Použitie viaccllenných evolučných stratégií pre nájdenie optima Rosenbrockovej funkcie

1. Viaccllenné evolučné stratégie

Evolučná stratégia, narozdiel od väčšiny stochastických metód, nie je založená na binárnej reprezentácii premennej, ale pracuje priamo reálnou reprezentáciou. Používa sa na hľadanie globálneho minima (maxima) reálnej funkcie f(**x**), kde $\mathbf{x} \in \mathbf{R}^N$. Hodnoty premenných vektora **x** možu byť obmedzené nejakým intervalom.

Najjednoduchšia verzia evolučnej stratégie je (1+1) evolučná stratégia. Jej základom je vzťah

$$\mathbf{x}' = \mathbf{x} + \mathbf{N}(0,\sigma),$$

kde $\mathbf{N}(0,\sigma)$ je vektor náhodných čísel so strednou hodnotou 0 a štandartnou odchýlkou σ. Nové riešenie $\mathbf{x}'$ je akceptované, ak $f(\mathbf{x}') < f(\mathbf{x})$. Ak nové riešenie je akceptované, potom ďalšie riešenie generujeme z nového, ináč zo starého riešenia, až kým nenájdeme optimum.

Viaccllenné evolučné stratégie sa vo všeobecnosti označujú (m+l) alebo (m,l). Stratégia (1+1) je ich špeciálnym prípadom a jej označenie znamená, že populácia rodičov má mohutnosť 1, rovnako ako populácia potomkov, ktorá z nej vzniká. Všeobecne (m+l) znamená, že veľkosť populácie rodičov je m a veľkosť populácie potomkov l. Pri výbere najlepších jedincov do novej generácie sa vyberá zo všetkých riešení, teda aj z rodičov aj z potomkov. Pri statégii (m,l) po vytvorení generácie potomkov generácia rodičov zaniká a potrebný počet najlepších riešení do novej populácie sa vyberá len z generácie potomkov.

Kríženie dvoch rodičov produkuje vždy len jedného potomka, narozdiel od genetických algoritmov, kde dvaja rodičia produkujú vždy dvoch potomkov.

Väčšinou sa pri evolučnej stratégii nepoužíva jednotná smerodajná odchýlka, ale smerodajná odchýlka pre každú premennú optimalizovanej funkcie zvlášť, pričom jej hodnota sa mení podľa toho, ako blízko predpokladáme, že sme od minima (maxima). Čím sme bližšie, tým by smerodajná odchýlka mala byť menšia, aby sme sa od nádejného minima veľmi nevzďaľovali. V pokročilých evolučných stratégiách vektor smerodajných odchýliek nie je jednotný pre celú populáciu chromozómov, ale každý chromozóm si nesie so sebou vlastný vektor smerodajných odchýliek. Nesie si tak so sebou informáciu o distribúcii pre novú mutáciu.

Pri viaccllenných evolučných stratégiách sú aj smerodajné odchýlky podrobené evolučnému procesu. Smerodajná odchýlka je mutovaná vzťahom

$$\sigma_j' = \sigma_j.\exp(N(0,\Delta\sigma_0)$$

Smerodajná odchýlka $\Delta\sigma_0$ je konštantná v celom časovom priebehu algoritmu, je parametrom metódy. Novovytvorená smerodajná odchýlka sa hneď použije na vygenerovanie novej hodnoty

$$x_j' = x_j + N(0,\sigma_j')$$

Okrem vlastnej hodnoty premennej vo vektore a jej smerodajnej odchýlky môže byť súbor premenných charakterizovaný aj ďalším vektorom strategických premenných, ktorý riadi koreláciu mutácií.

Pre dve premenné s rôznymi hodnotami smerodajnej odchýlky vrstevnice pravdepodobnosti umiestnenia mutovaného vektoru nepredstavujú elipsu ale kružnicu. Táto elipsa je však orientovaná v smere súradnicových osí. Optimum však nemusí byť umiestnené v smere dlhšej osi elipsy, ale niekde naprieč. Ideálne by potom bolo umiesniť túto elipsu vrstevníc pravdepodobnosti tak, aby hlavná os elipsy smerovala k optimu. Tak by mutovaný vektor mal najväčšiu šancu priblížiť sa k optimu. Toto natočenie sa dá zaistiť kovarianciami.

