Přírodou inspirované prohledávací algoritmy a jejich aplikace

Roman Neruda
Ústav informatiky AV ČR, v.v.i.
Pod Vodárenskou věží 2
182 07 Praha
roman@cs.cas.cz

Abstrakt

V tomto článku přinášíme stručný přehled algoritmů prohledávání, které jsou inspirované přírodními jevy jako je evoluce nebo chování sociálního hmyzu. Základními princpy těchto algoritmů je heuristické prohledávání prostoru pomocí generování kandidátů řešení problému a ověřeníni jejich úspěšnosti, populační prohledávání, kde jedinci v populacích soupeří o možnost reprodukce, a rekombinace kandidátů za účelem nalezení lepších řešení. V práci nejprve popíšeme obecný evoluční algoritmus a následně se zaměříme na jeho konkrétní varianty: genetické algoritmy, evoluční programování, evoluční strategie a genetické programování. Popíšeme také neuroevoluční algoritmy pro evoluční učení struktury a parametrů neuronových síti a rojové algoritmy.

1 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 prohledávaní, 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 prinicpipy.

V následujícím textu nejprve stručně zmíníme inspiraci a obecné rysy evolučních algoritmů a pak se budeme věnovat několika konkrétním oblastem vycházejících 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 dekódováním 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 inspiruje chováním společenského hmyzu a hejn pro efektivní prohledávání různých prostorů, např. reálných euklidovských prostorů nebo hledání cest v grafech.

2 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 využizí evolučního prohledávání z roku 1948 a Bremermannovy první pokusy o implementaci optimalizace pomocí evoluce a rekombinace z roku 1962. 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, zatímco skupina Fogela a spolupracovníků vyvinula metodu nazvanou evoluční programování. Nezávisle na nich přišli Rechenberg a Schwefel v Německu na metodu nazvanou evoluční strategie. 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 fenomény, konkrétně jde o Darwinovu teorii přírodního výběru a zjednodušené principy genetiky, které poprvé načrtl Mendel. Z genetiky se evoluční algoritmy inspirují diskrétní 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 řešeni 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 ohonotíme 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 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.

Algoritmus 1 Schéma evolučního algoritmu

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

Výběr jedinců do nové populace (tedy jakési slití rodičů a potomků) má na starosti enviomentální selekce beroucí v úvahu fitness jedinců a připadně 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.

3 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 adaptace [1]. 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 explorací a exploatací.

Nejjednodušší varianta genetického algoritmu pracuje s binárními jedinci, to znamená, že parametry řešené 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 vyměňí 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).

Algoritmus 2 Schéma Hollandova gentického algoritmu

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 $i \leftarrow 1, \ldots, N/2$ **do**
      vyber z P_t 2 rodiče *Ruletovou selekcí*
      S pravděpodobností p_C *Zkřiž* rodiče
      S pravděpodobností p_M *Mutuj* potomky
      Ohodnoť potomky
      Přidej potomky do P_{t+1}
    end for
    Zahoď P_t
    $t \leftarrow t + 1$
  end while
end procedure

Ruletovou selekci 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.

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 mutace [4, 5]. O některých se zmíníme dále.

4 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 automatů [2]. 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.

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.

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.

5 Evoluční strategie

Evoluční strategie byly navrženy pro práci s reálnými vektory reprezentujícími parametry složitých optimalizačních problémů [3]. V ukázkovém algoritmu vidíme jednoduchou verzi, která pracuje s jedním jedincem v populaci generujícím jednoho potomka pomocí gaussovské mutace popsané výše. Toto schéma lze snadno rozšířit na populaci n jedinců, která generuje m potomků (typicky je $m > n$). Enviromentální selekce se dělí na tzv. $(n + m)$ strategii, která n nových jedinců generuje deterministicky výběrem lepších z množiny rodičů i potomků, a na tzv. (n, m) strategii, která n nových jedinců vybírá deterministicky jen z potomků. Ukazuje se, že druhá strategie je robustnější proti uváznutí v lokálních extrémech.

