Subject
Date

۹۷/۱۱/۳۰

الگوریتم PSO

(Particle Swarm optimization)

ابتدا در مورد رندوم سرچ صحبت می‌کنیم،

Random Search:

مسائل بهینه سازی

مسائلی که به دنبال max و یا min هستیم

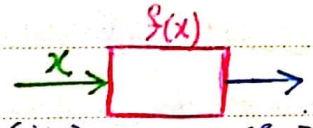
الگوریتم‌های هوش جمعی و تکامل برای مسائل بهینه سازی به کار می‌روند

مثال: پیدا کردن وزن‌های بهینه برای شبکه‌های عصبی

فرض کرده بر این تابع $f(x)$ می‌خواهیم max تابع را پیدا کنیم

به ازاء کدام x مقدار $f(x)$ بیشترین مقدار

(یا کمترین مقدار) را دارد (زمان البته بسته به مسئله)

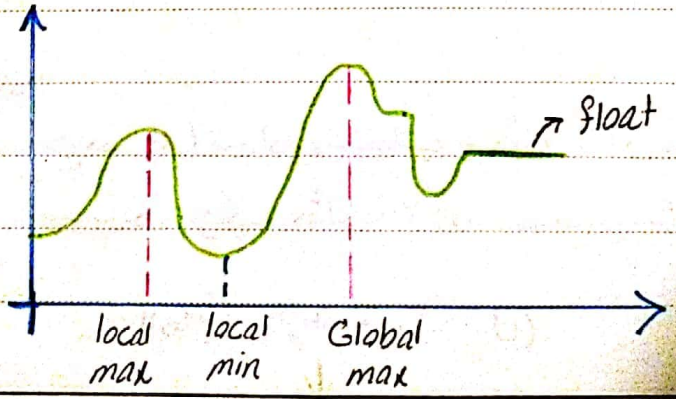


استفاده می‌کنیم که الگوریتم پیچیده است و

از روش‌های معمول می‌توان استفاده کرد و یا دقیق‌تر از تابع حذر کنیم)

xهای تصادفی می‌دهیم و $f(x)$ هایش را حاصل کرده و از بین آن‌ها بیشترین

و یا کمترین را پیدا کرده البته جواب قطعی نیست بلکه local را می‌توان یافت.



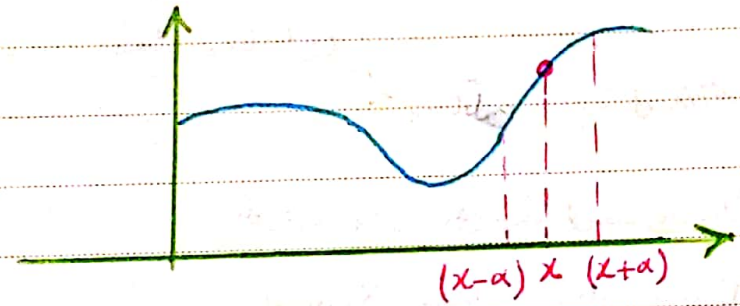
PqPCO

Subject
Date

این Random Search بود.
معایب:
- گیر کردن در float
- گیر کردن در local
- (جواب local را بجای جواب Global برگرداند)

الگوریتم تپه نوردی:

یک نقطه به صورت تصادفی پیدا کرده و بعد همسایه‌ها (مزرعه‌دان) را پیدا کرده، سپس از بین مزرعه‌دان کدام بهتر بود انتخاب می‌شود، در حالی که مزرعه‌دان از والد بهتر نباشد متوقف می‌شود.



معایب: حریصانه است

الگوریتم SA:

مانند الگوریتم تپه نوردی است با این تفاوت که یکی از مزرعه‌دان به صورت تصادفی انتخاب می‌شود اگر بهتر بود به سمتش رفته وگرنه با یک شانسی به سمتش حرکت کرده (این شانس به مرور کم می‌شود)

اگر در local Bim Search اشتراک گذاری اطلاعات داشت: می‌شد الگوریتم جمعیتی

الگوریتم جمعیتی:

چند نقطه را تصادفی انتخاب می‌کنند که این نقاط در کنار هم یک جمعیت می‌شوند. از یک دیدگاه به ۲ گروه تقسیم می‌شوند

۱) الهام گرفته از طبیعت (تکامل) → داروین → GA
۲) جمعیتی → Swarm

Subject
Date

• اولین الگوریتم هوش جمعی PSO

گام اول طراحی ← Particle Solution

که با بردار نشان می‌دهیم (البته بستگی به مسئله دارد)

تعیین ساختار پارتیکل و ایجاد جمعیت اولیه

مثال:

$$\min f(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2$$

$$-5 \leq x_i \leq 5$$

$$i = 1, 2, 3$$

Position | x_1 | x_2 | x_3 |

$-5 \leq x_i \leq 5$

$i = 1, 2, 3$

جمعیت اولیه ← PopSize = 10

حال ما از این 3x1 ها ایجاد کنیم و به هر کدام عدد تصادفی در بازه داده شده نسبت داده (این 10 جمعیت را کاملاً تصادفی مقداردهی کردیم)

در الگوریتم PSO برخلاف الگوریتم ژنتیک هر ذره با ۳ فاکتور نشان داده می‌شود

برای هر ذره ۳ بردار تشکیل داده

1- Position
2- سرعت
3- PBest

x

y

Pb

Subject
Date

Pbest در لحظه اول همان x است.

سرعت را در لحظه اول برابر صفر یا مقداری نزدیک به صفر در نظر می‌گیرند.

