Relativita ordinální reprezentace způsobí, že nyní můžeme dva jedince křížit jednobodově (potomci jsou syntakticky správně), ale význam takového křížení není zřejmý, jde spíše o makromutaci.

³⁷ reprezentace bufferem

Existuje i několik pokusů o maticové reprezentace cest jako jedinců evolučního algoritmu, většinou jde o binární matice, kde jednička na pozici (i, j) znamená buď, že v cestě vede hrana $i \rightarrow j$, anebo (častěji), že město i předchází v cestě město j . Operátory nad maticemi používají často logické operace nad prvky matic a opravné mechanismy. Většinou tyto reprezentace nedosahují výrazných výsledků.

Jiné problémy

Batoh

Evoluční algoritmy jsou aplikovány i na řešení dalších kombinatorických problémů, přičemž je zajímavé, že některé problémy, ač jsou např. také NP-těžké jako POC, představují jiné překážky pro EVA. Uvažme například problém 0-1 batohu, který je definován následovně:

Mějme množinu věcí očíslovaných od 1 do n , každá má hmotnost w_i a cenu v_i , a maximální nosnost batohu W . Úkolem je maximalizovat:

$$\sum_{i=1}^n v_i x_i$$

za podmínek

$$\sum_{i=1}^n w_i x_i \leq W \quad \text{a zároveň} \quad x_i \in \{0, 1\}$$

. x_i reprezentuje počet věci i v batohu, my počítáme s tím, že každou věc vezmeme maximálně jednou, proto 0-1 batoh. Zakódování problému se zdá —

na rozdíl od POC — triviální, můžeme použít bitovou mapu pro věci určené do batohu. Operátory křížení a mutace se zdají být také jednoduché. Problémem v tomto případě je fitness jedince. Za prvé chceme maximalizovat cenu, za druhé nechceme překročit kapacitu W , ale blížit se k ní. Lze využít multi-kriteriální EVA, ale v praxi se osvědčilo i velmi jednoduché řešení z rodiny opravných algoritmů dekódování jedince, které bývá překvapivě efektivní. Dekodér procházející bitový vektor jedince interpretuje jedničku na místě i jako: dej věc i do batohu, pokud by se nepřeplní.

Rozvrhování

Rozvrhování je dalším těžkým problémem, kde se evoluční algoritmy osvědčují jako heuristika pro nalezení přibližných řešení. Rozvrh reprezentujeme přímočaře maticí, kde řádky odpovídají učitelům, sloupce hodinám a hodnoty jsou kódy předmětů. Mutace většinou míchají předměty, křížení mění lepší řádky z jedinců. Problémem u rozvrhování je, jak určit fitness a jak pracovat s omezujícími podmínkami. Obecně se ukazuje, že dobré je definovat fitness řádku (t.j. jak je spokojen učitel) a tu pak zkombinovat s dalšími (často měkkými) kritérii kvality rozvrhu. Existující tvrdá omezení (kdy mohou učitelé, co kde lze učit, ...) je z důvodů efektivity často nutné řešit již v operátorech, aby se zbytečně negenerovala řada nepřipustných řešení.

Plánování výroby

Na problému plánování výroby ³⁸ si můžeme ukázat, že různá zakódování nám vyřeší různé těžkosti s řešením (a přinesou jiná). Problém je definován takto: Mějme výrobky o_1, \dots, o_N , sestávající z částí p_1, \dots, p_K . Pro každou část máme stejný (flow shop) nebo více (job shop) plánů, jak ji vyrobit na strojích m_1, \dots, m_M , a stroje mají různé doby nastavení na jiný výrobek. Úkolem samozřejmě je minimalizovat čas výroby. Tento čas se také jednoduše může stát fitness.

