علوم شناختی(رویکرد تکاملی)

گزارشکار پیاده سازی

استاد:

دکتر سعید ستایشی

تهیه کننده:

الهام اسمعیل نیا شیروانی

بهمن ماه ۹۶

فهرست

مقدمه:

نظریه انتخاب طبیعی که توسط چارلز داروین در ۱۸۵۹ ارایه شد، مسیر فکری ما را در مورد زیست شناسی متحول کرد. انتخاب طبیعی بیان می کند که خصوصیات سازگار کننده به حیوانی که مالک این خصوصیات باشد، این توانایی را می دهند تا بقا یابد و این خصوصیات را به نسل های آتی خود انتقال دهند. در این دیدگاه، محیط به عنوان یک عامل انتخاب کننده در نظر گرفته می شود که از میان صفات متنوعی که مقاصد کارکردی دارند، صفاتی خاص را انتخاب می کند. رویکرد تکاملی را می توان یک شیوه ی کاملا عمومی در نظر گرفت و آن را برای شرح و توصیف پدیده های خارج از حوزه ی زیست شناسی نیز مورد استفاده قرار داد. حوزه ی روانشناسی تکاملی، نظریه ی انتخاب را برای فرایندهای ذهنی انسان در نظر گرفته و بکار می برد. سعی آن بر این است که روشن کند که نیروهای انتخاب گر که بر اجداد ما عمل کرده اند، کدام اند و چگونه این نیروها، ساختارهای شناختی که اکنون واجد آن هستیم را شکل داده اند. دانشمندان حوزه ی روان شناسی تکاملی رویکرد مدولار نسبت به ذهن را می پذیرند. در این مورد، هر مدل با ظرفیت های شناختی" مساعد" خودش تناسب پیدا کرده است که توسط اجداد ما برای حل مشکل معینی به طور موفقیت آمیزی استفاده شده اند. نظریه های تکاملی، این نظر و این علت را برای نتایج تجربی در سطح وسیعی از پدیده ها و ظرفیت ها مطرح می کنند. ظرفیت هایی نظیر طبقه بندی حافظه، برهان منطقی و برهان احتمالات، زبان و تفاوت های شناختی بین جنسیت ها. در این بین متغیری به نام محاسبات تکاملی هست که در آن، قوانین تکامل برای ابداع الگوریتم های موفق و کارا استفاده می شوند. یک شاخه یا اشتقاق از این نوع محاسبه، زندگی مصنوعی است. این ها شبیه سازی شده های نرم افزاری هستند که اکولوژی های زیستی را تقلید می کنند. همچنین نوعی نظریه داروینیسم عصبی هست که برای توضیح شکل گیری مدارهای عصبی از پدیده ی تکامل استفاده می کند.

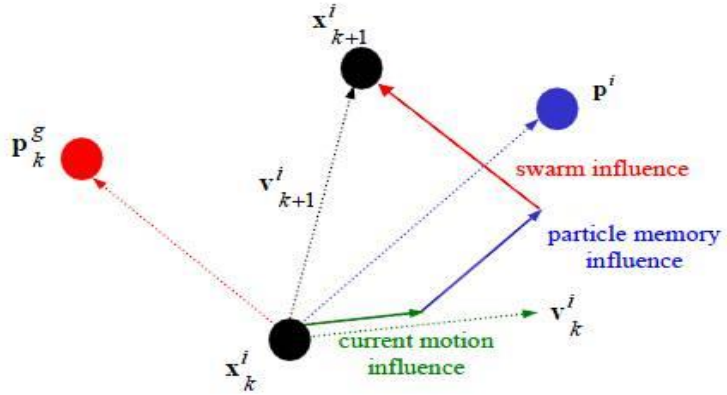
محاسبات تکاملی:

در سال های اخیر، مقدماتی در مورد حوزه ی جدید مطالعاتی به عنوان محاسبات تکاملی معرفی شده است. این حوزه ی جدید مجموعه ای از روش های محاسباتی است که بر اساس قوانین تکامل زیستی مدل سازی شده است. (میچل ۱۹۹۶). محاسبات تکاملی در مسیر سه هدف حرکت می کند که مسائل جهان واقعی را در بر می گیرد و شامل پیش بینی های اقتصادی، طراحی برنامه های رایانه ای و یادگیری ربات است. هم چنین از محاسبات تکاملی برای مدل سازی و درک بهتر سیستم های تکامل طبیعی استفاده می شود که در حوزه هایی مثل اقتصاد، ایمنی شناسی و بوم شناسی به وجود آمده اند. سومین و مهم ترین هدف ما این است که آن را به عنوان یک استعاره برای عملکرد فرآیندهای فکری انسان ارایه کنیم. محاسبات تکاملی سه روش شناسی دارد؛ الگوریتم های ژنتیکی، راهبردهای تکاملی و برنامه ریزی تکاملی. صر نظر از روش های مورد استفاده، تمام فرم های محاسبات تکاملی دارای یک رویکرد کلی برای حل مسائل هستند که شامل مراحل متعددی می باشند:

۱. ایجاد راه حل های ممکن یا "راه حل های پیشنهادی" شامل شماری از افرادی است که دارای مجموعه ای از خصوصیات یا مشخصه ها هستند.
۲. ارزیابی تناسب راه حل ها. ارزیابی این که هر راه حل چگونه مسائل داده شده را حل می کند.
۳. انتخاب راه حل هایی که بالاترین تناسب رادارند و مبتنی بر حداقل تناسب آستانه های از پیش تعیین شده هستند.
۴. تولید نسل جدید"فرزندان" از روش هایی که بیش ترین تناسب رادارند. این نسل جدید می تواند محصول قطع و روی هم افتادن مواد ژنتیکی دو یا چند "والد" یا منتج از جهش های تصادفی باشد.
۵. مراحل ۴-۲ تا دست یابی به راه حل بهینه تکرار می شوند.