Skutěcně používané evoluční strategie jsou také charakteristické

Algoritmus 3 Schéma meta-evolučního programování nad vektorem reálných čisel

procedure META-EP
  $t \leftarrow 0$
  Inicializuj populaci P_t N náhodně vygenerovanými reálnými vektory $\vec{x^t} = (x_1^t, \ldots, x_n^t, \sigma_1^t, \ldots, \sigma_n^t)$
  Ohodnoť jedince v populaci P_t
  while neplatí *kritérium ukončení* **do**
    for $i \leftarrow 1, \ldots, N$ **do**
      k rodiči $\vec{x_i^t}$ vygenruj potomka $\vec{y_i^t}$ *mutací*:
      for $j \leftarrow 1, \ldots, n$ **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 4 Schéma základní dvouprvkové evoluční strategie pro minimalizaci účelové funkce $f : \mathcal{R}^n \rightarrow \mathcal{R}$

procedure EVOLUČNÍ STRATEGIE (1+1)
  $t \leftarrow 0$
  Generuj náhodně počáteční bod $(x_1^t, \ldots, x_l^t) \in \mathcal{R}^n$
  while $\left(f(\vec{x^t}) > \textit{kritérium ukončení}\right)$ **do**
    for $i \leftarrow 1, \ldots, l$ **do**
      zvol z z normálního rozdělení
      $y_i^t \leftarrow x_i^t + z$
    end for
    if $f(\vec{x^t}) \leq f(\vec{y^t})$ **then**
      $x^{\vec{t+1}} \leftarrow \vec{x^t}$
    else
      $x^{\vec{t+1}} \leftarrow \vec{y^t}$
    end if
    $t \leftarrow t + 1$
  end while
end procedure

používáním adaptačních mutací popsaných v předchozí části, které mohou kromě rozptylů obsahovat i další parametry. Na rozdíl od evolučního programování, křížení je v evolučních strategiích používáno. U křížení reálných vektorů jsou nejčastější dva přístupy (používané i v ostatních oblastech evolučních algoritmů). Uniformní křížení je zobecněním jednobodového křížení, které ale pro každou složku jedince rozhoduje náhodně, zda bude pocházet z prvního nebo druhého rodiče. Dalším typem křížení jsou tzv. aritmetická křížení, která se snaží kombinovat hodnoty jednotlivých parametrů rodičů. Nejčastějším způsobem je průměrování nebo konvexní kombinace hodnot.

6 Genetické programování

Gentické programování je zajímavé tím, že přináší práci s jedinci odlišného typu — jde o stromy reprezentující programy [7]. 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 suplují 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 nestejně dlouhými větvemi. Pro inicializaci se doporučuje kombinace těchto metod v poměru 1:1 (ramped half-and-half).

7 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

Algoritmus 5 Schéma Kozova algoritmu genetického programování nad syntaktickými stromy

```
procedure Genetické programování
    t ← 0
    Inicializuj populaci P_t metodou ramped half-and-half
    Ohodnoť jedince v populaci P_t
    while neplatí kritérium ukončení do
        i ← 0
        repeat
            vyber pravděpodobnostně variační operaci o ∈ {C, M}
            if o=C then                                          ▷ Křížení
                vyber z P_t 2 rodiče
                Zkřiž rodiče
                vlož potomky do P_{t+1}
                i ← i + 2
            else                                                 ▷ Mutace
                vyber z P_t 1 rodiče
                Mutuj rodiče
                vlož potomka do P_{t+1}
                i ← i + 1
            end if
        until i ≥ n
        Ohodnoť jedince v populaci P_{t+1}
        Zahoď P_t
        t ← t + 1
    end while
end procedure
```

jako alternativy ke standardnímu učení neuronových sítí založených typicky na lokálních metodách gradientní optimalizace [9]. 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.

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 uváznutí v lokálních extrémech a snadná paralelizace. Jednou z oblastí, kde je evoluční algoritmus pro učení neuronových síti 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 evoluční 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 připadě reprezentace struktury) nebo reálná (pro reprezentaci struktrury 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 možnosti zachytit symetrie struktury. Gruau navrhl jinou metodu založenou na principu genetického programování, kde program je vlastně návod na sestrojení sítě metodou růstu z minimální konfigurace [10].

Jednou z dnes nejúspěšnějších neuroevolučních metod je Stanleyho Neat (neuroevolution of augmenting topologies) [11]. 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.

8 Rojové algoritmy

Rojové algoritmy mají některé společné rysy s evolučními algoritmy, jako je populační prohledávání, ale jsou charakteistické 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 [8].

Principem fungování rojově optimalizace je organizovaný pohyb roje částic (jedinců) po prostoru parametrů. Každá částice je charakterizována svou polohou a vektorem rychlosti. Každá častice 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 tři vektorů: současné rychlosti, směru k lokálnimu extrému častice 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.

9 Závěr

V tomto příspěvku jsme stručně představili oblast přírodou inspirovaných prohledávacích algoritmů. Nedostalo se na některé moderní či pokročilejší metody v této oblasti, jako je koevoluce, memetické algoritmy, diferenciální evoluce, efektivní paralelní implementace nebo multikriteriální evoluční algoritmy. Vývoj v posledních letech směřuje jednak ke specializaci evolučních technik v závislosti na doméně aplikace, a jednak ke kombinování metod

Algoritmus 6 Schéma optimalizace rojem částic minimalizujících $f : \mathcal{R}^n \to \mathcal{R}$

procedure OPTIMALIZACE ROJEM ČÁSTIC
  $\vec{g} \leftarrow \arg\max_{\vec{y}} f(\vec{y})$ ▷ inicializace globální nejlepší pozice
  for $i \leftarrow 1, \ldots, l$ **do**
    Inicializuj pozici částice $\vec{x_i}$ náhodně z rovnoměrného rozdělení
    $\vec{p_i} \leftarrow \vec{x_i}$ ▷ inicializace nejlepší známé pozice částice
    if $f(\vec{p_i}) < f(\vec{g})$ **then**
      $\vec{g} \leftarrow \vec{p_i}$ ▷ update globální nejlepší pozice
    end if
    Inicializuj rychlost částice $\vec{v_i}$ náhodně z rovnoměrného rozdělení
  end for
  while $\left(f(\vec{x^t}) > \textit{kritérium ukončení}\right)$ **do**
    for $i \leftarrow 1, \ldots, l$ **do**
      zvol r_g a r_p z ronvoměrného rozdělení $U(0, 1)$
      for $d \leftarrow 1, \ldots, n$ **do** ▷ 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
      $\vec{x_i} \leftarrow \vec{x_i} + \vec{v_i}$ ▷ update pozice částice
      if $f(\vec{x_i}) < f(\vec{p_i})$ **then**
        $\vec{p_i} \leftarrow \vec{x_i}$ ▷ update nejlepší pozice částice
        if $f(\vec{p_i}) < f(\vec{g})$ **then**
          $\vec{g} \leftarrow \vec{p_i}$ ▷ update nejlepší globální pozice
        end if
      end if
    end for
  end while
end procedure

mezi sebou. Časté jsou dnes i kombinace evolučních přístupů s dalšími obory jako jsou například metody kombinatorické optimalizace nebo jiné heuristiky lokálního prohledávání. Evoluční algoritmy jsou svým obecným rámcem vhodné pro nejrůznější kombinace, které se typicky uplatňují v návrhu kódování jedinců a příslušných evolučních operátorů. Jde o bouřlivě se rozvíjející obor, který zapadá do širšího rámce tzv. výpočetní inteligence — části umělé inteligence inspirované přírodou.

Poděkování

Tento článek vznikl za podpory projektu MŠMT COST LD 13002.

Reference

[1] J. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, 1995.

[2] D. Fogel, *Evolutionary Computation*, MIT Press, 1995.

[3] H.-G. Beyer and H. P. Schwefel, Evolutionary Strategies: A Comprehensive Introduction, *Natural Computing*, 1, 3-52, 2002.

[4] Z. Michalewicz, *Gentic Algorithms + Data Structures = Evolutionary Programs*, Springer, 1996.

[5] M. Mitchel, *Introduction to Genetic Algorithms*, MIT Press, 1996.

[6] A. E. Eiben and J. E. Smith *Introduction to Evolutionary Computing*, Springer, 2003.

[7] J. Koza, *Genetic Programming*, MIT Press, 1992.

[8] J. Kennedy and R. C. Eberhardt, *Swarm intelligence*, Morgan Kaufmann, 2001.

[9] I. G. Sher, *Handbook of Neuroevolution Through Erlang*. Springer, 2012.

[10] F. Gruau, *Neural network synthesis using cellular encoding and the genetic algorithm*, Ecole Normale Superieure de Lyon, France, 1994.

[11] K. O. Stanley and R. Miikkulainen. A Taxonomy for Artificial Embryogeny. *Artificial Life*, 9, 93-130, MIT Press, 2003.