Evoluční řešení volí dvě cesty jak zakódovat jedince. První typ kódování je jednoduchý permutační — vyvíjený plán je jen permutace pořadí výrobků. Dekodér pak musí zvolit plány pro jejich části. Výhody jsou jednoduchá reprezentace a možnost použít křížení např. z problému POC. Ale ukazuje se, že tento přístup není efektivní. Dekodér je nucen řešit tu komplikovanou část úlohy, výběr plánu, a také většina operátorů z POC není vhodná. Na druhou stranu, přímá reprezentace jedince jako celého plánu (nazývá se paralelní kódování strojů) klade nároky na specializované (komplexní) evoluční operátory ³⁹.

³⁸ shop scheduling

³⁹ Frank Werner. A survey of genetic algorithms for shop scheduling problems. In *Heuristics: Theory and Applications*, pages 161–222. 04 2013

Evoluce pro strojové učení

Evoluce pravidlových systémů

S rozvojem expertních systémů v 70. a 80. letech přišly i pokusy jak takové systémy znalostí navrhovat jinak než ručně ve spolupráci odborníků na umělou inteligenci s experty v dané problémové doméně. Znalosti v takových systémech, kterým se říká pravidlové, jsou většinou realizovány ve formě pravidel:

IF (podmínka) THEN (výsledek)

Problém tedy je, jak navrhnout evoluční algoritmus, který bude vyvíjet daná pravidla nebo jejich množiny. Důležitým faktorem, který ovlivňuje návrh takového algoritmu, je typ učení, pro který ho chceme použít. Evoluce pravidlových systémů se historicky uvažuje v kontextu učení s učitelem (např. úlohy klasifikace) a zpětnovazebného učení (vývoj řídicích mechanismů pro agenty či roboty).

Obecně se dá říci, že pro vývoj pravidel v úlohách učení s učitelem máme jednodušší způsob výpočtu a práce s fitness, zatímco v úlohách zpětnovazebného učení (někdy též nazývané on-line učení) je mechanismus interakce pravidlového systému s prostředím a distribuce odměn mezi pravidly složitější.

Základní principy pro evoluci pravidlových systémů byly položeny v 80. letech 20. století v rámci tzv. Michiganského a Pittsburghského přístupu. Od poloviny 90. let pak mluvíme o nové generaci klasifikačních systémů reprezentované algoritmem Rozšířený klasifikační systém (XCS).

Michiganský a Pittsburghský přístup

Prvním, kdo se pokusil skloubit pravidlové systémy a evoluční algoritmy, byl J. Holland, který navrhl koncept *učících se pravidlových systémů*. Pro Hollandův přístup je typické, že uvažoval od počátku problémy zpětnovazebného učení a jeho důraz je tedy na mechanismy distribuce odměny prostředí, zatímco tvar pravidel je co nejjednodušší. Prvním pokusem byl tzv. *Kognitivní systém 1 (CS-1)*, který již obsahuje hlavní rysy toho, čemu dnes říkáme Michiganský přístup, protože ho Holland vyvinul na Michiganské univerzitě. Skupina kolem DeJonga a jeho studentů z Pittsburghské univerzity přišla v algoritmu Učící systém 1 (CS-1) s alternativním přístupem, kterému se dnes říká Pittsburghský, a který je orientován na učení s učitelem a klade větší důraz na tvar pravidel a bohatost evolučních operátorů.

Hlavním rozdílem mezi oběma přístupy je, že v Hollandově případě je jedincem evoluce jedno pravidlo a jako systém pak slouží celá populace

pravidel. V Pittsburském přístupu se naopak uvažuje s jedincem jako kompletním pravidlovým systémem o několika (desítkách) pravidel. Populace pak obsahuje více takových jedinců, jak jsme z evolučních algoritmů zvyklí. Výsledkem evoluce je nejlepší jedinec, který představuje kompletní řešení problému.

Michiganský přístup: Hollandův LCS

Jako typický příklad Michiganského přístupu si ukážeme klasický model Hollandova LCS, což je nástupce algoritmu CS-1, který se snaží řešit adaptivní učení agentů, kde zpětná vazba od prostředí přichází jen pro určité stavy a ne po každé akci agenta.

Jedincem v Hollandově LCS je jedno pravidlo, které má co nejjednodušší tvar. Podmínka je tvořena bitovou maskou pro detekci jednotlivých vstupů/příznaků a může obsahovat hodnoty 0 - příznak není přítomen, 1 - příznak je přítomen a * - na příznaku nezáleží. Vstupy pravidla jsou buď přímo informace z prostředí poskytované receptory, anebo příznaky z interních zpráv, které jsou generovány THEN částí pravidel. Tato pravá strana pravidel tedy obsahuje tzv. zprávu, kterou pravidlo, když je aktivováno, vyšle do globální paměti typu black board. Zpráva slouží jednak k zakódování akcí agenta (např. robot postoupí o krok kupředu) a jednak k přenosu informací mezi pravidly (např. robotu dochází baterie).

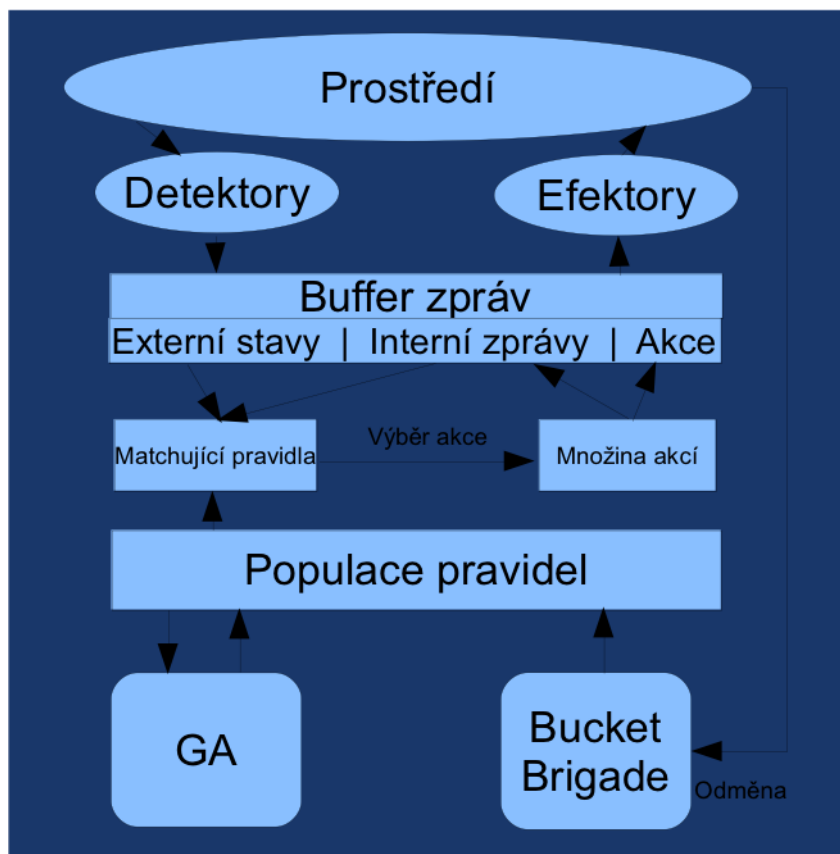
Formálně je jeden jedinec reprezentující pravidlo jednoduchým řetězcem v ternární abecedě, takže genetické operátory křížení a mutace jsou standardní a jednoduché. Holland používá také standardní genetický algoritmus, který ale přechod mezi generacemi realizuje jen zřídka — po mnoha krocích interakce agentů s prostředím — a nemění celou populaci, ale nahradí jen nějakou část nejhorších jedinců.

Pro práci s fitness navrhl Holland komplikovaný algoritmus hasičského družstva (bucket brigade), jenž se snaží řešit problém distribuce odměny posloupnosti akcí, které společně vedou k úspěšné interakci agenta s prostředím. Každý jedinec má kromě pravidla ještě číselnou proměnnou, jejíž hodnota určuje sílu pravidla danou jeho úspěšností.

Již jsme zmínili, že systém obsahuje sdílenou paměť, která uchovává aktuální zprávy jak od prostředí tak od předchozí aplikace pravidel. V jedné iteraci se zprávy z paměti porovnají se všemi pravidly a vyberou se ta pravidla, jejichž levá strana odpovídá nějaké zprávě. Tato pravidla vsadí část své síly jako poplatek za účast v soutěži, které pravidlo se skutečně uplatní. Hodnota sázky se počítá jako součin síly pravidla, jeho specifity (počtu 0 a 1 na v podmínce) a konstanty. Pravidlo, které vsadí nejvíce, vyhrává a je aplikováno — v další iteraci umístí svou zprávu do paměti, ale zároveň musí svou sázku zaplatit. Tato hodnota je pak distribuována mezi pravidla, která byla aplikována v minulých krocích. Tím dochází k tomu, že pravidla, která jsou součástí řetězce akcí vedoucího k dobrému výsledku, posilují (metafora řady hasičů předávající si vědro).

Pittsburghský přístup: GIL

Jako příklad Pittsburského přístupu uvedeme systém GIL, který je praktickým návrhem evoluce binárního klasifikačního systému, který je celý



Obrázek 10: Hollandův LCS s algoritmem požárního družstva.

reprezentován jedincem v evolučním algoritmu. Věnujeme se hlavně reprezentaci jedince a operátorům, kterých je velké množství a pracují na různých úrovních.

Jedinec je tedy v tomto systému množina pravidel, které klasifikují do jedné třídy, pravá strana je vždy stejná a proto ji můžeme v jedinci zanedbat. Každé pravidlo si můžeme představit jako disjunkci komplexů, kde každý komplex je konjunkce selektorů z jedné proměnné. Selektor reprezentuje množinu hodnot proměnné a je reprezentován bitovou mapou.

$((X=A1) \text{AND}(Z=C3)) \text{OR}((X=A2) \text{AND}(Y=B2))$
 $[001|11|0011 \text{ OR } 010|10|1111]$

Uvedme stručně výčet operátorů, které se podílejí na úpravách a rekombinacích jedinců, spíše než o mutacích a kříženích mluvíme o úrovních, na kterých operátory pracují.

Operátory na úrovni jedince: výměna pravidel, kopírování pravidel, generalizace pravidla, smazání pravidla, specializace pravidla, zahrnutí jednoho pozitivního příkladu

Operátory na úrovni komplexů: rozdělení komplexu na 1 selektoru, generalizace selektoru (nahrazení 11...1), specializace generalizovaného selektoru, zahrnutí negativního příkladu

Operace na selektorech: Mutace $0 \leftrightarrow 1$, rozšíření $0 \rightarrow 1$, zúžení $1 \rightarrow 0$,

ZCS a XCS

ZCS: Klasifikační systém nulté úrovně

V roce 1994 navrhl Wilson výrazně zjednodušenou verzi LCS, kterou nazval učícím se klasifikátorem nulté úrovně (ZCS). Hlavním rysem algoritmu bylo nahrazení distribuce odměny požárním družstvem pomocí přehledného a jednoduššího algoritmu, a zavedení nového operátoru pokrytí.

ZCS v každém kroku identifikuje množinu M (matching) pravidel, která jsou aplikovatelná na daný stav prostředí, z nich se vybere to, které se aplikuje čistě na základě fitness jednotlivých pravidel. Všechna pravidla z M , která navrhuji stejnou akci jako vítězné pravidlo, tvoří pak množinu A (action set).

Malý zlomek fitness každého pravidla z A je odečten a připraven k redistribuci. Nasčítané části fitness jsou zmenšeny o konstantu GAMMA. Systém si pamatuje členy množiny A z předchozí iterace, mezi které je tato hodnota rovnoměrně rozdělena.