سرعت در واقع خلاصه PSO است زیرا:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

GBest → LBest Personal Best

- PSO دو توپولوژی
  - ۱) توپولوژی GBest (star) ← Global Best یک داریم ← ممکن است به بهینه محلی همگرا شود (ایراد)
  - ۲) توپولوژی LBest (ring) ← Global Best چند داریم

* PSO پایه برای حل مسائل پیوسته است.

مثال) مسأله TSP را با PSO حل کنید؟

جواب: TSP یک مسأله گسسته است و PSO پایه برای حل مسائل پیوسته. پس قابل حل نیست.

* گام‌های اصلی PSO:

۱) تعیین ساختار Particle و ایجاد جمعیت اولیه

۲) تعیین Fitness

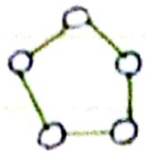
۳) تعیین Pbest و gbest

۴) update v و update x

PAPCO

Subject
Date

- توپولوژی LBest
ring ⟵ حلقه



همگی باهم در ارتباط نیستند و یک محدوده همسایگی داریم. یک سایز همسایگی تعریف می‌کنیم. بهترین محدوده lBest ، Personal Best می‌شود
Global Best ، مثل gbest داریم.

- در الگوریتم تکاملی ۲ مفهوم اصلی وجود دارد

۱) Exploration ⟵ (یعنی فضا را خوب ببینیم) مفهوم تنوع را در بر دارد

۲) Exploitation ⟵ (یعنی محدوده مورد نظر را خوب بگردم) با گام‌های کوچک کار می‌کند

GBest:
- exploration کمی دارد (تنوع)
- exploitation بهتری دارد

LBest:
- exploration بهتری دارد (چون همسایگی را دارم و کم‌کم همسایگان دارند)
- exploitation کمی دارد.

Subject
Date

۹۷/۱۲/۷

• پارامترهای آزاد:

- Swarm size (اندازه جمعیت):

هرچه بیشتر باشد (ضمنا زمان بیشتری را اشغال کرده) و درنتیجه exploration بیشتری داریم ، محاسبات fitness بیشتر شده و هزینه بالا می رود.
اندازه جمعیت به نوع مساله بستگی دارد (اگر پیچیده و سخت باشد جمعیت را بیشتر گرفته)

- اندازه همسایگی در Lbest :

هرچه اندازه همسایگی بیشتر باشد ← exploitation بالا می رود

اگر همسایگی کم باشد دیر تر همگرا شده و exploration بالاتری داریم

! بهتر است ابتدا سایز همسایگی را کم گرفت که exploration بالا برود و بعد سایز را افزایش دهیم که exploitation افزایش یابد

- تعداد تکرارها:

هرچه کمتر باشد
↙ قبل از رسیدن به پاسخ الگوریتم را پایان داده و هرچه تا پایان یافته

هرچه بیشتر باشد
↙ هزینه بالا می رود

- C_1 و C_2

بسته به مساله در نظر گرفته می شود و معمولا" برابر می گیرند

اگر $C_1=0$ ← Gbest حذف می شود ← exploitation را داریم

اگر $C_2=0$ ← Pbest حذف می شود ← exploration را داریم

P4PCO

Subject
Date

این الگوریتم بر مبنای Exploration و Exploitation هستند

w به بردار سرعت اضافه شده

۹۸/۱/۲۷

	الگوریتم PSO A		الگوریتم B	
f	min	4×10^{-8}	min	5×10^{-2}
حواب بهینه min(o)	std	10	std	10^{-2}
کم انحراف معیار			✓	

مثال) کدام الگوریتم بهتر است؟

جواب:

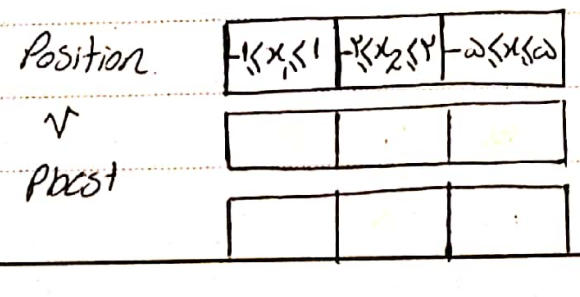
چون جواب بهینه کمینه سازی است. هرچه جواب به صفر نزدیکتر باشد بهتر است در این جا الگوریتم PSO کمتر است ولی با توجه به انحراف معیار (زمانی که انحراف معیار زیاد باشد به آن الگوریتم نمی‌توان اعتماد کرد ــ انحراف معیار یعنی جواب‌ها روی یک حول و حوش پخش می‌کند) پس الگوریتم B بهتر است.

مثال) PSO را برای $\min f(x_1,x_2,x_3)$ تعریف کنید.

$$-1 \le x_1 \le 1$$
$$-2 \le x_2 \le 2$$
$$-5 \le x_3 \le 5$$

۱) جمعیت اولیه و تعریف

هر ذره یک بردار دارد:



nvar = 3

ns = 10

$c_1, c_2 = 4$

maxgen = 100

ct

۷ ← در ابتدا صفر و یا نزدیک به صفر

Pbest ← در لحظه اول برابر Position

Gbest ← در لحظه اول برابر Pbest

آپدیت v

آپدیت x

(تپولوژی را انتخاب کرده اگر LBest بود ، اندازه همسایگی را در نظر گرفت)

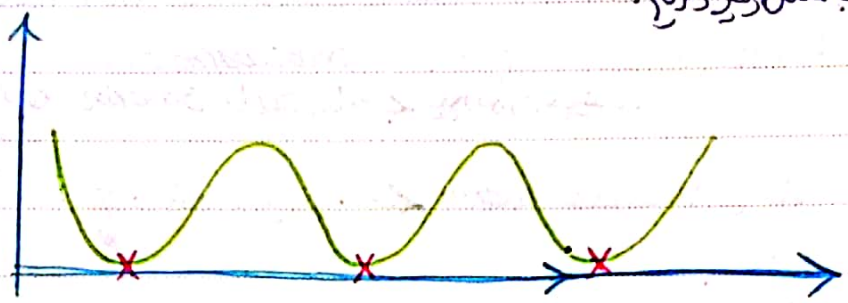
co

Subject
Date

Niching PSO :

ازخانواده PSO است .