Mutácie, ktoré berú do úvahy aj kovariancie vyzerajú nasledovne:

$$\sigma_j' = \sigma_j.\exp(\tau_0.N(0,1) + \tau.N_j.(0,1))$$

$$\alpha_j' = \alpha_j + \beta.N_j.(0,1)$$

$$x_j' = x_j + z_j\ (\sigma',\alpha')$$

Generovať náhodný vektor mutácií s členmi $z_j\ (\sigma',\alpha')$ môže byť obtiažnym problémom (ťažko sa dá garantovať ortogonálnosť výsledného koordinačného systému). V praxi sa tento problém nahradzuje postupnou rotáciou. Ak máme dve premenné x_1 a x_2 s príslušnými smerodajnými odchýlkami, potom odchýlky upravené pomocou korelačného koeficientu majú tvar

$$\Delta x_1' = \Delta x_1 \cos\alpha - \Delta x_2 \sin\alpha$$

$$\Delta x_2' = \Delta x_1 \sin\alpha + \Delta x_2 \cos\alpha.$$

Nasledovný algoritmus 1.1 v pseudokóde je algoritmom pre pokročilú evolučnú stratégiu.

```
1   {
2   t= 0; σ = σ_in;
3   P_0 = [ náhodne generovaná populácia chromozómov ( = vektorov reálnzch premenných
4           spolu so štandartnými odchýlkami, prípadne kovarianciami) ]
5   Ohodnotenie každého chromozómu populácie zodpovedajúcou funkčnou hodnotou.
6   while (t < t_max || dostatočne dobré riešenie nie je nájdené) {
7      t++;
8      Q = [nové chromozómy vytvorené krížením a vždy použitou mutáciou z náhodne
9             vybratých chromozómov z P_i-1]
10     Ohodnotenie každého chromozómu Q zodpovedajúcou funkčnou hodnotou
11     P_i = [ Najlepšie chromozómy z Q, alebo z Q ∪ P_i-1]
12  }
```

Algoritmus 1.1 Evolučná stratégia typu (m,n) alebo (m+n) .

2. Rosenbrockova funkcia

Rosenbrockova funkcia je definovaná predpisom

$$f(x_1,x_2) = 100(x_2 - x_1)^2 + (x_1 - 1)^2 \qquad ()$$

Jej globálne minimum je v bode $x_1 = 1$, $x_2 = 1$, ktorý leží v hlbokom zatočenom údolí.

Nájsť globálne minimum Rosenbrockovej funkcie v rozumnom čase pomocou genetického algoritmu je takmer nemožné. V genetickom algoritme sa nedá nastaviť smer mutácií, preto by algoritmus rýchlo našiel údolie funkcie, ale nebol by sa schopný dostať až do globálneho minima. Toto nasmerovanie nám umožňuje práve viacčlenná evolučná stratégia pomocou kovariancií.

3. Aplikácia evolučnej stratégie na Rosenbrockovu funkciu

Globálne minimum Rosenbrockovej funkcie budeme hľadať pomocou viacčlennej evolučnej stratégie s použitím smerodajných odchýliek pre každú premennú zvlášť a kovariancií medzi premennými, teda uhol α. Chromozóm sa teda skladá z piatich členov:

$$\text{chromozóm} = [x_1, x_2, \sigma_1, \sigma_2, \alpha]$$

Počiatočné hodnoty premenných x_1, x_2 náhodne zvolíme z intervalu (-100,100), smerodajné odchýlky inicializujeme na 3.0 a uhol α na 0. Použijeme stratégiu (15,100), teda z rodičovskej 15-člennej polulácie vyrobíme 100 potomkov, z ktorej následne vyberieme 15 najlepších do novej generácie. Rodičovské chromozómy po vytvorení potomkov zmažeme.

Vyskúšame tri verzie viacčlennej evolučnej stratégie a porovnáme ich rýchlosť nájdenia globálneho minima.

Základná verzia používa uniformné kríženie dvoch náhodne vybratých rodičov pre premenné x_1, x_2 a kríženie priemerom pre strategické premenné σ_1, σ_2 a α. Ďalšie použité konštanty sú $\tau_0 = 0{,}5946$, $\tau = 0{,}5$, $\beta = 0{,}0873$. Táto verzia používa mutáciu s využitím kovariancií a kríženie.
Druhá verzia algoritmu pracuje bez použitia kovariancií., teda uhol α je vždy rovný nule, ale používa kríženie chromozómov.
Tretia verzia pracuje s kovarianciami, ale nepoužíva kríženie. Ak vytvárame nový chromozóm, vyberieme z rodičovskej populácie len jeden chromozóm, ten zmutujeme (aj s použitím kovariancií) a zmutované hodnoty priradíme novému chromozómu.