الگوریتم بهینه سازی ازدحام ذرات :

PSO (الگوریتم بهینه سازی ازدحام ذرات PSO) از دسته الگوریتم های بهینه سازی است که بر مبنای تولید تصادفی جمعیت اولیه عمل می کنند. در این الگوریتم با الگوگیری و شبیه سازی رفتار پرواز دسته جمعی (گروهی) پرندگان یا حرکت دسته جمعی (گروهی) ماهی ها بنا نهاده شده است. هر عضو در این گروه توسط بردار سرعت و بردار موقعیت در فضای جستجو تعریف می گردد. در هر تکرار زمانی، موقعیت جدید ذرات با توجه به برار سرعت و بردار موقعیت در فضای جستجو تعریف می گردد. در هر تکرار زمانی، موقعیت جدید ذرات با توجه به بردار سرعت فعلی، بهترین موقعیت یافت شده توسط آن ذره و بهترین موقعیت یافت شده توسط بهترین ذره موجود در گروه، به روز رسانی می گردد. این الگوریتم ر ابتدا برای پارامترهای پیوسته تعریف شده بود اما با توجه به اینکه در برخی از کاربردها با پارامترهای گسسته سروکار داریم، این الگوریتم به حالت گسسته نیز بست داده شده است. الگوریتم بهینه سازی ازدحام ذرات را در حالت گسسته با (BPSO) معرفی می گردد. در این الگوریتم موقعیت هر ذره با مقدار یک تعریف می گردد. در این الگوریتم موقعیت هر ذره با مقداربااینری صفر و یا یک نشان داده می شود. در BPSO مقدار هر ذره می تواند از صفر به یک و یا از یک به صفر تغییر کند. سرعت هر ذره نیز به عنوان احتمال تغییر هر ذره به مقدار یک تعریف می گردد. در این فصل بخش های مختلف این الگوریتم معرفی و بررسی خواهد شد.



شکل (۱) موقعیت ذرات در فضا

تعاریف اولیه الگوریتم PSO:

فرض کنید یک فضای جستجوی d بعدی داریم. iاُمین ذره در این فضای d بعدی باب بردار موقعیت Xi به شکل زیر توصیف می گردد:

$$X_i = (x_{i_1}, x_{i_2}, x_{i_3}, \dots, x_{i_d}) \quad (۱)$$

بردار سرعت iاُمین ذره نیز با بردار Vi به شکل زیر تعریف می گردد:

$$V_i = (v_{i_1}, v_{i_2}, v_{i_3}, \dots, v_{i_d}) \quad (۲)$$

بهترین موقعیتی که ذره iاُم پیدا کرده است را با $P_{i\ .best}$ تعریف می کنیم:

$$P_{i\ .best} = (p_{i_1}, p_{i_2}, p_{i_3}, \dots, p_{i_d}) \quad (۳)$$

بهترین موقعیتی که بهترین ذره در بین کل ذرات پیدا کرده است را با $P_{g\ .best}$ به صورت زیر تعریف می کنیم:

$$P_{g\ .best} = (p_{g_1}, p_{g_2}, p_{g_3}, \dots, p_{g_d}) \quad (۴)$$

برای به روز رسانی محل هر کدام از ذرات از رابطه زیر استفاده می کنیم:

$$V_i(t) = w * V_i\ (t-1) + c_1 * rand_1 * (P_{i\ .best} - X_i\ (t-1)\ ) + c_2 * rand_2 * (P_{g\ .best} - X_i\ (t-1)\ )$$

$$X_i = X_i\ (t-1) + V_i(t) \quad (۵)$$

- W: ضریب وزنی اینرسی (حرکت در مسیر خودی) که نشان دهنده میزان تأثیر بردار سرعت تکرار قبل $(V_i(t))$ بر روی بردار سرعت در تکرار فعلی $(V_i(t+1))$ است.
- c1: ضریب ثابت آموزش (حرکت در مسیر بهترین مقدار ذره مورد بررسی)
- c2: ضریب ثابت آموزش (حرکت در مسیر بهترین ذره یافت شده در بین کل جمعیت)
- rand1 , rand2 : دو عدد تصادفی با توزیع یکنواخت در بازه ۰ تا ۱
- Vi(t-1) بردار سرعت در تکرار (t-1)ام
- Xi(t-1) بردار موقعیت در تکرار (t-1) ام

برای جلوگیری از افزایش بیش از حد سرعت حرکت یک ذره در حرکت از یک محل به محل دیگر (واگرا شدن بردار سرعت)، تغییرات سرعت را به رنج Vmin تا Vmax محدود می کنیم؛ یعنی $V_{min} \leq V \leq V_{max}$. حد بالا و پایین سرعت با توجه به نوع مسئله تعیین می گردد.