مزمن کرده تابعی به شکل زیر داریم :



می خواهیم تمام min های این تابع را به دست آوریم ، در مثل (PSO) به دنبال یک min بودیم و در این جا چندین min (به این مسائل ، مسائل multi solution گویند)

خود PSO به تنهایی قادر به پیدا کردن چند نقطه min نیست زیرا به دنبال global best است و همه ذرات را به آن نقطه سوق می دهد پس با آن نمی توان چند min را یافت .
پس باید PSO را طوری تغییر دهیم که بتواند تمام راه حل های ممکن را پیدا کند .
روش پیشنهادی ، روش Niching PSO می باشد .

در این روش

۱) ایجاد جمعیت اولیه تصادفی با نام main swarm

در روش PSO می توان جواب را یافت زیرا همه ذرات به سمت یک پاسخ می روند
بنابراین در این روش ، هر ذره از تأثیر جمعیت به دنبال پاسخ می گردد .
پس بخش سوشال (آپدیت سرعت سوشال) را از جمعیت حذف می کنیم

Subject
Date

نحوه تشکیل SubSwarm

ذراتی را در لحظات مختلف بررسی می‌کنیم بعد از تعدادی تکرار مشاهده می‌شود که fitness برخی ذرات ثابت شده و زیاد تغییر نمی‌کند (حتی با تکرارهای بیشتر هم تغییر نمی‌کند) پس احتمالاً موقعیتش زیاد تغییر نکرده که fitness اش ثابت است، پس آن را به عنوان sub swarm در نظر می‌گیریم. آن ذره را با نزدیکترین همسایه‌اش (در واقع همسایه‌ای که از لحاظ اقلیدسی با آن نزدیکتر باشد) برای تشکیل subswarm در نظر می‌گیرند. و از main swarm جدا حذف می‌شود.

! به تعداد پاسخ‌ها ← SubSwarm تشکیل می‌شود

$S = S \setminus S^*$

S: main swarm
S^*: sub swarm

حال در subswarm هر دو ذره به دنبال یک جواب می‌گردند پس از رابطه کامل PSO استفاده می‌کنیم (در ابتدا دو ذره هستند و با مرور بیشتر شدن بیشتر می‌شود)

هر subswarm ای برای خودش یک شعاع دارد.
یک ذره در اثر تکرارهای وارد شعاع subswarm می‌شود.

* شعاع ← فاصله هر ذره با gbest در نظر گرفته، بیشترین فاصله می‌شود شعاع

فاصله ذرات داخل subswarm با gbest در آن لحظه را به دست آورده بزرگترین فاصله می‌شود شعاع subswarm

ذره‌ای که در اثر تکرارها وارد شعاع subswarm شود به عنوان جواب وارد subswarm می‌شود و از main swarm حذف می‌شود.

* هر تکرار مقدار gbest تغییر می‌کند

* تا وقتی که در هر لحظه gbest آپدیت می‌شود پس شعاع هم آپدیت می‌شود

P4PCO

Subject
Date

اگر هر Subswarm حول وحوش یک مقدار باشد، بهتر است آن‌ها را با هم merg کنیم

* شرط merg کردن:

۱) فاصله gbest هایشان را حساب کرده، اگر از مجموع شعاع Subswarm ها کمتر بود آن‌ها را با هم merg می‌کند.

بعد از تعدادی تکرار دوباره استفاده کنند یعنی شعاع صفر

۲) شرط دیگر اگر فاصله gbest از یک حدی کمتر بود آن زمان آن‌ها را با هم merg کن

* نحوه بخش کردن جمعیت اولیه خیلی مهم است

الگوریتم BSO:

۱) ایجاد جمعیت اولیه
۲) کروسوزوم را sort کرده
۳) به هر ذره براساس ارزشش (fitness) ۲ پارامتر اختصاص می‌دهند

- μ ← نخ اطلاعات‌دهی
- λ ← نخ اطلاعات‌گیری

* ذره خوب μ بالا و λ پایین دارد

λ و μ ارتباط مستقیمی با fitness دارند (که یعنی آن موجودیت می‌خواهد چقدر اطلاعات بدهد چقدر اطلاعات بگیرد)

در BSO ← به هرزن یک SIV گوید
← به هر موجودیت یک HSI گوید

PqPCO

Subject
Date

۲ عملگر اصلی BBO

1) migration
2) mutation

Sharing اطلاعات را انجام می‌دهد

موجودیت H_i از یک موجودیت به خاطر H_j یک اطلاعاتی را می‌گیرد

$H_i(SIV) \leftarrow H_j(SIV)$

برای مسائل non-separation کاربرد دارد یعنی متغیرها

migration کپی است ← Copy ← Replace را دارد.