Aby sme mohli porovnať, ako rýchlo jednotlivé verzie viacčlennej evolučnej stratégie nájdu globálne minimum, každý program spustíme 100-krát a vypočítame priemerný počet generácií, ktoré boli potrebné na nájdenie bodu s funkčnou hodnotou menšou ako 10^{-6}.
Najväčšiu očakávanú rýchlosť by mala mať verzia, ktorá používa kríženie a kovariancie. Kovariancie nám totiž umožňujú lepšie algoritmus nasmerovať smerom ku globálnemu minimu. Preto aj verzia s kovarianciami a bez kríženia by mala byť rýchlejšia ako verzia bez kovariancií s použitím kríženia.

Verzia 1 : S použitím mutáciií a kovariancií

Nebudeme si podrobne popisovať celý program, použitý na otestovanie týchto troch verzií a ich porovnanie. Uvedieme si len funkciu, v ktorej sa tieto tri programy líšia.

Je to funkcia kriz_mut() typu CHROMOZOM, ktorá vyberie z populácie rodičov chromozómy a vytvorí križením a zmutovaním nový chromozóm.

```
CHROMOZOM P[CNT_P];

  . . . . .

1  CHROMOZOM kriz_mut()
2  {
3    int r[1];
4    CHROMOZOM novy;
5    double N_0_1_glob;
6    double delta_x1, delta_x2, d_x1, d_x2;

7    r[0] = random(CNT_P);
8    while (r[0] == (r[1] = random(CNT_P)));

9    novy.x1   = P[r[random(1)]].x1;
10   novy.x2   = P[r[random(1)]].x2;
11   novy.sig1 = (P[r[0]].sig1 + P[r[1]].sig1)/2;
12   novy.sig2 = (P[r[0]].sig2 + P[r[1]].sig2)/2;
13   novy.alfa  = (P[r[0]].alfa + P[r[1]].alfa)/2;

14   N_0_1_glob = gaussdev();

15   novy.sig1 *= exp(TAU_0 * N_0_1_glob) * exp(TAU * gaussdev());
16   novy.sig2 *= exp(TAU_0 * N_0_1_glob) * exp(TAU * gaussdev());
17   novy.alfa += BETA * gaussdev();
18   d_x1 = novy.sig1 * gaussdev();
19   d_x2 = novy.sig2 * gaussdev();
20   delta_x1 = d_x1 * cos(novy.alfa) – d_x2 * sin(novy.alfa);
21   delta_x2 = d_x1 * sin(novy.alfa) + d_x2 * cos(novy.alfa);
22   novy.x1 = perturbation(novy.x1 + delta_x1);
23   novy.x2 = perturbation(novy.x2 + delta_x2);

24   novy.f = f(x1,x2);

25   return novy;
}
```

Algoritmus 3.1 Funckia kriz_mut() je časť programu pre evolučnú stratégiu – základnú verziu s križením a mutáciou chromozómu s použitím kovariancií.

CNP_P označuje počet chromozómov v rodičovskej populácii. Pole P typu CHROMOZOM je pole chromozómov rodičovskej populácie. Funkcia gaussdev() je generátor náhodnej premennej s normálnym rozdelením so strednou hodnotou 0 a smerodajnou odchýlkou 1.

V riadkoch 7 – 10 náhodne vyberieme dvoch rodičov. Nasleduje kríženie v riadkoch 9 – 10. Pre premenné x_1 a x_2 je použité uniformné kríženie s náhodným výberom, teda pre každú premennú náhodne zvolíme rodiča, po ktorom potomok zdedí hodnotu príslušnej premennej. Pre smerodajné odchýlky σ_1, σ_2 a uhol α (kovarianciu premenných x_1, x_2) sme použili kríženie priemerom.
V riadkoch 14 – 21 nasleduje mutácia hodnôt premenných x_1, x_2 (novy.x1, novy.x2) a strategických premenných σ_1, σ_2 a α (novy.sig1, novy.sig2 a novy.alfa). Najprv zmutujeme smerodajné odchýlky σ_1, σ_2 a uhol α. Na základe zmutovaných strategických premenných získame odchýlky Δx_1 a Δx_2 (delta_x1, delta_x2), o ktoré sa zmutované premenné budú líšiť. Po pripočítaní zmien k premenným x_1, x_2, ešte hodnoty upravíme funkciou perturbation(). Táto funkcia upraví hodnotu premennej, ak padne mimo jej vymedzeného intervalu. K hodnote premennej priradí jej zrkadlový obraz vzhľadom na hranicu intervalu. Umiestni hodnotu na opačnú stranu hranice intervalu, v rovnakej vzdialenosti od hradice ako bola pôvodná hodnota.
Na záver vypočítame funkčnú hodnotu v novozískanom bode, teda funkčné ohodnotenie nového chromozómu, ktorý bude zaradený do novej populácie.