محدود سازی فضا:

بعضی از مسائل دامنه تعریفی خاصی برای پارامترهای خود دارند و تنها در این دامنه دارای مقداری محدود، منطقی و تعریف شده هستند. به عبارت دیگر اگر در مسئله مورد بررسی قید و یا قیودی وجود داشته باشد، باید توسط مکانیزمی این قیود لحاظ گردند تا از ورود ذرات به فضای غیر مجاز جلوگیری شود. این مکانیزم را اصطلاحاً محدودسازی فضا می نامند. اگر از این مکانیزم ها استفاده نشود، پاسخ پیدا شه توسط الگوریتم اشتباه و یا غیر قابل اطمینان است. مثلاً تابع زیر برای مقادیر منفی x در اکثر زبان های برنامه نویسی، خطا محسوب می شود.

$$f(x) = \sum_{d=1}^{D} \sqrt{x} \qquad (۶)$$

مکانیزمی که برای لحاظ کردن این قید استفاده می شود، بصورت زیر است:

$$X = \max(0,x) \qquad (۷)$$

در تابع فوق مقادیر مجاز x؛ یعنی $x \geq ۰$ بدون هیچ گونه تغییری نگاشت می شوند اما مقادیر غیر مجاز x؛ یعنی $x < ۰$ به مقدار مجاز $x=0$ نگاشت می شوند. در حالت کلی تر اگر بخواهیم محل ذرات بصورت $[\alpha_{min}, \alpha_{max}]^D$ باشد، برای محدودسازی می توان از رابطه زیر استفاده کرد:

$$\alpha_d = \min \{\max (X_d, V_d, \alpha_{min}), \alpha_{max}\} \qquad (۸)$$

با استفاده از رابطه فوق محل ذراتی که در خارج از محدوده تعریف شده قرار داشته باشند به داخل محدوده مجاز نگاشت می شوند و محل سایر ذراتی که در محدوده مجاز قرار دارند تغییری داده نمی شود.

مراحل اجرای الگوریتم PSO:

بعضاً در مراجع مختلف در نحوه تفکیک مراحل اجرای الگوریتم اختلافاتی دیده می شود؛ یعنی در یک مواقع مراحل بصورت تفکیک شده تری ذکر می شوند و در برخی مواقع دو یا تعداد بیشتری از مراحل را با هم ترکیب کرده و تبدیل به یک مرحله می کنند. اما این موضوع اشکالی در برنامه نویسی های انجام شده ایجاد نمی کند زیرا آنچه که مهم است اجرا شدن مراحل برنامه به ترتیبی که در ادامه خواهد آمد، است و نحوه تفکیک این مراحل. مثلاً در برخی مراجع مرحله ۴ و ۵ را با هم ترکیب می کنند؛ یعنی مرحله به روز رسانی سرعت ذرات و انتقال ذرات به محل های جدید را به عنوان یک مرحله در نظر می گیرند. این تغییر اشکالی در روند اجرای الگوریتم ایجاد نخواهد کرد.

مرحله ۱، تولید تصادفی جمعیت اولیه ذرات:

تولید تصادفی جمعیت اولیه بطور ساده عبارت است از تعیین تصادفی محل اولیه ذرات با توزیع یکنواخت در فضای حل (فضای جستجو). مرحله تولید تصادفی جمعیت اولیه تقریباً در تمامی الگوریتم های بهینه سازی احتمالاتی وجود دارد. اما در این الگوریتم علاوه بر محل تصادفی اولیه ذرات، مقداری برای سرعت اولیه ذرات نیز اختصاص می یابد. رنج پیشنهادی اولیه برای سرعت ذرات را می توان از رابطه زیر استخراج کرد.

$$\frac{X_{min} - X_{max}}{2} \leq V \leq \frac{X_{max} - X_{min}}{2} \qquad (۹)$$

انتخاب تعداد ذرات اولیه:

می دانیم که افزایش تعداد ذرات اولیه موجب کاهش تعداد تکرارهای لازم برای همگرا شدن الگوریتم می گردد. اما گاهی مشاهده می شود که کاربران الگوریتم های بهینه سازی تصور می کنند که این کاهش در تعداد تکرارها به معنی کاهش زمان اجرای برنامه برای رسیدن به همگرایی است، در حالی که چنین تصوری کاملاً غلط است. هرچند که افزایش تعداد ذرات اولیه کاهش تعداد تکرارها را در پی دارد. اما افزایش در تعداد ذرات باعث می گردد که الگوریتم در مرحله ارزیابی ذرات زمان بیشتری را صرف نماید که این افزایش در زمان ارزیابی باعث می شود که زمان اجرای الگوریتم تا رسیدن به همگرایی با وجود کاهش در تعداد تکرارها، کاهش نیابد. پس افزایش تعداد ذرات نمی تواند برای کاهش زمان اجرای الگوریتم مورد استفاده قرار گیرد. تصور غلط دیگری وجود دارد و آن این است که برای کاهش زمان اجرای الگوریتم می توان تعداد ذرات را کاهش داد. این تصور نیز وجود دراد و آن این است که برای کاهش زمان لازم برای ارزیابی ذرات می گردد، اما برای این که الگوریتم به جواب بهینه برسد. تعداد تکرارها افزایش می یابد. (اگر شرط همگرایی را عدم تغییر در هزینه بهترین عضو در چندین تکرار متوالی در نظر بگیریم) که در نهایت باعث می شود زمان اجرای برنامه برای رسیدن به پاسخ بهینه کاهشی نداشته باشد. همچنین باید متذکر شد که کاهش تعداد ذرات ممکن است موجب گیر افتادن در مینیم های محلی شود و الگوریتم از رسیدن به مینیمم اصلی باز ماند. اگر ما شرط همگرایی را تعدا تکرارها در نظر گرفته باشیم، هرچند با کاهش تعداد ذرات اولیه زمان اجرای الگوریتم کاهش می یابد اما جواب به دست آمده، حل بهینه ای برای مسئله نخواهد بود. زیرا الگوریتم بصورت ناقص اجرا شده است. بطور خلاصه، تعداد جمعیت اولیه با توجه به مسئله تعیین می گردد. در حالت کلی تعداد ذرات اولیه مصالحه ای بین پارامترهای درگیر در مسئله است. بطور تجربی انتخاب جمعیت اولیه ذرات به تعداد ۲۰ تا ۳۰ ذره انتخاب مناسبی است که تقریباً برای تمامی مسائل تست به خوبی جواب می دهد. می توانید تعداد ذرات را کمی بیشتر از حد لازم نیز در نظر بگیرید تا کمی حاشیه ایمنی در مواجهه با مینیمم های محلی داشته باشید.

ارزیابی تابع هدف (محاسبه هزینه یا برازندگی) ذرات:

در این مرحله باید هر یک از ذرات را که نشان دهنده یک حل برای مسئله مورد بررسی است، ارزیابی کنیم. بسته به مسئله مورد بررسی، روش ارزیابی متفاوت خواهد بود. مثلاً اگر امکان تعریف یک تابع ریاضی برای هدف وجود داشته باشد، با جایگذاری پارامترهای ورودی (که از بردار موقعیت ذره استخراج شده اند) در این تابع ریاضی، به راحتی مقدار هزینه این ذره محاسبه خواهد شد. توجه داشته باشید که هر ذره حاوی اطلاعات کاملی از پارامترهای ورودی مسئله است که این اطلاعات استخراج شده و در تابع هدف قرار می گیرد. گاهی اوقات امکان تعریف یک تابع ریاضی برای ارزیابی ذرات وجود ندارد. این حالت زمانی پیش می آید که ما الگوریتم را با یک نرم افزار دیگر لینک کرده باشیم و یا الگوریتم را برای داده های تجربی (آزمایش) استفاده می کنیم. در این گونه موارد باید اطلاعات مربوط به پارامترهای ورودی نرم افزار یا آزمایش را از بردار موقعیت ذرات استخراج کرده و در اختیار نرم افزار لینک شده با الگوریتم جایگذاری کرد و یا درآزمایش مربوطه اعمال نمود. با اجرای نرم افزار و یا انجام آزمایش و مشاهده و اندازه گیری نتایج هزینه ای را که هر یک از ذرات در پی دارند مشخص خواهد شد.

ثبت بهترین موقعیت برای هر ذره ($P_{i\ .best}$) و بهترین موقعیت در بین کل ذره ها ($P_{g\ .best}$)

در این مرحله با توجه به شماره تکرار، دو حالت قابل بررسی است:

- اگر در تکرار اول باشیم .(t=1) موقعیت فعلی هر ذره را به عنوان بهترین محل یافت شده برای آن ذره در نظر می گیریم.

$$P_{i\ ,best} = X_i(t) \quad ,i = 1, 2, 3, \dots, d$$

$$\text{cost}\left(P_{i\ ,best}\right) = \text{cost}\left(X_j(t)\right) \qquad (۱۰)$$

در سایر تکرارها مقدار هزینه به دست آمده برای ذرات در مرحله ۲ را با مقدار بهترین هزینه به دست آمده برای تک تک ذرات مقایسه می کنیم. اگر این هزینه کمتر از بهترین هزینه ثبت شده برای این ذره باشد، آنگاه محل و هزینه این ذره جایگزین مقدار قبلی می گردد. در غیر این صورت تغییری در محل و هزینه ثبت شده برای این ذره ایجاد نمی شود؛ یعنی:

$$\begin{cases} \text{if } \mathrm{cost}(X_i(t)) < cost\ (P_{i,best}) \\ \text{else Not change} \end{cases} \Longrightarrow \begin{cases} \mathrm{cost}\ (P_{i,best}) = \mathrm{cost}\ (X_j(t)) \\ P_{i,best} = X_i(t) \end{cases} i = 1,2,3,\dots,d \qquad (۱۱)$$