هست نحوه اطلاعات گیری و اطلاعات دهی ژن‌ها

اگر متغیرها مستقل از هم باشند ← مشکل پیش می‌آید

اگر متغیرها مستقل از هم نباشند ← مشکل پیش می‌آید و خوب جواب می‌دهد

mutation ← وقوع، خروج از local ← امکان کمک به ژن‌ها برای بهتر شدن

P4PCO

Subject
Date
۹۷/۲/۳

○ الگوریتم ICA : فضای جستجو را سگمنت می‌کند

- ایده گرفته شده از روابط اجتماعی انسان‌ها، مستعمره‌ها و مستعمره‌گرها سرچشمه گرفته

- هدف استثمار ← استفاده از منابع

- به هر موجودیت ← کشور (Country) گفته می‌شود { - کشورهای بهتر ← استعمارگر / - باقی کشورها ← مستعمره }

- در واقع رقابت استعماری وجود دارد و هر امپراطوری برای بقا خودش عمل می‌کند

- بر پایه ۲ عملگر { - عملگر جذب / - عملگر انقلاب }

۱) ایجاد جمعیت اولیه (به هر ژن گفته می‌شود P)

خصوصیت

$$Country = [P_1, P_2, \dots, P_{Nvar}]$$

← موجودیت ← خصوصیات آن موجودیت

۲) ارزش هر موجودیت را محاسبه کرده (تعیین فیتنس به مسأله بستگی دارد)

۳) به هر کشور یک حالت Power نسبت می‌دهیم

فرض کرده تعداد اعضا ۵۰ است و تعداد استعمارگرها ۱۰ تا جمعیت را تولید کرده، ارزش (f.f) هر یک را حساب کرده، ۱۰ تای بهتر را استعمارگر و حال ۴۰ تای باقی‌مانده را مستعمره می‌گذاریم حال باید بگوییم به هر استعمارگر چه تعداد مستعمره می‌رسد برای این کار باید Power هر یک را حساب کنیم

جهت مشخص بودن این که به هر استعمارگر چند مستعمره می‌رسد برای محاسبه Power، ابتدا Cost ها را باید نرمال کرد آن هم از فرمول زیر

Cost آن موجودیت — ماکسیمم Cost

$$C_n = C_n^{\prime} - \max_i \{C_i\}$$

و یا

$$C_n = \max_i \{C_i\} - C_n$$

Subject
Date

حال با به دست آوردن Cost نرمال شده از رابطه زیر می‌توان Power را محاسبه کرد

$$P_n = \left| \frac{C_n}{\sum_{i=1}^{N} C_i} \right|$$

N: تعداد استعمارگرها

حال Power را در یک کلون ضرب می‌کند

$$NC_n = round\{P_n \cdot (N_{Col})\}$$

N_{Col}: تعداد کلون‌ها = تعداد مستعمره‌ها

NC_n: تعداد مستعمره‌هایی که هر استعمارگر می‌گیرد

۲ عملگر دارد:

۱) عملگر جذب (cross over): استعمارگر مستعمره را به سمت خود جذب می‌کند هر استعمارگری دوست دارد مستعمره خودش را شبیه به خود کند (طرز لباس پوشیدن، زبان و ...)

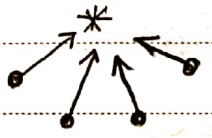
۲) عملگر انقلاب (mutation): مستعمره از استعمارگر می‌خواهد جدا شود یکسری ویژگی مستعمره را تغییر می‌دهد

۹۷/۲/۱۷

• عملگر جذب:

استعمارگر ← مستعمره را به سمت خود می‌کشد

از این اصطلاح برای در مورد همسایگی نزدیکی نیست



$$x + \beta(Im - Col)$$

β: فاصله استعمارگر

Im: خانه مستعمره

دلیل این که مستعمره را از اعضا نمی‌دارد نزدیکی نزدیکی نیست.

به این دلیل نزدیکی و دوری مهم نیست S کل فضا را در نظر گرفته لزوماً S country در است

مجبور نیست و در واقع یک شانس به دیگران می‌دهد

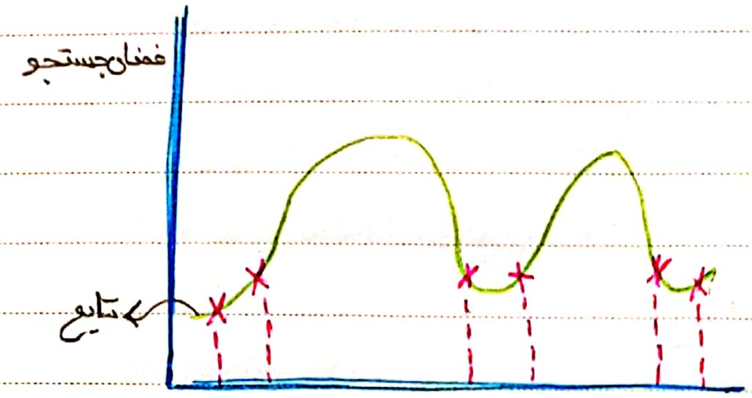
و اگر همسایه‌ها را کلون کند پس exploration را متوقف می‌کند و باعث همگرایی زودرس می‌شود

PAPCO

Subject
Date

۹۷/۲/۱۷

* ممکن است یک مستعمره از استعمارگر قوی‌تر شود، در آن صورت جای مستعمره و استعمارگر تغییر می‌کند



به صورت رندوم یکسری نقاط را انتخاب کرده و بعد یکسری نقاط از بقیه بهتر هستند (شانس پیدا کردن Global Best) بر حسب خوب بودن ارزششان یکسری از جمعیت را به آن‌ها نسبت می‌دهد (هر کدام کلونی که بهتر است آن‌جا نیروی بیشتری می‌دهم)

فضا را سگمنت می‌کند و Sub population هایی را درست می‌کند
و در آن Sub ها سرچ را انجام می‌دهد

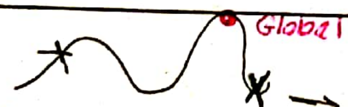
• تابع ارزش (f.f)

هزینه‌ی آن موجودیت استعمارگر = هزینه‌ی استعمارگر (خودش) + ضریبی از مستعمراتش (میانگین cost مستعمراتش)