Verzia 2 : S použitím kríženia, bez použitia kovariancií

Druhá verzia evolučnej stratégie používa kríženie rodičovských chromozómov, ale pri mutáciách premenných a strategických premenných nepoužíva kovariancie (uhol α). Funkcia kriz_mut() sa podobá na funkciu použitú vo verzii evolučnej stratégie s kovarianciami. Tiež najprv náhodne vyberie dva chromozómy z rodičovskej populácie, ktoré následne skríži.
Funkcia sa však líši v mutácii nového vznikajúceho nového chromozómu. Oproti algoritmu 3.1 sa teda líši v riadkoch 10 – 23. Tie nahradíme riadkami, ktoré sú uvedené v algoritme 3.2.

```
13  novy.sig1 *= exp(TAU_0 * N_0_1_glob) * exp(TAU * gaussdev());
14   novy.sig2 *= exp(TAU_0 * N_0_1_glob) * exp(TAU * gaussdev());
15   d_x1 = novy.sig1 * gaussdev();
16   d_x2 = novy.sig2 * gaussdev();
17   delta_x1 = d_x1;
18   delta_x2 = d_x2;
19   novy.x1 = perturbation(novy.x1 + delta_x1);
20   novy.x2 = perturbation(novy.x2 + delta_x2);
```

Algoritmus 3.2 Časť funkcie kriz_mut() evolučnej stratégie bez použitia kovariancií. Uvedená je len časť, v ktorej sa vykonáva mutácia vznikajúceho nového chromozómu.

Ako vidíme, zanedbali sme uhol α. To je vlastne to isté, ako nahradenie uhla uhlom nula. Tým pádom výraz z algoritmu 3.1

```
d_x1 * cos(novy.alfa) – d_x2 * sin(novy.alfa)
```

po dosadení novy.alfa = 0 nadobúda hodnotu d_x1. Podobne, výraz

```
d_x1 * sin(novy.alfa) + d_x2 * cos(novy.alfa)
```

po dosadení novy.alfa = 0 nadobúda hodnotu d_x2.

Verzia 3 : Bez použitia kríženia, s použitím kovariancií

Tretia verzia evolučnej stratégie využíva kovarianciu medzi premennými, ale nepoužíva kríženie. Časť funkcie kriz_mut(), v ktorej chromozóm dedí hodnoty premenných od náhodne zvoleného chromozómu z populácie rodičov, je uvedená v algoritme 3.3. Jedine touto časťou sa líši od algoritmu 3.1. Teda funkciu kriz_mut() tejto verzie evolučnej stratégie dostaneme nahradením riadkov 7 – 13 v algoritme 3.1 algoritmom 3.3.

```
7    r = random(CNT_P);

 9   novy.x1   = P[r].x1;
 10  novy.x2   = P[r].x2;
 11  novy.sig1 = P[r].sig1;
 12  novy.sig2 = P[r].sig2;
 13  novy.alfa  = P[r].alfa;
```

Algoritmus 3.3 Časť funkcie kriz_mut() evolučnej stratégie s kovarianciami, bez použitia kríženia. Uvedená časť, v ktorej nový chromozóm dedí hodnoty premenných od náhodne zvoleného chrozómu z rodičovskej populácie.

Na rozdiel od algoritmu 3.1 je z rodičovskej populácie vybratý len jediný chromozóm. Teda kríženie odpadá a priamo hodnoty premenných vybratého rodiča sú zmutované a preberá ich novovytváraný chromozóm.

V tabuľke 3.1 je pre každú verziu evolučnej stratégie priemerný počet generácií, ktorý bol potrebný na dosiahnutie funkčnej hodnoty Rosenbrockovej funkcie menšej ako 10^{-6}.

Algoritmus	Priemerný počet generácií
Verzia 1 Evolučná stratégia s krížením a kovarianciami	49,36
Verzia 2 Evolučná stratégia s krížením bez kovariancií	725,98
Verzia 3 Evolučná stratégia bez kríženia s kovarianciami	38,54

Tabuľka 3.1 Priemerný počet generácií, ktorý bol potrebný, aby najlepšia nákdená funkčná hodnota bola nižšia ako $10^{=6}$. Každý program bol spustený 100 krát.

Ako vidíme, potvrdila sa predpoklad o tom, že použitie kovariancií, teda nasmerovanie elipsoidu rovnakej pravdepodobnosti distribúcie premenných smerom ku hľadanému optimu, výrazne urýchlilo hľadanie globálneho minima.

Trochu prekvapivý je záver, že po odstránení kríženia sa rýchlosť algoritmu ešte trochu (aj keď nie až tak výrazne) zvýšila.