Pokud systém v aktuálním kroku obdrží odměnu od prostředí, je od ní odečtena malá konstanta a zbytek je rovnoměrně rozdělen mezi členy aktuální množiny A .

Nakonec je ještě zmenšena fitness pravidel z množiny $M - A$ (těch, která jsou aplikovatelná na danou situaci, ale navrhuji jinou akci než vítězné pravidlo) o malý faktor τ .

Vlastní prohledávání je realizováno kombinací stabilního GA (steady-state GA) a operátoru pokrytí. V každém kroku je s určitou pravděpodobností realizována jedna iterace genetického algoritmu, který ruletovou selekcí vybere dva rodiče, vygeneruje dva potomky, a těmi nahradí dva jedince z populace, vybrané opět ruletovou selekcí. Rodiče předají potomkům polovinu své fitness.

Operátor pokrytí se aplikuje tehdy, když je množina M v daném kroku prázdná (nebo pokud obsahuje jen podprůměrné jedince), jeho cílem je zajistit, aby celý systém pokrýval co nejvíce případů, které prostředí poskytuje. Operátor pokrytí vygeneruje pravidlo s podmínkou odpovídající aktuálnímu vstupu a náhodnou akci na pravé straně.

XCS: Rozšířený klasifikační systém

Dodnes nejúspěšnějším přístupem v oblasti klasifikačních systémů je druhý algoritmus navržený Wilsonem, tak zvaný rozšířený klasifikační systém (XCS). Autor v něm explicitně aplikoval principy zpětnovazebního Q-učení, které se snaží o úplnost reprezentace problému výběru akce. Algoritmus byl navržen po zkušenostech se ZCS, který, ač byl na řadě úloh úspěšný, vykazuje tendenci soustředit prohledávání jen do oblastí přinášejících velkou odměnu od prostředí. Algoritmus XCS nepočítá fitness pravidel na základě hodnoty této odměny, ale na přesnosti předpovědi pravidel.

Ke každému pravidlu je přiřazena trojice hodnot — fitness F , chyba ϵ a predikce p . Algoritmus zpětnovazebního učení pak upravuje tyto hodnoty následujícím způsobem:

V každém kroku se identifikuje množina M a pro každou akci a v M se spočte predikce systému jako průměr predikcí pravidel v každé množině A

vážený jejich fitness. Odpověď systému se pak stanoví deterministicky nebo s náhodným faktorem. Pokud je množina M prázdná, použije se operátor pokrytí.

V následujících pěti krocích se pak upraví fitness a další parametry pravidel podle relativní přesnosti pravidla v dané množině A .

1. aktualizuj chybu každého pravidla: $\epsilon_j = \epsilon_j + \beta (|P - p_j| - \epsilon_j)$
2. aktualizuj predikci každého pravidla: $p_j = p_j + \beta(P - p_j)$
3. spočti přesnost pravidla κ_j jako: $\kappa = \alpha(\epsilon_0/\epsilon)^v$, anebo $\kappa = 1$ pro $\epsilon < \epsilon_0$
4. urči relativní přesnot pravidla κ_j' jako podíl jeho přesnosti a součtu všech přesností pro danou A .
5. Aktualizuj fitness F_j na základě relativní přesnosti pomocí moyennovy adaptivní procedury:

Pokud už fitness byla aktualizována $1/\beta$ krát, $F_j = F_j + \beta(\kappa_j' - F_j)$.

Jinak nastav F_j jako průměr aktuální a předchozí hodnoty κ_{ppj}' .

Nakonec se maximální hodnota $P(a_i)$ zmenší o faktor γ a použije se k aktualizaci pravidel z předchozího kroku.

Genetický algoritmus je opět stabilního typu, ale navíc pracuje s faktorizací prostoru pravidel do množin A , jde tedy o nikový GA vybírající pravidla z aktuální množiny A . Nahrazovaná pravidla jsou vybírána globálně s ohledem na vyvážení velikostí jednotlivých A . Pro danou množinu A se rozhoduje o tom, zda se GA aplikuje na základě průměrného času, kdy se naposledy daná nika A podílela na genetickém algoritmu.