به طور کلی:
۱) جمعیت اولیه ۲) نسبت دادن Power ۳) مشخص کردن کلونی‌ها (چه تعداد مستعمره به هر استعمارگر برسد)
۴) عملگرهای جذب و انقلاب (موجودیت‌ها را تغییر داده) ۵) اگر یک مستعمره بهتر بود با استعمارگر جابجا شود
۶) حال ارزش کلی استعمارگرها را به دست آورده

حال وضعیت کلونی‌ها مشخص شده که هزینه‌ی واقعی یک استعمارگر را محاسبه کرده
این (هر بار) را در کد تکرار می‌کنند زیرا هر دور ممکن است جای را خوب نگشته باشد حذف شود
← حال در هر بار (در هر دور) ضعیف‌ترین مستعمره‌ی ضعیف‌ترین استعمارگر را شناسایی کرده

می‌خواهیم بررسی کنیم این ضعیف‌ترین به کدام استعمارگر تعلق می‌گیرد حال فرض کنید
راه کلی کنیم: به قوی‌ترین و یا نزدیک‌ترین ← ممکن در local گیر بیفتد؛ زیرا ممکن است
ضعیف‌ترین جواب خوب را به ما می‌دهد



(به نظر خوب نیست اما به واقع به Global نزدیک‌تر است و ما را زودتر به جواب می‌رساند پس قوی‌ترین معنی ندارد)

Subject
Date

با یک شانسی (همان تصادفی بودن) Cost ها را دوباره جایگذاری کند Power را دوباره محاسبه کرده و در یک آرایه قرار می دهد.

$$R = [r_1, r_2, \dots, r_N]$$

حال عدد تصادفی ایجاد می کند

$$P = [P_1, P_2, \dots, P_N]$$

بعد حاصل اختلاف Power و عدد تصادفی را محاسبه کرده هرکدام که حاصل بیشتری داد، مفیدترین مسئوله مفیدترین استعمارگر را به آن می دهد.

$$D = P - R = [P_1, P_2, \dots, P_N]$$

$$= [P_{P_1} - r_1, P_{P_2} - r_2, \dots, P_{P_N} - r_N]$$

مفیدترین مسئوله
مفیدترین استعمارگر

یک عدد تصادفی ایجاد می کند و سپس حاصل اختلاف Power و عدد تصادفی هریک را به دست آورده، هرکدام که حاصل بیشتری داشت (عدد بزرگتری شود) به آن نسبت می دهد به این ترتیب کم کم کلونی ها کم می شود و در نهایت یک امپراتوری باقی می ماند

در کلاس های آخر هم مفیدها حذف می شوند زیرا ما می خواهیم به یک جواب بهینه برسیم و با دنبال یک جواب هستیم و چون با یک نقطه همگرا می شود

* از معایب این الگوریتم همگرایی زودرس *

Subject
Date

الگوریتم HPSO :

ابتدا به یک دوره اجمالی به الگوریتم PSO داریم:

در هر تکرار، سرعت و موقعیت هر ذره با دو معادله زیر به روز می‌شوند:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_{2j}\left[\hat{y}_j(t) - x_{ij}(t)\right]$$

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

$\hat{y}$ (بهترین موقعیتی که تاکنون توسط کل ذرات یافت شده است) gbest

c_1 و c_2 : (ثابت‌های مثبت) ثابت اند

r_1 و r_2 : عدد تصادفی بین (0 و 1) هستند

الگوریتم PSO تکرار یک فضای جستجو چند بعدی را با یک دسته از افراد، که به عنوان Swarm خوانده می‌شوند، بررسی می‌کند

هر ذره (پارتیکل) را می‌توان با یک گره از بردار (vector) تعریف کرد:

$$(x_i, v_i, \bar{y}_i)$$

$x_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$: Position of ith Particle

$v_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$: Velocity of ith particle

$y_i = (y_{i1}, y_{i2}, \ldots, y_{iD})$: Personal best position found by the ith Particle

Subject
Date

: Neighborhood Topologies

عملکرد PSO به شدت به ساختار شبکه اجتماعی بستگی دارد

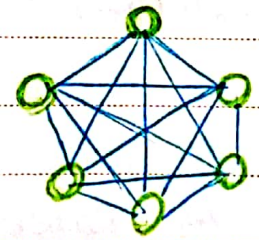
Star ← یک gbest داشتیم - exploration بالا

ring ← exploitation بالاست

wheel

(star) gbest

- تمام ذرات متصل هستند
- هر ذره می تواند با هر ذره دیگر ارتباط داشته باشد (برقرار کند)
- هر ذره به سمت بهترین راه حل که توسط کل ذرات یافت می شود حرکت می کند.
- سریعتر از سایر ساختارهای شبکه همگرا می شود
- حساس به افتادن در local min ضعف اصلی در gbest PSO
- برای unimodal problems بهترین Performs را دارد.
که برای مشکلات یکپارچه بهترین کار را می کند



ring (lbest)

- هر ذره با همسایگان نزدیک خود ارتباط برقرار می کند.
هر ذره تلاش می کند تا بهترین همسایگان را با بهترین همسایگان خود را با تقلید به بهترین راه حل که در محله پیدا می شود، تقلید کند
همسایگی همپوشانی دارد.
۱) تسهیل می کند تبادل اطلاعات بین همسایگان محله ها

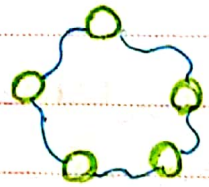
Subject
Date

۲) در نهایت ، همگرایی به یک راه‌حل واحد

جداکردن

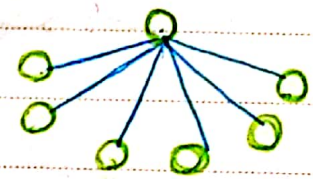
همکرد

* ساختار Ring ، Performance بهتری از لحاظ کیفیت راه‌حل‌های موجود برای مشکلات چندوجهی ، نسبت به ساختار Star (gbest) دارد.



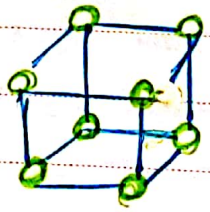
wheel

انتشار راه‌حل‌های خوب را از طریق Swarm کاهش می‌دهد



Von Neumann

نسبت به ساختارهای دیگر، در حل a large number of problems بهتر عمل کرده



الگوریتم HPSO :

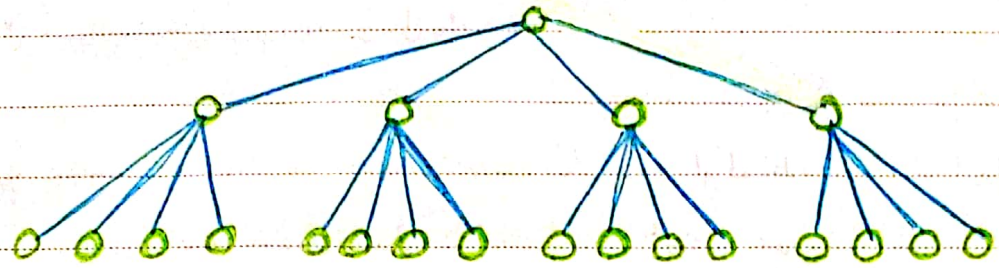
Hierarchical Particle Swarm optimizer

- تمام ذرات در یک درخت مرتب شده‌اند که ساختار درخت از نوع سلسله مراتبی است به طوری که هرگره درخت حاوی دقیقاً یک پارتیکل (ذره) است.

این سلسله مراتب ساختار neighborhood را تعریف می‌کند

هر پارتیکل در سلسله مراتب به خود و والدینش همسایه است

P4PCO



$h=3$
$d=4$
$m=21$

به طور مثال در این درخت ...

h ← ارتفاع درخت (عمق درخت)

d ← تعداد فرزند هر گره

m ← تعداد کل گره‌های درخت

از بالا به پایین سرچ می‌کند، از سطح اول شروع به مقایسه می‌کند

در هر تکرار پس از ارزیابی عملکرد تابع هدف در موقعیت واقعی ذرات

قبل از به‌روزرسانی سرعت و تعیین موقعیت‌های جدید در فضای جستجو، موقعیت‌های جدید ذرات در سلسله مراتب به صورت زیر تعیین می‌شود:

برای هر ذره i در گره درخت، بهترین راه حل خود را با بهترین راه حل که توسط ذرات گره‌های فرزند پیدا شده مقایسه می‌کند (یعنی با فرزندانش مقایسه می‌شود) اگر فرزند بهتر بود فرزند و پدر جای خود را با هم جابجا می‌کنند.

در هر تکرار این ساختار درخت تغییر می‌کند به این صورت که پدر با فرزند اشتی مقایسه می‌شود بهترین فرزند که از پدر هم بهتر بود جای خود را با پدر تغییر می‌دهد)

exploration را آرام آرام بالا می‌برد

(exploration , exploitation به خاطر این که یه یله بالا می‌رود، خوب دیده می‌شوند)

این مقایسه‌ها از ابتدای سلسله مراتب (از اول درخت) آغاز می‌شود.

Subject
Date

یک ذره می‌تواند در سلسله مراتب در جایگاه LEVEL باقی بیاید اما برای بالارفتن حداکثر یک سطح می‌تواند بالا برود.

repeat
  for each Particle i do
    Evaluate objective function $f(x_i)$ and update Personal Best
    if $(f(x_i) < f(y_i))$ then
      $y_i = x_i$
    end if
  end for
  for each Particle i do
    Determine the best successor $j = \arg\min_{k \in S(i)} \{f(y_k)\}$ /* S(i) are the successors of i */
    if $(f(y_j) < f(y_i))$ then
      Swap Particles i and j
    end if
  end for
  for each Particle i do
    update the velocity v_i in each dimension d:
    $v_{id} = w v_{id} + c_1 r_1 (y_{i,d} - x_{i,d}) + c_2 r_2 (y_{n(i),d} - x_{i,d})$
    move the particle: $x_{i,d} = x_{i,d} + v_{i,d}$
  end for
until Stopping Criterion is met

Subject
Date

Evolutionary Dynamic optimization

انواع مساله را به ۲ گروه تقسیم می کنند

۱- استاتیک که تا الان استاتیک خواندیم ، یک تابع داریم و در یک بازه خاص به دنبال جواب بهینه هستیم

۲- داینامیک

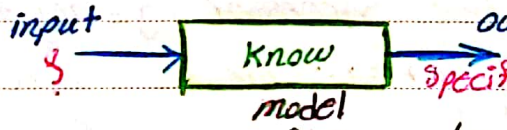
بهینه سازی
optimization

بهترین مقادیر برای object function ها

یا متن بهترین مقادیر برای متغیرهای یک مشکل خاص برای به حداقل رساندن یا به حداکثر رساندن یک تابع هدف

در یک مساله بهینه سازی

خروجی و مدل را می دانیم و به دنبال این هستیم که چگونه یا ورودی ها را مقدار دهیم که به خروجی مدنظر برسیم :



(مدل و خروجی موردنظر شناخته شده است و وظیفه یافتن ورودی (ها) منجر به این خروجی است)

✱ مسائل بهینه سازی را می توان به ۲ گروه تقسیم کرد :

(۱) Static problem

(۲) Dynamic problem

Subject
Date

Static Problem:

یک تابع داریم و در یک بازه خاص به دنبال جواب هستیم

نگران تغییر محیط نیستیم

فضای جستجو بسته → Ω

یک تابع کیفیت → $f: \Omega \to R$

نسبت مقایسه → $">" \in \{<, >\}$

وظیفه تعیین مجموعه‌ای از global optima ، $X \subseteq \Omega$ که تعریف می‌شود:

$$X = \{ x \in \Omega \mid \forall x' \in \Omega \; f(x) \geq f(x') \}$$

گاهی شناسایی یک راه‌حل $x \in X$ برای بهینه‌سازی موفق کافی است.

(DOP)

Dynamic Problem:

که وابستگی به زمان

بازه‌های زمانی داریم در آن بازه زمانی تابع fix زمانی که از آن تابع رد می‌شویم

شکل تابع تغییر می‌کند

نگران این هستیم که تابع، نقطه بهینه‌اش کجاست.

(دایناميک)

زندگی در زمین یک مزاحم همیشه در حال تغییر است و ارگانیسم‌ها باید با این تغییر سازگار و مقاوم شوند (تغییرات گاهی سریع‌اند، گاهی بزرگ و ...)

* الگوریتم بهینه‌سازی پویا دارای توابع هدف است که با گذشت زمان تغییر می‌کند

* تغییرات در objective function باعث تغییر در Position of optima و سرگشتگی فضای جستجو می‌شود

PAPCO

Subject
Date

- تغییر محیط می‌تواند براساس معیارهای زیر طبقه‌بندی شود:

۱- فرکانس تغییر
۲- شدت تغییر
۳- پیش‌بینی‌پذیری تغییرات
۴- طول دوره / دقت دوره

(Dynamic optimization Problem)

روش‌های تکاملی - حسابان (EC) ، ابزارهای خوب برای حل DOP ها
به‌دلیل الهام از سیستم‌های طبیعی ، که همیشه درمعرض تغییر محیط قرار گرفته اند
هستند

EDO :

- استفاده از الگوریتم‌های تکاملی (EAs) و تکنیک‌های مشابه برای حل DOP ها
بهینه‌سازی تکاملی در محیط‌های پویا (EDO) نامیده می‌شود.
(Evolutionary Dynamic optimization)

- PSO چاپو می‌خواهد یک global پیدا کند (باگذر زمان متوجه از بین رفتن و همه به یک سمت همگرا شدند)
  - در استاتیک به دنبال global هستیم
  - در داینامیک این سوال مطرح می‌شود آیا محیط تغییر کرده اگر جواب مثبت باشد باید جمعیت را رها کرد که به دنبال جواب بگردد

- تشخیص تغییرات:

در بسیاری از applications ، زمان وقوع تغییر به سیستم شناخته شده است . در غیر این صورت باید تغییرات شناسایی شوند.

۱) تشخیص تغییرات با ارزیابی مجدد detector های اختصاصی
۲) تشخیص تغییرات براساس رفتار الگوریتم

PAPCO

- تشخیص تغییرات محیط :
  ۱) detector ها
  ۲) رفتار الگوریتم

Subject
Date

• شایع‌ترین روش تشخیص تغییرات محیط: ← تشخیص تغییرات با ارزیابی دوباره راه‌حل‌ها

الگوریتم برخی از راه‌حل‌های خاص (detectors) را برای تشخیص تغییرات در Function Values (ارزش‌های عملکرد) و یا Feasibility (امکان‌پذیری) را دوباره ارزیابی می‌کند.

Re-evaluate → (دوباره ارزیابی کردن)

• Detectors

موجودیت‌هایی که می‌گویند محیط تغییر کرده است یا خیر

می‌توانند بخشی از جمعیت مانند: براساس بهترین راه‌حل‌های فعلی موجودیت‌ها

- memory based Sub-Population
- feasible Sub-Population

• اگر Detectors ها به اندازه‌کافی نباشد تغییرات را حس نمی‌کنیم

• اگر Detectors ها زیاد باشد Function call بالا می‌رود

به‌صورت جداگانه از فضای جستجو باشد:

یک نقطه ثابت (a fix point)

یک یا یک مجموعه از راه‌حل‌های تصادفی

...

• Detecting Change by Re-evaluatig Solutions:

تشخیص تغییرات با ارزیابی مجدد راه‌حل‌ها

(بهترین راه‌حل‌های کنونی)

- برای PSO:

در هر generation، موفقیت global optimum دوباره ارزیابی می‌شود. اگر یک تغییر وجود داشت به این معنی است که evaluted function تغییر کرده است.

PaPCO

Subject
Date

۲) یکسری نقاط تصادفی را در فضای جستجو ارزیابی می‌کند.

↳ براین فرض استوار است که اگر مکان بهترین سیستم دینامیک تغییر کند، مقدار بهترین آن هم تغییر می‌کند.

second_best

۳) gbest و دوم بهتر gbest را برای تعداد معینی از تکرار (iteration) را ارزیابی می‌کند

(memory based Sub-Population)

Explicit memory

هر موجودیت همراه خود یک حافظه دارد ، درون یک مخزن زیخه

اگر یکی از موجودیت‌های درون مخزن فیتنس اش تغییر کرده بود

یعنی محیط تغییر کرده . (بهترین ها را با جمعیت جدیدی رهسپار)

بهترین ها در نسل های بعد ذخیره و دوباره وارد جایهای میشوند

یا در صفحه ۲۳

Subject
Date

- Detectors را می‌توان به طور جداگانه از جمعیت جستجو کرد.

۱- فقط یک نقطه کاندیدا ← در فضای جستجو به طور منظم Re-evaluated می‌شود

۲- یک یا مجموعه‌ای از راه‌حل‌های تصادفی

که مجموعه‌ای از نقاط تصادفی برای Re-evaluated استفاده می‌شود

برای افزایش احتمال برای مقابله با تغییرات محلی که ممکن است

در همه جا قابل شناسایی نباشند

* تشخیص تغییرات با ارزیابی دوباره راه‌حل‌ها:

- نقاط قوت و ضعف

additional function evaluation

از آن‌جا که استفاده از detectors شامل ارزیابی تابع اضافی است، ممکن است

لازم باشد که تعداد بهینه‌ای از detectors را برای به حداکثر رساندن عملکرد الگوریتم

شناسایی کنیم.

- استفاده از تعداد کمی از detectors ها:

✓ اجتناب از تحت تأثیر قرار گرفتن ارزیابی‌های اضافی

✗ عدم تضمین شناسایی تغییرات

- استفاده از تعداد زیادی از detectors ها:

✗ بالا رفتن مزاحمان تابع

✓ تشخیص ۱۰۰٪

* تشخیص تغییرات براساس رفتار الگوریتم:

نقاط قوت و ضعف:

✓ نیازی به ارزیابی عملکرد اضافی ندارد

✗ به دلیل عدم استفاده از detector اختصاصی، هیچ تضمینی وجود ندارد که

تغییرات شناسایی شود.

PqPCO

Subject
Date

یکی دیگر از ضعف های احتمالی این است که

بعضی از روش های تشخیص تغییر نیاز مند الگوریتم خاص باشند (۲۸)

تا این جا روش های تشخیص تغییرات را در محیط بررسی کردیم حال به بررسی EDO می پردازیم

EDO :

حفظ تنوع :

EAs می خواهند از مشکل همگرایی زودرس رنج ببرند (یعنی جستجو را در یک منطقه خاص از فضای راه حل انجام می دهند)

← درحالی که همه ژنوتیپ ها بسیار شبیه هم اند.

همگرایی زودرس با کاهش تنوع مرتبط است

از دست رفتن تنوع می تواند EA را در نتیجه عنر مطلوب حفظ کند.

در مسائل دینامیک هم است که از همگرایی زودرس جلوگیری شود و

حفظ تنوع یک مسأله حیاتی است زیرا EA باید فضای جستجو را

به خوبی بررسی کند

یک راه حل : با توجه به هر تغییر رسیده شده از مسئله بهینه ساز جدید باید حل شود

• برخی از معایب :

(زمان مجاز یافتن جواب کوتاه باشد)

- زمان های سخگویی برای پیدا کردن بهینه جدید ممکن است کوتاه باشد

- اطلاعات از دست رفته از جستجوی قبلی:

زمانی که بهینه جدید نزدیک به قبلی است جستجو را می توان به

محدوده (همسایگی) بهترین بهینه قبلی محدود کرد، دانش (اطلاعات)

در مورد فضای جستجو قبلی می تواند بسیار مفید باشد.

PAPCO

Subject
Date

• با توجه به هگراین ، رویکرد هایی که با محیط های داینامیک برخورد می کنند را می توان به چهار دسته تقسیم کرد:

1) ایجاد تنوع پس از یک تغییر
2) حفظ تنوع در طول اجرا
3) رویکردهای مبتنی بر حافظه
4) رویکردهای حذف گانه

1) ایجاد تنوع پس از یک تغییر:

یک EA به طور معمول اجرا می شود ، اما هنگامی که یک تغییر شناسایی شد ، اقدامات خاصی برای افزایش تنوع انجام می شود.

(روش) - الگوریتم 1 : ایجاد تنوع پس از یک تغییر:

1) initialize (تشکیل جمعیت اولیه)
2) برای هر نسل اعمال زیر را انجام می دهد:

a) Evaluate (ارزیابی) :
↓
هر عضو از جمعیت را ارزیابی می کند

b) check for change:
تشخیص تغییرات توسط نظارت کردن بر علائم احتمالی تغییرات : کاهش ارزش بهترین fitness و Re-evaluation (ارزیابی مجدد) راه حل های قبلی (قدیمی)

c) افزایش تنوع :

اگر تغییر رخ داده بود ، تنوع جمعیت را با mutation (اندازه های بزرگ) یا انتقال افراد افزایش دهید

Subject
Date

d) تولید مجدد:

یک جمعیت جدید با استفاده از mutation / learning / adaptation rate
ایجاد کنید — سازگاری تنظیم شده — آموزش — نرخ جهش

e) بازگشت به مرحله 2a

- یک مکانیزم ساده‌ای است که در آن جمعیتی از Swarm و یا کل Swarm مجدداً به صورت تصادفی بیش از یک تشخیص یک تغییر، متنوع می‌شود
- جمعیتی بیش از حد VLS

• مزایا:

- لازم نیست که همیشه تلاش خود را برای حفظ متنوع انجام دهیم
تمرکز کامل در فرآیند جستجو دارند و زمانی که تغییر را حس کنند
واکنش نشان می‌دهند (حفظ متنوع)
- در حل مسائل با تغییرات بسیار مکرر که تغییرات کوچک و متوسط
هستند، خوب است.

(زیرا جمعیتی در اطراف یک بهینه نزدیک به یک نوع جستجوی
محلی است که برای مشاهده مکان‌های نزدیک این optimum
مفید است بنابراین اگر optimum به مکان‌های نزدیک
حرکت کند ممکن است در این سوی)

• معایب:

- وابستگی به این که آیا تغییرات شناخته شده است یا به راحتی تشخیص داده می‌شود یا خیر
- دشواری در شناسایی اندازه درست جهش (mutation)
- اطلاعات کمی از جستجوی قبلی حفظ می‌کند

PAPCO