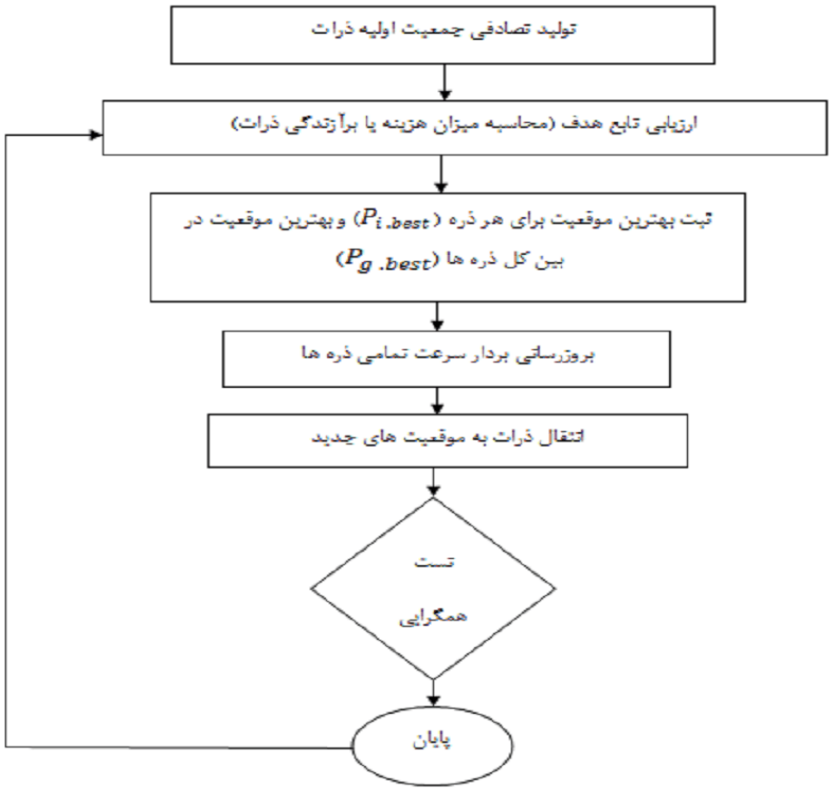
به روز رسانی بردار سرعت تمامی ذره ها:

$$V_i(t) = w * V_i\ (t-1) + c_1 * rand_1 * (P_{i\ .best} - X_i\ (t-1)\ ) + c_2 * rand_2 * (P_{g\ .best} - X_i\ (t-1)\ ) \qquad (۱۲)$$

ضرایب w,c1, c2 با توجه به مسئله مورد نظر به روش تجربی تعیین می گردند. اما به عنوان یک قانون کلی در نظر داشته باشید که w باید کمتر از یک باشد زیرا اگر بزرگتر از یک انتخاب شود، V(t) دائماً افزایش می یابد تا جایی که واگرا گردد. همچنین توجه داشته باشید، هرچند در تئوری ضریب w می تواند منفی نیز باشد اما در استفاده عملی از این الگوریتم هیچ گاه این ضرایب را منفی در نظر نگیرید زیرا منفی بودن w موجب ایجاد نوسان در(V(t می شود. انتخاب مقدار کوچک برای این ضریب (w) نیز مشکلاتی را در پی خواهد داشت. اغلب در الگوریتم PSO مقدار این ضریب را مثبت و در رنج ۰,۷ تا ۰,۸ در نظر می گیرند. C2و c3 نیز نباید زیاد بزرگ انتخاب شوند زیرا انتخاب مقادیر بزرگ برای این دو ضریب باعث انحراف شدید ذره از مسیر خودی می شود. اغلب در الگوریتم PSO مقدار این ضرایب را مثبت و در رنج ۱,۵ الی ۱,۷ در نظر می گیرند. لازم به یادآوری است که الزاماً مقادیر پیشنهادی فوق تنها انتخاب های ممکن برای ضرایب w,c ,۱c ,۲نیست بلکه با توجه به مسئله مورد بررسی ممکن است انتخاب های بهتری غیر از موارد فوق وجود داشته باشد.

تست همگرایی:

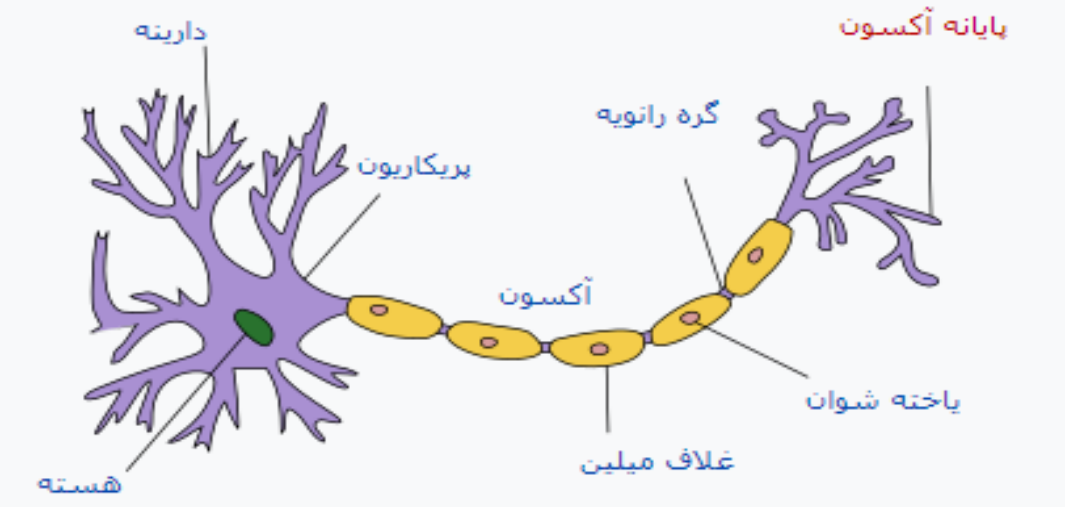
تست همگرایی در این الگوریتم مانند سایر الگوریتم های بهینه سازی است. برای بررسی الگوریتم روش های گوناگونی وجود دارد. برای مثال می توان تعداد مشخصی تکرار را از همان ابتدا معلوم کرد و در هر مرحله بررسی کرد که آیا تعداد تکرارها به مقدار تعیین شده رسیده است؟ اگر تعداد تکرارها کوچکتر از مقدار تعیین شده اولیه باشد، آن گاه باید به مرحله ۲ بازگردید در غیر این صورت الگوریتم پایان می پذیرد. روش دیگری که اغلب در تست همگرایی الگوریتم استفاده می شود، این است که اگر در چند تکرار متوالی مثلاً ۱۵ یا ۲۰ تکرار تغییری در مقدار هزینه بهترین ذره ایجاد نگردد، آنگاه الگوریتم پایان می یابد، در غیر این صورت باید به مرحله ۲ بازگردید. دیاگرام گردشی (فلوچارت) الگوریتم PSO در شکل نشان داده شده است.



شکل(۲)- فلوچارت الگوریتم ازدحام ذرات

دندریت:

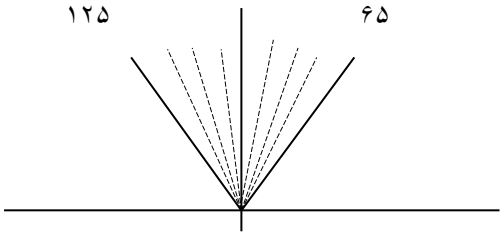
دِندریتها زائده‌های رشته مانندی هستند که به جسم سلولی یاخته‌های عصبی (نورون‌ها)، متصل هستند. دندریت ها نقشِ آورنده پیام‌های صادر شده از یاخته حسی مجاور یا یاخته‌های عصبی رده بالاتر به درون یاخته را به عهده دارند. این پیام‌های عصبی، بصورت سیگنال‌های الکتریکی در درازای دندریت ها حرکت می‌کنند. دندریت‌ها معمولاً بیش از یک عدد، کوتاه و منشعب (بسیار بلند در نورونهای حسی، سلولهای پورکینیه در مخچه و سلولهای پیرامیدال در مغز) هستند و دارای همه ارگانلها بجز دستگاه گلژی، ختم به انشعابات ظریف و متعدد در انتها. وظیفه آن گیرنده اصلی نورون (انتقال تحریکات دریافتی سایر نورونها یا سلولهای حساس به پریکاریون) است. فرهنگستان زبان فارسی برای دندریت واژه فارسی**"دارینه"** را برگزیده است. دار در فارسی به معنای **"درخت"** است و شکل درختواره دندریت‌ها علت این نامگذاری است. گفتنی است نورونهای حرکتی آکسون بلند و دندریت کوتاه دارند.



شکل(۳) ساختار نورون

شرح مسأله:

این مساله ساخت درختی شبیه به دندریت با استفاده از الگوریتم تکاملی است. در این مسأله فرض بر این است که تعدادی پاره خط در فضای صفحه داریم. می خواهیم این پاره خط ها را بنحوی جابجا کنیم که تشکیل یک درخت (دندریت) را بدهد. در پیاده سازی این مسأله فرض می کنیم که طول همه پاره خط ها عدد ثابت ۲ باشند که در یک فضای 10×10 دو بعدی توزیع شده اند. شیب تمام پاره خطهای صفحه فرض می کنیم که بین ۶۵ تا ۱۰۵ درجه در نوسان باشد.



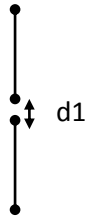
در در این مسأله x،y نقطه شروع پاره خط و زاویه ɵ خط باشد. بصورت

X	Y	ɵ

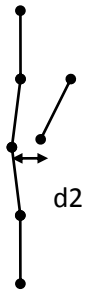
با توجه به این که با داشتن این اطلاعات برای هر خط دیگر نیازی به نگهداری x_2 و y_2 (انتهای پاره خط) نیست و این نقاط با استفاده از رابطه $x_2 = x + 2cosd(\theta)$ و $y_2 = y + 2sind(\theta)$ بدست می آید، بنابراین مقادیر x_2و y_2 را نگهداری نمی کنیم. مهمترین بخش هر الگوریتم نوشتن تابع fitness آن است. در اینجا نیز برای نوشتن تابع fitness به معیارهای زیر توجه می کنیم:

- تعداد خطوط مشابه (یعنی خطوطی که دارای x,y و زاویه θ یکسانی هستند)- بعنوان یک پارامتر منفی. (ما این معیار را same می نامیم)
- تعداد خطوطی که زاویه بین ۸۵ تا ۹۵ دارند. از این خطوط بعنوان تنه درختمان استفاده می کنیم. قرار گذاشته ایم که تعداد خطوط اینچنینی حتماً بین n/2 تا n/3 نقاط باشد و آن دسته با تعداد خطوط اینچنینی خارج از این رنج عددی بزرگ را بعنوان خروجی تابع fitness داشته باشند که این بخودی خود سبب دور انداختن آنها خواهد شد. در صورتی که تعداد این خطوط در رنج مورد نظر باشد آنها را براساس مقدار y خروجی اشان مرتب

می کنیم و مجموع فاصله انتهای هر خط از ابتدای خط بعدی را بعنوان یک معیار در نظر می گیریم. بدیهی است کوچکتر بودن این مقدار سبب تولید درختانی خوش فرم تر خواهد شد. (این معیار را d1 است)



- فاصله سایر خطوط از خطوط موجود در ساقه که در این صورت هرچه این مقدار کمتر باشد درخت بهتری خواهیم داشت. (این معیار را d2 است)



خروجی تابع fitness بصورت d=d1+d2+same خواهد بود که بدیهی است هرچه که این مقدار کمتر باشد درخت بهتری خواهیم داشت.

شرح پیاده سازی:

برای پیاده سازی و رسم درخت از الگوریتم pso و از حالت فرض شده برای مساله استفاده می کنیم. در پیاده سازی ابتدا متغیر های اولیه را تعریف کرده و ماترس های اولیه ی swarm را می سازیم.به x وy و angle مقادیر تصادفی می دهیم. البته زاویه باید در محدوده ی خواسته شده در مساله باشد.بنابر این هرparticle سایزی برابر با ۳ دارد. به تعداد دلخواه در این جا ۱۰ تا پاره خط می خواهیم ایجاد کنیم. تعداد کل swarmها نیز ۱۵۰۰ داده ایم. پارامتر های update رانیز مقداردهی می کنیم. ماتریس های particle_position وparticle_best_position با مقادیر تصادفی اولیه پر می شوند. ابتدا این دو با هم برابرند.

```
%%%%%%%%%% initial values %%%%%%%%%%%%%
%number of iterations
iteration=700;

%size of swarms
swarm_size=1500;

%size of  each particle,we have (x & y & Angle) for each line
particle_size=3;

%size of lines,in fact we repmat our particle matrix,equal to the number of lines we want
line_size=10;

%number of rows of particle matrix
row_size=line_size*particle_size;

%bounds of angles we use in this algorithm
angle_min=65; angle_max=105;
bound=(angle_max-angle_min);

%parameters use for updating
phi1 = 1.99;
phi2 = 0.3;
%phi = phi1 + phi2;
%khi = 2/abs(2-phi-sqrt(phi^2-4*phi));
khi=0.6;

%define paricle ,best ,velocity,cost matrixes
particle_position=zeros(row_size,swarm_size);
particle_best_position=zeros(row_size,swarm_size);
particle_velocity=zeros(row_size,swarm_size);
particle_cost=zeros(swarm_size,1);
particle_best_cost=[];
globalbest=[];

%define random cooardinate for each particle
% x & y must be in rang of 0 to 10 ,angle must be in range of bound
for i=1:swarm_size
    for v=1:line_size
        particle_position(((v-1)*particle_size)+1,i)=rand*10;
        particle_position(((v-1)*particle_size)+2,i)=rand*10;
        particle_position(((v-1)*particle_size)+3,i)=rand *bound+angle_min;
    end;
end

%first our best matrix is equal to our current matrix
particle_best_position=particle_position;
```

سپس تابع fitness فراخوانی می شود و costهر یک از مجموعه خط ها حساب می شود. به global best براین مبنا مقدار می دهیم. velocity را update می کنیم. شرط های جایگزینی را چک می کنیم تا در صورت لزوم ،ماتریس های بهترین مکان و costوglobalbest با مقادیر بهتر بروز شوند.

```
%calculate fitness for each group of particle in each swarms

for j=1:swarm_size
    p1=particle_best_position(:,j);
    particle_cost(j)=treecost(p1);
end
 %calculate initial global best
 [globalbest,ind]=sort(particle_cost);
 particle_global_best_cost=globalbest(1);
 particle_global_best_ind=ind(1);

%initial iteration
for i=1:iteration

    %updating velacity & position
    for j=1:swarm_size
        for v=1:row_size
            particle_velocity(v,j)=khi*particle_velocity(v,j)+phi1*(particle_best_position(v,j)-particle_position(v,j))+...
                phi2*((particle_best_position(v,particle_global_best_ind))-particle_position(v,j));
            particle_position(v,j)=particle_position(v,j)+particle_velocity(v,j);
        end
    end

    %%%%%%%%%%%%%5 replacing the best %%%%%%%%%%%%%
    for j=1:swarm_size

        p2=particle_position(:,j);
        % changing best positions & particle cost if it's necessery
        test(j)=treecost(p2);
        if particle_cost(j)>test(j)
            for v=1:row_size
                particle_best_position(v,j)=particle_position(v,j);
            end
            particle_cost(j)=test(j);
        end

        % checking the global best if it's necessery
        if particle_global_best_cost>test(j)
            particle_global_best_cost=test(j);
            particle_global_best_ind=j;
        end

    end

end
```