Systém XCS se od původního návrhu dočkal mnoha rozšíření a vylepšení, jako jsou UCS (sUpervised classifier system) vhodný pro učení s učitelem, a jeho další varianty ExSTraCS v. 1 a 2, které byly úspěšné v řešení těžkých praktických problémů. Systém XCSF je zase rozšířením XCS pro problémy aproximace funkcí.

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 ⁴⁰.

40

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 10: 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 programy ⁴¹. 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 11: 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 sítě s předem danou strukturou, jde tedy o využití evolučního algoritmu jako alternativy ke standardnímu učení neuronových sítí založených typicky na lokálních metodách gradientní optimalizace ⁴². 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.

42

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 konfigurace ⁴³.

43

Jednou z dnes nejúspěšnějších neuroevolučních metod je Stanleyho Neat

(neuroevolution of augmenting topologies)⁴⁴. 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>Vícekriteriální optimalizace</i> |
21 |
| <i>Metriky pro porovnání algoritmů</i> | 22 |
| <i>Evoluční algoritmus s nedominovaným tříděním</i> | 23 |
| <i>Vícekriteriální evoluční strategie</i> | 23 |
| <i>Vícekriteriální evoluce založená na dekompozici</i> | 24 |
|
<i>Kombinatorická optimalizace</i> |
25 |
| <i>Problém obchodního cestujícího</i> | 25 |
| <i>Reprezentace cestou</i> | 25 |
| <i>Křížení rekombinací hran</i> | 26 |
| <i>Další operátory</i> | 27 |
| <i>Jiné reprezentace</i> | 29 |
| <i>Jiné problémy</i> | 29 |
| <i>Batož</i> | 29 |
| <i>Rozvrhování</i> | 30 |
| <i>Plánování výroby</i> | 30 |
|
<i>Evoluce pro strojové učení</i> |
31 |
| <i>Evoluce pravidlových systémů</i> | 31 |
| <i>Michiganský a Pittsburghský přístup</i> | 31 |
| <i>Michiganský přístup: Hollandův LCS</i> | 32 |
| <i>Pittsburghský přístup: GIL</i> | 32 |
| <i>ZCS a XCS</i> | 34 |
| <i>ZCS: Klasifikační systém nulté úrovně</i> | 34 |
| <i>XCS: Rozšířený klasifikační systém</i> | 34 |
|
<i>Rojové algoritmy</i> |
37 |
|
<i>Genetické programování</i> |
39 |

Neuroevoluce 41

Rejstřík 51

Literatura 55

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 |
| 4 | Příklad křížení PMX. | 26 |
| 5 | Příklad křížení OX. | 26 |
| 6 | Příklad křížení ModX. | 26 |
| 7 | Příklad křížení CX. | 27 |
| 8 | Příklad křížení ER. | 28 |
| 9 | Příklad křížení ER2. | 28 |
| 10 | Hollandův LCS s algoritmem požárního družstva. | 33 |

Seznam tabulek

Seznam algoritmů

| | | |
|----|---|----|
| 1 | Schéma evolučního algoritmu | 6 |
| 2 | Schéma Hollandova gentického 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 | Křížení rekombinací hran | 27 |
| 8 | Inicializace vkládáním hran | 28 |
| 9 | Heuristika 2-opt pro lokální zlepšení cesty | 29 |
| 10 | Schéma optimalizace rojem částic minimalizujících $f : \mathbb{R}^n \rightarrow \mathbb{R}$ | 38 |
| 11 | Schéma Kozova algoritmu genetického programování nad syntaktickými stromy | 40 |





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.
- [26] Frank Werner. A survey of genetic algorithms for shop scheduling problems. In *Heuristics: Theory and Applications*, pages 161–222. 04 2013.