از دستور line استفاده کرده و درخت را رسم می کنیم و در خروجی نمایش می دهیم.تصویر ما ۱۰x۱۰ می باشد.

```
axis ([0 10 0 10]);
xlabel('X');
ylabel('Y');
show = particle_best_position(:,particle_global_best_ind);
for j=1:line_size
   tree= line([show(((j-1)*particle_size)+1) (show(((j-1)*particle_size)+1)+2*...
       cosd(show(((j-1)*particle_size)+3)))],[show(((j-1)*particle_size)+2) (show(((j-1)*particle_size)+2)+2*sind(show(((j-1)*particle_size)+3)))]);
end;
p=min(particle_cost);
clc;
disp('" pso dendrite"')
disp([' best fitness in last iteration(same+d1+d2)= ' num2str(p)])
```

و اما تابع fitness که در یک mfile جدا می باشد. مقدارsame+d1+d2 را حساب می کند.که مسلما هر چه این مقدار بزرگتر باشد بدتر است.ورودی این تابع در واقع دسته ای از خط های ما هستند که به صورت بردار ستونی می باشد. به طوری که :

X1

Y1

E1

X2

Y2

E2

.

.

پس تابع fitness ما در واقع مقدار cost را برای بردار فوق، (برای هر دسته swarm) حساب می کند. ابتدا چک می کند که اگر خطوطی با مختصات یکسان باشند، به مقدار same اضافه می کند. سپس طبق شرط زوایا برگ ها را از تنه جدا می کند. متغیر ی را بازای خط های تنه زیاد می کنیم .اگر در شرط محدوده ی خواسته شده (که می خواهیم تعداد انها در رنج تعداد خطوط /۳ وتعداد خطوط/۲ باشد) صدق نکند. پس

same بزرگی به ان اختصاص می دهیم. اگر غیر از ان بود آنها را بر اسای مختصه ی y شان مرتب می کنیم و سپس معیار d1 را بکار می بریم که فاصله ی اتنهای خط با ابتد ای خط دیگر تنه را حساب می کند که مسلما هر چه بیشتر باشد بدتر است. (در اینجا طول پاره خط را ۲ فرض کردیم.) و d2 نیز فاصله ی برگ را با تنه در نظر می گیرد،که هر چه مقدار فاصله ی آنها از هم کمتر باشد بهتر است.

```
function cost=treecost(a)
    line_size=10;
    particle_size=3;
    row_size=line_size*particle_size;
    n1=line_size/3;
    n2=line_size/2;
    particle_group=a;
    same=0; d1=0;  d2=0; d12=0; cnt=0;
    pedicle=[];leaf=[];
     % calculating 'same '
     %if x & y & angle are equal,then increase same
    for j=1:line_size
        for v=j+1:line_size
            if ((abs(particle_group(((j-1)*particle_size)+1)-particle_group(((v-1)*particle_size)+1))==0)&&...
                    (abs(particle_group(((j-1)*particle_size)+2)-particle_group(((v-1)*particle_size)+2))==0)&&...
                    (abs(particle_group(((j-1)*particle_size)+3)-particle_group(((v-1)*particle_size)+3))==0))
                same = same+1;
            end
        end
    end
   % separating leaf lines from pedicle lines
    for j=1:line_size
        if ((particle_group(((j-1)*particle_size)+3)>85)&&(particle_group(((j-1)*particle_size)+3)<95))%then they are pedicle

            pedicle=[ pedicle ; particle_group(((j-1)*particle_size)+1) , particle_group(((j-1)*particle_size)+2) , particle_group(((j-1)*particle_size)+3) ];
            cnt = cnt+1;
        else
            leaf=[ leaf ; particle_group(((j-1)*particle_size)+1) , particle_group(((j-1)*particle_size)+2) , particle_group(((j-1)*particle_size)+3) ];
        end
    end
    % cheking if out of bound,it should be discarded
    if ~((cnt>n1)&&(cnt<n2))
        same = same+9999999;
    else
        pedicle=sortrows(pedicle,2);%sort pedicle base on y
        % calculating d1
        for j=1:cnt-1
            d12=(((pedicle(j+1,2)-(pedicle(j,2)+2*sind(pedicle(j,3))))^2)+((pedicle(j+1,1)-(pedicle(j,1)+2*cosd(pedicle(j,3))))^2));
            d1 = d1+ d12;
        end
        % calculating d2
        % calculate leaf size
         leafnum=line_size-cnt ;
        for j=1:leafnum
              bignum=9999999;
            for v=1:cnt
                lp_distance = ((leaf(j,2)-(pedicle(v,2)))^2)+((leaf(j,1)-(pedicle(v,1)))^2);
                if (lp_distance<bignum)
                    bignum=lp_distance;
                end
            end
            d2 = d2+ bignum;
        end
    end
    %result of our fitness
    result=same+d1+d2;
    cost=result;
end
```

نمونه ای از ترسیم درخت در متلب با تعداد ۱۰ خط با ۱۰۰۰ بار تکرار الگوریتم: