مجله محاسبات نرم

SOFT COMPUTING JOURNAL

تارنمای مجله: scj.kashanu.ac.ir

دانشگاه کاشان
University of Kashan

ارائهٔ یک الگوریتم فرااکتشافی جدید مبتنی بر رفتار پرندهٔ تیهو برای حل مسائل بهینه‌سازی پویا✧

مجید محمدپور[۱]، کارشناسی ارشد، بهروز مینایی بیدگلی[۲]، دانشیار، حمید پروین[۳*]، استادیار

[۱] باشگاه پژوهشگران و نخبگان، واحد یاسوج، دانشگاه آزاد اسلامی، کهگیلویه و بویراحمد، ایران.

[۲] دانشکده مهندسی کامپیوتر، دانشگاه علم و صنعت، تهران، ایران.

[۳] دانشکده مهندسی کامپیوتر، واحد نورآباد ممسنی، دانشگاه آزاد اسلامی، نورآباد، ایران.

اطلاعات مقاله

تاریخچه مقاله:

دریافت ۱۵ دی ماه ۱۳۹۸

پذیرش ۰۴ مهر ماه ۱۳۹۹

کلمات کلیدی:

الگوریتم SSPCO

محک قله‌های متحرک

بهینه‌سازی پویا

خطای برون‌خطی

حافظه

خوشه‌بندی

MMSSPCO

چکیده

الگوریتم SSPCO گونه‌ای از الگوریتم‌های هوش جمعی و برگرفته‌شده از رفتار پرنده تیهو است. کارآیی این الگوریتم برای حل مسائل بهینه‌سازی ایستا به اثبات رسیده است؛ اما کارآیی این الگوریتم تا به‌حال برای حل مسائل بهینه‌سازی پویا مورد آزمایش قرار نگرفته است. به‌دلیل ماهیت NP-Hard بودن مسائل پویا، این الگوریتم به‌تنهایی قادر به‌حل این‌گونه از مسائل بهینه‌سازی نمی‌باشد. بنابراین برای این‌که الگوریتم قادر به ردیابی بهینه متغیر در این مسائل باشد، باید راهکارهایی به‌همراه این الگوریتم ارائه داد که بتوانند عملکرد این الگوریتم را درمواجهه با محیط‌های پویا افزایش دهد. در این مقاله دو راه‌حل برای ترکیب با الگوریتم SSPCO ارائه شده است که عبارتند از، روش چندجمعیتی و حافظه با تخمین تراکم گوسی. مشکلی که در اکثر روش‌های چندجمعیتی وجود دارد این است که با افزایش کنترل نشده جمعیت، سرعت و راندمان الگوریتم به تدریج کاهش می یابد. روش چندجمعیتی ارائه شده در این مقاله به‌صورت تطبیقی با فضای مسئله می‌باشد و هر زمان که نیاز به افزایش جمعیت باشد یک جمعیت به‌صورت تطبیقی ایجاد می‌شود و این موضوع باعث می‌شود که مشکل روش‌های قبلی کاهش یابد. یکی از مواردی که در حل مسائل غیرقطعی باید مشخص شود، استفاده از داده‌های گذشته نزدیک برای پیش‌بینی آینده نزدیک است. در این مقاله با توجه به این موضوع برای حفظ اطلاعات گذشته از حافظه جدیدی به‌نام حافظه تخمین تراکم گوسی استفاده شده است. این حافظه عیوب حافظه استاندارد را برطرف نموده و باعث بهبود کارآیی الگوریتم پیشنهادی می‌شود. برای آزمایش کارآیی روش پیشنهادی از تابع معروف محک قله‌های متحرک که رفتاری شبیه به مسائل پویا را شبیه‌سازی می‌کند، استفاده شده است. الگوریتم پیشنهادی با ۱۰ مورد از مشهورترین الگوریتم‌های بهینه‌سازی پویا مقایسه گردیده است. همان‌گونه که از نتایج تجربی و آزمایش‌ها مشخص می‌باشد روش پیشنهادی توانسته خطای برون‌خطی را تا حدود بسیار زیادی نسبت به سایر روش کاهش دهد و خطای تولید شده برای روش پیشنهادی بسیار ناچیز است.

۱. مقدمه

در مسائل مطرح‌شده در حوزهٔ بهینه‌سازی، طبقه‌بندی‌های مختلفی بسته به حوزهٔ کاربردی وجود دارد. یکی از این طبقه‌بندی‌ها بر اساس تغییرپذیری یا عدم تغییرپذیری محیط اطراف تعیین می‌شود. به این ترتیب در صورتی که محیط شاهد عدم تغییرات در خود باشد، مسائل مربوط تحت عنوان مسائل بهینه‌سازی ایستا مطرح شده و در صورت تغییرات محیطی، با مسائل بهینه‌سازی پویا روبه‌رو می‌شود. در مسائل نوع اول، هدف اصلی یافتن و یا تخمین هرچه بهتر نقطهٔ (نقاط) بهینه

✧ نوع مقاله: پژوهشی

* نویسنده مسئول

پست‌های الکترونیک: m.mohammadpour@iauyasooj.ac.ir (محمدپور)

b_minaei@iust.ac.ir (مینایی بیدگلی)

parvin@iust.ac.ir (پروین)

نحوه ارجاع به مقاله: محمدپور، مجید، مینایی بیدگلی، بهروز، پروین، حمید، «ارائهٔ یک الگوریتم فرااکتشافی جدید مبتنی بر رفتار پرندهٔ تیهو برای حل مسائل بهینه‌سازی پویا»، مجله محاسبات نرم، جلد ۸، شماره ۲، ص ۳۸ـ۶۵، پاییز و زمستان ۱۳۹۸.

است. این در حالی است که در مسائل نوع دوم نه‌تنها باید هدف اصلی در حالت ایستا ارضا شود بلکه بایستی بتوان هرچه سریع‌تر نقطهٔ (نقاط) بهینه را دنبال کرد. این امر از آنجا ناشی می‌شود که در محیط‌های پویا به‌دلیل تغییرپذیری محیطی امکان تغییر نقطهٔ (نقاط) بهینه به منطقهٔ دیگری از فضای جست‌وجو وجود دارد. بنابراین چنین مسائلی با چالش‌های بیشتری نسبت به حالت ایستا مواجه می‌شوند و این نوع مسائل از جنس مسائل سخت محسوب می‌گردند. در واقع مسائل بهینه‌سازی پویا از کلاس NP-hard هستند. بهینه نمودن این مسائل نیاز به الگوریتمی غیرقطعی و فرااکتشافی دارد. این الگوریتم باید قادر باشد در شرایطی که محیط با عدم قطعیت روبه‌روست، کارایی مناسبی از خود نشان داده و بهینهٔ متغیر را ردیابی کند. برای اثبات اینکه این مسائل از نوع NP-hard هستند به مراجع [۱ـ۵ و ۲۹ـ۳۵] رجوع شود. ازاین‌رو محققان بر آن شدند که از الگوریتم‌هایی بهره گیرند که بتوانند خود را با شرایط متغیر محیطی وفق دهند. به این ترتیب توجه آن‌ها به‌سمت الگوریتم‌های تکاملی معطوف شد؛ زیرا این الگوریتم‌ها از تکامل طبیعی الهام گرفته و تکامل طبیعی نیز فرایندی پیوسته از سازگاری با محیط است.

در پژوهش‌های انجام‌شده تاکنون از سه روش اصلی در الگوریتم‌های تکاملی برای حل مسائل بهینه‌سازی پویا استفاده شده است [۱]: ۱. ایجاد تنوع، ۲. به‌کارگیری حافظه، ۳. رویکرد چندجمعیتی. به‌دلیل امکان ایجاد تغییرات در فضایی که هیچ عضوی در آن حضور ندارد و یا امکان همگرایی اعضا در آن منطقه، از رویکرد ایجاد تنوع استفاده می‌شود. برای تحقق این امر در الگوریتم‌هایی چون الگوریتم ژنتیک، از مهاجران تصادفی و گاهی از فنون خودتطبیقی در نرخ جابه‌جایی مهاجران واردشده به جمعیت بهره گرفته شده است [۲]. در الگوریتم‌های ازدحام ذرات از خودکار سلولی استفاده شده است [۳]. همچنین محققان سعی کردند تا با به‌کارگیری حافظه و استفاده از بهترین اعضای جمعیت بتوانند الگوریتم ژنتیک را با محیط‌هایی که در معرض تغییرات کم قرار می‌گیرند تا حدی سازگار نمایند [۴]. در نمونه‌ای دیگر از رویکرد چندجمعیتی استفاده شده که در آن‌ها از چندین جمعیت در فضای جست‌وجو برای ایجاد تعادل بین اکتشاف[1] و استخراج[2] نقطهٔ (نقاط) بهینه و ردیابی هرچه بهتر آن(ها) بهره گرفته شده است [۵].

در این مقاله، یکی از جدیدترین الگوریتم‌های تکاملی، الگوریتم SSPCO[3] [۶]، برای حل مسائل بهینه‌سازی پویا ارائه می‌شود. کارایی این الگوریتم برای حل مسائل ایستا به اثبات رسیده، ولی این الگوریتم تابه‌حال برای مسائل پویا آزمایش نشده است. این الگوریتم همان طور که از نامش پیداست، از نحوهٔ زندگی پرنده‌ای به نام تیهو الهام گرفته شده است. در این مقاله با ارائهٔ یک الگوریتم مبتنی بر SSPCO، سعی شده است تا یک الگوریتم مناسب و کارا برای حل مسائل بهینه‌سازی پویا ارائه شود و این روش بتواند به‌نحو مطلوبی چالش‌های اساسی ذکرشده برای مسائل پویا را حل نماید. روش ارائه‌شده در این مقاله ترکیبی از الگوریتم SSPCO آشوب‌گونه، چندجمعیتی و حافظه است. باید خاطرنشان کرد که تابه‌حال رفتار پرندهٔ تیهو و جوجه‌هایش برای حل مسائل بهینه‌سازی پویا ارائه نشده است. بسیاری از رفتارهای طبیعی در جهان واقعی به‌صورت آشوب‌گونه است. پدیده‌های تصادفی[4] که تاکنون با عجز و ناتوانی، دلیلی برای آن‌ها نمی‌یافتیم، به‌کمک نظریهٔ آشوب، توجیه می‌شوند. نظریهٔ آشوب بر پایه‌های ریاضی، فیزیک و حتی فلسفه استوار است؛ هریک از این علوم با ابزارهای خود این نظریه را بررسی و اثبات کرده‌اند؛ بنابراین برای ایجاد جمعیت اولیه در روش پیشنهادی از آشوب استفاده می‌شود [۷]. در بسیاری از مسائل بهینه‌سازی پویا حالت فعلی محیط اغلب شبیه به حالات دیده‌شدهٔ قبلی می‌باشد. استفاده از اطلاعات گذشته ممکن است به سیستم کمک کند تا با تغییرات بزرگ در محیط بهتر تطبیق پیدا کند و در طول زمان بهتر اجرا شود. یک راه برای حفظ و بهره‌برداری از اطلاعات گذشته استفاده از حافظه است. بنابراین در روش پیشنهادی برای ایجاد تنوع

1. Exploration
2. Exploitation
3. See-See Partridge Chicks Optimization
4. Random

حداکثری در فضای مسئله از حافظه استفاده شده است. در یک محیط پویا چندین قله وجود دارد که پس از تغییر در محیط هریک از این قله‌ها ممکن است به بهینه تبدیل شوند. بنابراین الگوریتم باید بتواند تا جای ممکن این قله‌ها را پوشش دهد تا هنگامی که یکی از این قله‌ها به بهینه تبدیل شد تعدادی از ذرات در اطراف آن قرار گیرند. در صورتی که تعدادی از این قله‌ها بدون پوشش باشند، ممکن است بعد از تغییر محیط به بهینه تبدیل شوند که این امر باعث پایین آمدن دقت و کارایی الگوریتم می‌شود. یکی از راه‌حل‌هایی که برای این مشکل وجود دارد، استفاده از روش چندجمعیتی است. از آنجایی که حل مسائل بهینه‌سازی در محیط‌های پویا کار آسانی نیست، در این پژوهش با آزمایش‌های مختلف سعی شده تا یک مسئلهٔ مهم را که شبیه‌ساز مناسبی برای مسائل بهینه‌سازی پویاست (محک قله‌های متحرک) را حل نمود.

یکی از اهداف اصلی این مقاله، ارائهٔ رویکردی جدید برای حل مسائل بهینه‌سازی پویا با استفاده از ترکیب الگوریتم SSPCO با راهکارهای دیگر است. روش پیشنهادی به نام MMSSPCO[1] در این مقاله نام‌گذاری شده است. این الگوریتم برای مسائل بهینه‌سازی ایستا مورد آزمایش قرار گرفته و نتایج مطلوبی به همراه داشته است [۶]؛ اما تابه‌حال رفتار این الگوریتم برای حل مسائل بهینه‌سازی پویا مورد آزمایش قرار نگرفته است. از آنجایی که تمامی پژوهش‌های پیشین اثبات کرده‌اند که یک الگوریتم تکاملی استاندارد به‌تنهایی قادر به حل مسائل بهینه‌سازی پویا نیست [۱]، در این مقاله روش استاندارد SSPCO با دو روشِ حافظه و چندجمعیتی ترکیب می‌شود. در این روش از یک راهکار جدید برای تولید زیرجمعیت‌ها به‌صورت تطبیقی استفاده شده است. همچنین از آنجایی که ظرفیت حافظهٔ استاندارد برای ذخیره‌سازی اطلاعات محدود می‌باشد، از یک حافظهٔ جدید با عنوان حافظهٔ تخمین تراکم با خوشه‌بندی گوسی استفاده شده است.

اهدف اصلی در محیط‌های پویا و به‌دنبال آن مسائل بهینه‌سازی پویا ردیابی نقطهٔ (نقاط) بهینه و سپس یافتن هرچه دقیق‌تر آن(ها) است. ازاین‌رو مسئله‌ای که حائز اهمیت است سرعت ردیابی و دنبال کردن بهینه(ها) است؛ به‌عبارتی بایستی بتوان قبل از ایجاد تغییر محیطی بعدی تخمین هرچه بهتری از جواب(های) بهینه داشت. به همین دلیل سعی بر آن است تا با بهره‌گیری از الگوریتم‌های هوشمند و تکاملی که الهام گرفته از تکامل طبیعی‌اند، بتوان اهداف مطرح‌شده را برآورده کرد.

این مقاله شامل ۵ بخش است: بخش اول به مقدمه و کلیات موضوع می‌پردازد. بخش دوم مقاله مروری بر کارهای گذشته دارد. بخش سوم روش پیشنهادی را تشریح می‌کند. بخش چهارم به ارزیابی روش پیشنهادی و نتایج تجربی می‌پردازد. بخش پنجم نتیجه‌گیری و کارهای آتی را بیان می‌کند.

۲. ادبیات تحقیق و مرور کارهای گذشته

در این بخش، ابتدا به معرفی مسائلی که روش پیشنهادی این مقاله با آن‌ها سروکار خواهد داشت و سپس به مرور کارهای گذشته در زمینهٔ مسائل بهینه‌سازی پویا پرداخته می‌شود.

۲. ۱. محیط‌های پویا

محیط‌های پویا که به نام مسائل پویا[2] یا مسائل وابسته به زمان[3] شناخته می‌شوند، محیط‌هایی هستند که در طول زمان می‌توانند شاهد تغییرات پیوسته یا گسسته در خود باشند. این تغییرات می‌تواند در حوزهٔ وسیعی اتفاق بیفتد. از جمله اینکه در این محیط‌ها، تعداد، ابعاد، دامنهٔ پارامترهای محیطی و یا دیگر ویژگی‌ها می‌تواند تغییر کند. از دیگر مواردی که ممکن است اتفاق بیفتد، تغییر مقدار پارامترها با توجه به زمان است. در نهایت در کلیهٔ تغییرات ایجادشده، محیط با تغییر نقطه یا نقاط بهینهٔ سراسری و محلی روبه‌رو می‌شود. مسائلی که در این حوزه تعریف می‌شوند، هم در بخش آزمایشگاهی و هم در دنیای واقعی به این محیط‌ها عینیت می‌بخشند. از جملهٔ آن‌ها، مسائل بهینه‌سازی پویا[4] بوده که در آن‌ها مقدار تابع برازندگی مرتب تغییر می‌کند. به‌طور دقیق‌تر در تعریف ریاضی این گونه مسائل خواهیم داشت [۱].

1. Multi-swarm Memory SSPCO

2. Dynamic problems
3. Time-dependent problems
4. Dynamic Optimization Algorithm (DOP)

$$DOP = \begin{cases} optimize\ f(x,t) \\ s.t.\ \ x \in F(t) \subseteq S, t \in T \end{cases} \quad (۱)$$

که در آن $S \in R^n$ فضای جست‌وجو، t زمان جست‌وجو، $f(x,t) \in R$ تابع شایستگی که مقادیر عددی $f: S \times T \to R$ را به هر راه‌حل ممکن $(x \in S)$ در زمان t نسبت می‌دهد و $F(t)$ مجموع راه‌حل‌های ممکن $x \in F(t) \subseteq S$ در زمان t می‌باشد.

۲.۲. تغییرات پیوسته و ناپیوسته

از حیث تغییرات ایجادشده در نقطهٔ (نقاط) بهینهٔ تابع هدف، محیط می‌تواند شاهد دو نوع تغییر پیوسته یا ناپیوسته باشد [۱]. اگر f تابع هدف ایستا در نظر گرفته شود، با اعمال تغییرات خواهیم داشت: $f'(t,x) = f\ (x - \Delta(t))$ که در آن بردار جابه‌جایی $\Delta(t)$ مسیر حرکت بهینه[1] را کنترل می‌کند. بنابراین در صورتی که x^* بهینهٔ تابع f باشد $x^*(t) = x^* + \Delta(t)$، بهینهٔ f' در زمان t خواهد بود. حال $\Delta(t)$ می‌تواند به‌طرق مختلف تعریف شود: اگر $\Delta(t) = \delta.t$ باشد، مسیر حرکت بهینه به‌صورت خطی و اگر $\Delta(t) = \delta.\sin(2\pi.t/T)$ باشد، مسیر حرکت بهینه به‌طور تناوبی خواهد بود که در آن‌ها شدت تغییر به‌وسیلهٔ مقدار δ تعیین می‌شود. در شرایط ناپیوسته نقطهٔ بهینه از یک موقعیت به یک موقعیت دیگر پرش می‌کند. چنین رفتاری به‌وسیلهٔ توابع چندحالته[2] ایجاد می‌شود. به‌طور کلی مسائل با تغییرات ناپیوسته به‌مراتب سخت‌تر از تغییرات پیوسته هستند که در این مقاله، تغییرات به‌صورت ناپیوسته اعمال می‌شود.

۳.۲. نظریهٔ آشوب

در حالت عادی، آشوب به‌معنای حالتی از بی‌نظمی است. طبق تعریف معمولی آشوب، سیستم‌های پویا که در ردهٔ سیستم‌های آشوبی دسته‌بندی می‌شوند باید ویژگی‌های زیر را دارا باشد [۷]:

- سازگاری پویا[3]: در محیط در حال تغییر امروزی سازمان سیستم‌های بی‌نظم در ارتباط با محیط اطرافشان مانند موجودات زنده عمل می‌کنند که برای رسیدن به موفقیت همواره باید خلاق و در جست‌وجوی راه‌های جدیدی برای پیروزی و پیشرفت باشند؛ اما زمانی که سیستم به تعادل سازگار نزدیک می‌شود، برای حفظ پویایی نیاز به تغییرات اساسی درونی دارد که این تغییرات به‌جای سازگاری و تطبیق با محیط، موجب سازگاری پویا می‌شود که نتیجهٔ آن دگرگونی روابط پایدار بین افراد، الگوهای رفتاری، الگوهای کار، نگرش‌ها و فرهنگ‌هاست.
- خودمانایی[4]: در تئوری آشوب، هر جزئی از سیستم دارای ویژگی کل بوده و مشابه آن است. دنیس گابور در سال ۱۹۴۸ برای اولین بار هولوگرافی را بدین گونه بیان کرد: جزء خاصیت کل را دارد و مانند آن عمل می‌کند. سیستم توانایی یادگیری را دارد؛ سیستم دارای توانایی خودسازمان‌دهی است؛ حتی اگر قسمت‌هایی از سیستم برداشته شود سیستم به‌راحتی می‌تواند به فعالیت خود ادامه دهد. خاصیت خودمانایی در رفتار اعضای سازمان نیز می‌تواند نوعی وحدت ایجاد کند؛ همهٔ افراد به یک سو و یک جهت و هدف واحدی نظر دارند.
- حساس بودن به شرایط اولیه[5]: غیرقابل پیش‌بینی بودن رفتار در سیستم‌های آشوب‌گونه، تابع دو پدیده است: اولی مربوط به حساسیت نسبت به شرایط اولیه است و دوم اینکه تغییرات شدید رفتارهای نامنظم و دگرگونی‌های غیرقابل پیش‌بینی حرکات، همه در درون خود بی‌نظمی نهفته هستند.

در اکثر الگوریتم‌های فراابتکاری با اجزای تصادفی، تابع توزیع احتمال استفاده شده برای تولید اعداد تصادفی، تابع توزیع احتمال گاوسی[6] یا یکنواخت[7] می‌باشد[۸]. با جایگزینی مقادیر حاصل از نگاشت آشوب به‌جای این مقادیر تصادفی نتایج بهتری حاصل شده و علی‌رغم تصادفی بودن اعداد می‌توان به قطعی بودن نتایج حاصل اطمینان داشت [۹]. در جدول ۱ نگاشت‌های مورد نیاز در نظریه آشوب برای استفاده در این مقاله آورده شده است. نکته حائز اهمیت این است که نگاشت مناسب برای مسائل مختلف با توجه به فرکانس و دامنه موج آن

1. Optimum trajectory
2. Multimodal functions
3. Dynamic Addaptation
4. Self-Similarity
5. Strange Attractors
1. Gaussian
2. Uniform

نگاشت، می‌تواند متفاوت باشد، در مسئله مدنظر دامنه و فرکانس نگاشت انتخابی باید به‌گونه‌ای باشد که تمام فضای جست‌وجو مسئله را در تعداد تکرار کم پوشش دهد [۱۰] زیرا ممکن است الگوریتم با تعداد ذرات کمی شروع به کار کند و این مسئله به جست‌وجوی فضای مسئله و همگرا شدن به جواب بهینه کمک کند.

جدول (۱): نگاشت‌های آشوب

نام نگاشت	بازه	فرمول
چبیشف	(0,1)	$x_{k+1} = \cos(k.\cos^{-1}(x_k))$
تکراری	(0,1)	$x_{k+1} = \sin\left(\frac{\alpha\Pi}{x_k}\right),\ \ \alpha \in (0,1)$
لجستیک	(0,1)	$x_{k+1} = \alpha x_k(1-x_k),\ \ x \in (0,1),\ \ \alpha = 4$
دایره	(0,1)	$x_{k+1} = x_k + b - \left(\frac{\alpha}{2\Pi}\right).\sin(2\Pi x_k)\ mod\ e(1)$
لیبویتچ	(0,1)	$x_{k+1} = \begin{cases} \frac{\alpha x_k}{p_2 - p_1} & 0 < x_k < p_1 \\ \frac{p_2 - x_k}{p_2 - p_1} & p_1 < x_k < p_2 \\ 1 - \beta(1-x_k) & p_2 < x_k < 1 \end{cases}$

۲. ۴. چندجمعیتی

دستاورد دیگری که تا حدی به‌عنوان ترکیبی از نگهداری تنوع، حافظه و خودسازگاری استفاده می‌شود، پیاده‌سازی چندین زیرجمعیت به‌طور هم‌زمان است. هر زیرجمعیت ممکن است یک منطقهٔ جدایی از فضای جست‌وجو را در دست گرفته و یک وظیفهٔ جدا از همی را انجام دهند. برای مثال بعضی از زیرجمعیت‌ها ممکن است روی جست‌وجوی بهینهٔ سراسری و بعضی دیگر روی دنبال کردن تغییرات محیطی متمرکز شوند. سپس این دو نوع جمعیت برای تعادل در فرایند جست‌وجو با هم تبادل اطلاعات می‌کنند. روش‌هایی که از روش چندجمعیتی بودن پیروی می‌کنند، بایستی زیرجمعیت‌ها را به‌گونه‌ای تقسیم‌بندی کنند که با یکدیگر هم‌پوشانی نداشته باشند. به این ترتیب تنوع خوبی در کل فضای جست‌وجو ایجاد شده و شرایطی مهیا می‌شود که چندین زیرجمعیت برای پیدا کردن قله‌های یکسان وارد عمل نشوند. این موارد در ضمن ارائهٔ راهکارهای استفاده از چندجمعیتی بودن قابل مشاهده می‌باشند.

۲. ۵. حافظه

زمانی که تغییرات در مسائل پویا به‌طور بازگشتی و تناوبی صورت گیرد، نقطهٔ بهینهٔ جدید ممکن است به مناطقی که نزدیک به موقعیت‌های قبلی بهینه است برگردد. بنابراین استفادهٔ مجدد از نقاط قبلی می‌تواند در صرفه‌جویی زمان محاسباتی مفید واقع شود. ازاین‌رو به‌کارگیری حافظه به‌عنوان یک راهکار مناسب برای دنبال کردن بهینه پیشنهاد می‌شود. همچنین با ذخیره کردن راه‌حل‌های مناسب می‌تواند در حفظ تنوع جمعیت نقش بسزایی ایفا کند. به‌طور کلی دو نوع حافظه در حل مسائل می‌توان در نظر گرفت: حافظهٔ ضمنی[1] [۱۱] و حافظهٔ صریح[2] [۱۲].

- **حافظهٔ ضمنی:** در این نوع از حافظه سعی می‌شود از تعدادی اطلاعات افزونه در فضای کدگذاری مسئله استفاده شود؛ به‌گونه‌ای که راه‌حل‌های خوب مربوطه ذخیره و به‌موقع مورد استفادهٔ مجدد قرار گیرند.
- **حافظهٔ صریح:** این نوع از حافظه که بیشتر مورد استفاده قرار گرفته، از یک فضای حافظه‌ای جدا برای ذخیرهٔ اطلاعات مفید استفاده می‌کند.

برای انجام روش‌های مطرح‌شده در این نوع حافظه چهار سؤال زیر مطرح می‌شوند:

۱. چه اطلاعاتی را در حافظه ذخیره کنیم؟
۲. حافظه چگونه به‌روزرسانی شود؟
۳. چه زمان حافظه به‌روزرسانی شود؟
۴. چگونه اطلاعات حافظه را بازیابی کنیم؟

برای پاسخ‌گویی به این سؤالات به بررسی اجمالی هریک پرداخته می‌شود.

دربارهٔ اینکه چه اطلاعاتی در حافظه ذخیره شود، دو دسته حافظه وجود دارد: حافظهٔ مستقیم[3] و حافظهٔ انجمنی[4]. در حافظهٔ مستقیم بهترین راه‌حل‌های قبلی ذخیره شده و در حافظهٔ انجمنی علاوه بر راه‌حل‌های خوب گذشته، تعدادی اطلاعات محیطی (مانند اطلاعات آماری توزیع جمعیت در زمان مشخص شده، لیستی از وضعیت‌های محیطی و...) نیز ذخیره می‌شوند [۱۲].

با توجه به اینکه فضای حافظه محدود است، نیاز به تعدادی

1. Implicit memory
2. Explicit memory
3. Direct memory
4. Associative memory

سازوکار برای به‌روزرسانی حافظه احساس می‌شود. برای این کار یک اصل اساسی این است که ذره‌ای از حافظه برای حذف شدن یا به‌روزرسانی توسط بهترین ذره در جمعیت، انتخاب شود. به این‌ترتیب میزان شایستگی ذرات ذخیره‌شده از میانگین شایستگی بالاتر شده و حافظه به‌روز خواهد شد. چندین روش برای جایگزینی افراد حافظه وجود دارد:

۱. جابه‌جایی کمترین ذرات از نظر درجهٔ اهمیت (می‌تواند درجهٔ اهمیت بر اساس سن افراد، سهیم بودن در تنوع جمعیتی یا میزان شایستگی و... باشد) با باارزش‌ترین ذرات جمعیت.

۲. جابه‌جایی ذره‌ای از حافظه که کمترین نقش در واریانس حافظه را دارد.

۳. جابه‌جایی شبیه‌ترین ذره به بهترین عضو جمعیت در صورتی که بهترین ذره شایستگی بهتری داشته باشد.

۴. محاسبهٔ کمینهٔ فاصلهٔ بین جفت ذرات حافظه و جابه‌جایی کمترین میزان شایستگی بین این جفت‌ها.

دربارهٔ اینکه حافظه چه موقع به‌روزرسانی شود، حالت ایدئال زمانی است که بدانیم تغییرات چه موقع اتفاق می‌افتد. دقیقاً پس از اتفاق افتادن تغییرات حافظه به‌روزرسانی می‌شود. ولی در حالت کلی نمی‌توان گفت دقیقاً در چه زمانی تغییرات اتفاق می‌افتد. بنابراین می‌توان گفت حافظه بعد از هر نسل یا بعد از یک تعداد مشخص نسل یا حتی بعد از گذشت یک تعداد تصادفی از نسل‌ها به‌روزرسانی شود.

در بررسی اطلاعات بازیابی‌شده از حافظه یک راه ساده این است که بهترین ذره یا ذرات از حافظه بازیابی شده و با کم‌ارزش‌ترین ذره یا ذرات از نظر مقدار شایستگی در جمعیت جابه‌جا شوند. این کار می‌تواند در هر نسل یا در تعداد نسل‌های مشخص یا تنها زمانی که تغییرات محیطی رخ می‌دهند، اتفاق افتد.

۲.۶. تابع مولد محیط‌های پویا

تابع محک قله‌های متحرک [۱۳] یکی از توابع معروف برای شبیه‌سازی محیط‌های پویاست. این تابع در سال ۱۹۹۹، توسط برانک پیشنهاد شد و به‌طور گسترده‌ای در مقالات مورد استفاده قرار گرفته است [۱۳]. در این مسائل، نقطهٔ بهینه توسط تغییر در موقعیت، ارتفاع و عرض قله‌ها تغییر می‌کند. برای یک فضای D بعدی مسئله به‌صورت زیر تعریف می‌شود [۱۳]:

$$F(\vec{x},t) = max_{i=1\dots P}\ P_{sh}(\vec{x}, h_i(t), w_i(t), \vec{\rho}_i(t))) \quad (۲)$$

$P_{sh}(.)$ تابعی است که شایستگی یک نقطهٔ داده شده را برای یک قلهٔ مشخص‌شده توسط ارتفاع (h_i) و طول (w_i) و موقعیت راس (ρ_i) توصیف می‌کند. در هر θe بار ارزیابی تابع کارایی ارتفاع، طول و موقعیت برای هر قله تغییر می‌کند، این تغییر حالت محیط است.

ارتفاع و پهنای هر قله توسط متغیرهای تصادفی گاوسی (σ) تغییر یافته و توسط پارامترهای شدت ارتفاع[1] (h_{sev}) و شدت پهنا[2] (w_{sev}) مقیاس‌دهی می‌شوند. موقعیت شیفت داده شده از یک متغیر طولی S_l و یک فاکتور همبستگی λ استفاده می‌کند.

طول حرکت کنترل می‌کند که قله چقدر دور شود، فاکتور همبستگی مشخص می‌کند چگونه حرکت تصادفی قله انجام شود. اگر $\lambda = 0$، حرکت قله به‌صورت کاملاً تصادفی خواهد بود، اما اگر $\lambda = 1$، آنگاه قله همیشه به یک سمت یکسان حرکت می‌کند تا زمانی که به یک مرز فضای مختصات می‌رسد، که مسیر آن همانند یک شعاع از نور منعکس می‌شود. این کار تا رسیدن به مرز فضای مورد نظر ادامه می‌یابد که در این صورت مانند توپ بیلیارد به داخل محیط برمی‌گردد. در زمان تغییر در محیط، تغییرات در یک قله می‌تواند به‌صورت رابطه (۳) توصیف شده باشد [۱۳].

$$\begin{cases} h_i(t) &= h_i(t-1) + height_{severity}.\ \sigma \\ w_i(t) &= w_i(t-1) + width_{severity}.\sigma \\ \vec{p}_i(t) &= \vec{p}_i(t-1) + \vec{v}_i(t) \\ \sigma &\in N(0,1) \end{cases} \quad (۳)$$

بردار انتقالی $v_i(t)$ ترکیبی از یک بردار تصادفی $\vec{r}$ با یک بردار انتقال قبلی $v_i(t-1)$ است. در صورتی‌که پارامتر همبستگی λ به مقدار صفر تنظیم شود، جابه‌جایی قله‌ها بدون ارتباط با یکدیگر و به‌طور مستقل انجام می‌شود. بردار تصادفی ساخته‌شده توسط رسم یکنواخت از $[۰،۱]$ برای هر بعد و سپس مقیاس‌دهی بردار برای داشتن طول s_l است که بر اساس رابطه

1. Height Severity
2. Width Severity

(۴) می‌باشد [۱۳]. جابه‌جایی هر قله به‌صورت زیر تعریف می‌شود:

$$\overrightarrow{v_l} = \frac{S_l}{\vec{r} + \overrightarrow{v_l}}((1-\lambda)\vec{r} + \lambda\vec{v}(t-1)) \quad (۴)$$

تابع قله برای ارتفاع، عرض و موقعیت هر قله به‌صورت رابطه (۵) محاسبه می‌شود [۱۳].

$$P(\vec{x}, h(t), w(t), \vec{p}(t)) = h(t) - w(t)\sqrt{\sum_{j=1\ldots n}(x_j - p_j)^2} \quad (۵)$$

بخش مربوط به جذر، فاصلهٔ بین نقطهٔ موجود و موقعیت هر قله را بیان می‌کند [۱۳].

۲.۷. معیار ارزیابی کارایی

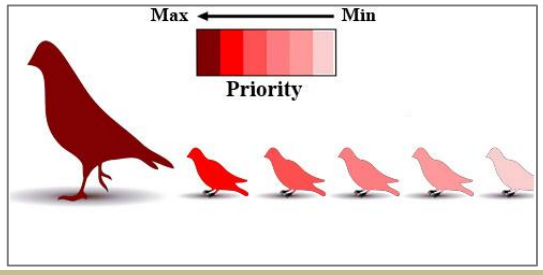
برای سنجش کارایی الگوریتم‌ها از معیار خطای برون‌خطی[1] استفاده می‌شود که برابر است با میانگین اختلاف بین مقدار شایستگی بهترین عضو پیداشده توسط الگوریتم و مقدار شایستگی بهینهٔ سراسری [۱۴]. در واقع به این اختلاف، خطای جاری[2] نیز گویند. بنابراین به تعبیری دیگر خطای برون‌خطی، میانگین همهٔ خطاهای جاری در زمان t است. فرمول ریاضی خطای برون‌خطی در رابطهٔ (۶) مشاهده می‌شود [۱۴]:

$$Offline\ Error = \frac{1}{FE_s}\sum_{t=1}^{FE_s}(f\ (best)(t) - f(gOpt)(t)) \quad (۶)$$

که در آن FE_s بیشینهٔ تعداد ارزیابی‌های تابع، $best$ و $gOpt$ به‌ترتیب بهترین موقعیت‌های پیداشده توسط الگوریتم و بهینهٔ سراسری در t-امین ارزیابی شایستگی است. شایان ذکر است که مقدار خطای برون‌خطی عددی نامنفی بوده و در حالت ایده‌ال صفر می‌باشد.

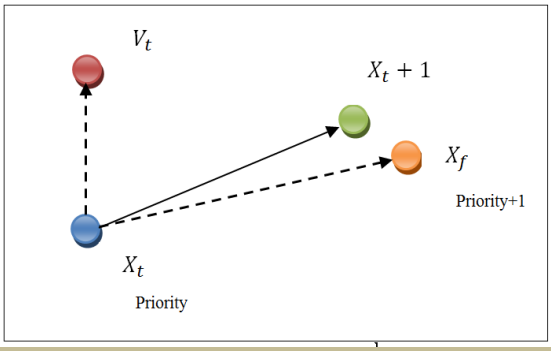
۲.۸. الگوریتم SSPCO

الگوریتم SSPCO [۶] از رفتار جوجه‌های یک نوع پرندهٔ وحشی به نام تیهو[3] الهام گرفته شده است. جوجه‌های این نوع پرنده در هنگام احساس خطر برای رسیدن به یک نقطهٔ امن در یک صف منظم قرار گرفته و درست در یک صف منظم پشت سر پرندهٔ مادر شروع به حرکت می‌کنند تا به یک نقطهٔ امن برسند. جوجه‌ها با بهره گرفتن از تجربهٔ موفق جوجه‌ای که در جلوی آن‌ها در حال حرکت هستند، به‌سمت منطقهٔ امن حرکت می‌کنند. در این الگوریتم سعی خواهد شد یک صف بر اساس شایستگی مکانی ذرات رو به‌سمت جواب بهینه شکل بگیرد، و این الگوریتم ذرات را در فضای جست‌وجو وادار خواهد کرد که برای رفتن به مکانی بهتر به‌دنبال ذره‌ای بروند که در این صف یک گام از آن‌ها جلوتر در حال حرکت است. برای این کار با تنظیم یک معادلهٔ حرکت و سرعت برای ذرات، ذرات آنقدر در صف‌های گوناگون قرار می‌گیرند و از آن جدا می‌شوند تا به بهترین صفی که آن صف حتماً به‌سمت جواب بهینه در حال حرکت است ملحق می‌شوند و لذا ذره‌ای که در آخر به‌جلو صف و نزدیک‌ترین ذره به جواب بهینه باشد، پرندهٔ مادر خواهد بود.



شکل (۱): نحوهٔ تنظیم حرکت در الگوریتم SSPCO [۶]

بر طبق شکل، هر جوجه در فضای جست‌وجو به‌دنبال جوجه‌ای با اولویت یک واحد از خودش بالاتر می‌گردد و سعی می‌کند معادله حرکت خود را بر اساس این جوجه تنظیم کند. مقدار متغیر اولویت عددی است که پس از مقداری حرکت ذره‌ها در فضای جست‌وجو باعث می‌شود تا ذرات در یک خط همگرایی منظم به‌سمت بهینهٔ عمومی حرکت کنند.



شکل (۲): نحوهٔ حرکت یک ذره در الگوریتم SSPCO [۶]

در شکل (۲)، ذرهٔ t حرکت بعدی خود را بر اساس دو

1. Offline error
2. Current error
3. See-See Partridge

مؤلفهٔ ذرهٔ f که دارای اولویتی با یک واحد بالاتر از خود است، و سرعت خود که با V_t نشان داده شده است، تنظیم می‌کند. یک متغیر برای هر ذره به نام متغیر اولویت به‌نام $X_i.priority$ در نظر گرفته می‌شود [۶].

$$\text{if } X_i.cost > P_{best} \rightarrow P_{best} = X_i.position \quad and \ X_i.priority = X_i.priority + 1 \qquad (۷)$$

$X_i.cost$ هزینهٔ هر ذره در تابع هزینه است، P_{best} بهترین تجربهٔ شخصی هر ذره می‌باشد و $X_i.position$ مکان هر ذره است. در هر بار ارزیابی، اگر بهینهٔ محلی بهتر از بهینهٔ عمومی و بالعکس باشد، متغیر اولویت ذره بالاتر می‌رود و یک واحد به آن طبق رابطهٔ (۸) اضافه می‌شود [۶]:

$$\text{if } P_{best} > G_{best} \rightarrow G_{best} = P_{best} \quad and \ X_i.priority = X_i.priority + 1 \qquad (۸)$$

در رابطهٔ (۸)، G_{best} بهینهٔ عمومی است. معادلهٔ حرکت هر ذره تقریباً شبیه به الگوریتم تودهٔ ذرات به شکل رابطه (۹) تنظیم می‌شود [۶]:

$$X_i.position = X_i.position + X_i.velocity \qquad (۹)$$

در رابطهٔ (۹)، $X_i.velocity$ سرعت هر ذره یا جوجه است. حال معادلهٔ سرعت ذره طبق رابطهٔ (۱۰) محاسبه می‌شود [۶]:

$$X_i.velocity = w * X_i.velocity + c * rand(0,1) * [position(X_{i+1}.priority)] - X_i.position \qquad (۱۰)$$

$X_i.velocity$ سرعت ذره، w ضریب تأثیرگزاری سرعت قبل در معادلهٔ سرعت کنونی ذره، c فاکتور اجتماعی بوده که در الگوریتم تودهٔ ذرات نیز آمده است، $rand()$ یک عدد تصادفی با توزیع یکنواخت در بازهٔ صفر و یک، $[position(X_i.priority+)]$ موقعیت مکانی ذره با اولویت یکی بالاتر از ذرهٔ کنونی می‌باشد که ذرهٔ کنونی سعی می‌کند سرعت خود را بر اساس این ذره تنظیم کند و $X_i.position$ نیز مکان فعلی ذره می‌باشد. مشاهده می‌شود طبق معادلهٔ (۱۰) هر ذره دقیقاً حرکت خود را بر اساس ذره‌ای با اولویت یکی بالاتر از خود تنظیم می‌کند، بدین صورت که دیگر کاری به بهینه‌های محلی و عمومی ندارد و در هر لحظه از زمان ذرات فقط و فقط به دنبال ذره‌ای حرکت می‌کنند که در فضای جست‌وجو از آن ذره یک واحد جلوتر باشد؛ به همین سبب تعداد و زمان محاسبات در این الگوریتم نسبت به الگوریتم‌های گذشته بهینه‌سازی دارای سودمندی بالایی می‌باشد. شبه کد الگوریتم SSPCO در شکل (۳) آورده شده است.

SSPCO Algorithm

```
//initialize all chicken
1. Initialize
2. repeat
3.   foreach chicken i
       //update the chicken's best position and priority
4.     if f(x_i) > f(pb_i) then
5.       pb_i = x_i
6.       prioirity_i = prioirity_i + 1
7.     endif
       //update the global best position and priority
8.     if f(pb_i) > f(gb) then
9.       gb = pb_i
10.      prioirity_i = prioirity_i + 1
11.    endif
12. endfor
    //update chicken's velocity and position
13.  foreach chicken i
14.    foreach dimension d
15.      v_{i,d} = v_{i,d} + c * rand(0,1) * [position(priority_{i+1})] - x_{i,d}
16.      x_{i,d} = x_{i,d} + v_{i,d}
17.    endfor
18.  endfor
     //advance iteration
19.  it = it + 1
20. until it > MaxIterations
```

شکل (۳): شبه کد الگوریتم SSPCO [۶]

۲،۹. مروری بر کارهای گذشته

در این بخش به مروری بر کارهای گذشته که در زمینهٔ بهینه‌سازی پویا توسط محققان صورت گرفته، پرداخته می‌شود. نویسندگان در مرجع [۱۵] یک الگوریتم ژنتیک آشوب‌گونه با ترکیب حافظه برای حفظ راه‌حل‌های مناسب و خوشه‌بندی جمعیت در داخل حافظه و جمعیت اصلی ارائه کرده‌اند. در این روش برخلاف الگوریتم ژنتیک استاندارد که جمعیت اولیه به‌صورت تصادفی ایجاد می‌شود، از تئوری آشوب برای ایجاد جمعیت اولیه استفاده شده است. در این روش از خوشه‌بندی مبتنی بر میانگین برای افزایش تنوع استفاده شده است. در مقالهٔ دیگری، مرجع [۱۶] روشی دیگر برای بهینه‌سازی پویا را که مبتنی بر حافظهٔ صریح است، ارائه کرده‌اند. در این روش از راهکاری مناسب برای به‌روزرسانی و بازیابی اطلاعات از حافظه استفاده شده است. نحوهٔ به‌کارگیری حافظه در این روش باعث شده تا افراد جمعیت در فضای مسئله به‌نحو مطلوبی دارای پراکندگی (تنوع) باشند و به‌سرعت به قله‌های متحرک همگرا

شوند. این روش نیز ترکیب الگوریتم ژنتیک با حافظه است. هدف اصلی این روش، حفظ تنوع در حین اجرای الگوریتم با استفاده از حافظه است. یانگ[1] در سال ۲۰۰۷ نسخهٔ بهبودیافته‌ای از به‌کارگیری مهاجران در الگوریتم ژنتیک را ارائه کرد که در آن از مهاجران ایجادشده بر پایهٔ بهترین فرد نسل قبلی برای ایجاد تنوع استفاده کرد تا بتواند بر مشکلات حاصل از اعمال مهاجران تصادفی فائق آید [۴]. الگوریتم حاصل به نام EIGA[2] می‌باشد. در این الگوریتم پس از آنکه عملگرهای ژنتیکی بر افراد جمعیت اعمال می‌شوند، در هر نسل، بهترین فرد نسل قبلی انتخاب و بر اساس آن به تعداد $n \times r_{ei}$ تا مهاجر تولید می‌گردند. r_{ei} نسبت تعداد مهاجران تولیدشده به‌اندازهٔ جمعیت و n اندازهٔ جمعیت است. این مهاجران براساس اعمال عملگر جهش بیتی[3] با احتمال P_m^i بر روی بهترین فرد نسل قبلی ایجاد شده و سپس افراد تولیدشده به‌جای $n \times r_{ei}$ تا از بدترین افراد جمعیت جاری قرار می‌گیرند. در DMGA [۱۷] از حافظه‌ای به اندازهٔ $m = 0.1 * n$ استفاده شده که به‌طور تصادفی مقداردهی می‌شود (n اندازهٔ جمعیت است). زمانی‌که حافظه می‌خواهد به‌روزرسانی شود، ابتدا بررسی می‌شود که آیا این نقاط تصادفی اولیه هنوز در حافظه وجود دارند یا خیر. در صورت وجود آن‌ها، بهترین فرد جمعیت با یکی از نقاط حافظه که به‌طور تصادفی انتخاب شده، جابه‌جا می‌شود. در غیر این صورت از روش نزدیک‌ترین فرد به بهترین فرد جمعیت بهره گرفته و در صورت بهتر بودن بهترین فرد جمعیت نسبت به آن، جابه‌جایی صورت می‌گیرد. گفتنی است که به‌روزرسانی در یک الگوی زمانی تصادفی اتفاق می‌افتد. در سال ۲۰۰۸ الگوریتم دیگری توسط لی و یانگ ارائه شد که این روش، بر مبنای الگوریتم بهینه‌سازی تجمعی ذرات بوده و از روش چنددسته‌ای شدن ذرات[4] استفاده می‌کند [۱۸]. در اینجا دسته‌ها به دو نوع تقسیم می‌شوند: ۱. یک گروه والد، ۲. چندین گروه فرزند. گروه والد برای نگهداری تنوع در طول اجرا و شناسایی مناطق امیدبخش با استفاده از برنامه‌ریزی تکاملی سریع[5] وارد عمل شده و گروه‌های فرزندان برای جست‌وجوی محلی با استفاده از الگوریتم بهینه‌سازی تجمعی ذرات سریع[6] به کار گرفته می‌شوند. شایان ذکر است که برنامه‌ریزی تکاملی سریع از عملگر جهش cauchy به‌جای گوسین استفاده می‌کند. بنابراین طول گام بلندتر در رسیدن به بهینهٔ سراسری برداشته و قدرت استخراج فضای جست‌وجو را به‌طور کارا دارد [۱۹]. الگوریتم FMSO نیز قدرت همگرایی سریع داشته و مناسب جست‌وجوی محلی است. در بررسی هم‌پوشانی گروه‌ها باید گفت در صورتی که دو گروه یک منطقه را جست‌وجو کنند، همان طور که قبلاً نیز اشاره شد، باعث کاهش کارایی الگوریتم و از دست رفتن منابع می‌شوند. برای عدم هم‌پوشانی بین گروه فرزندان اگر فاصلهٔ دو تا گروه از هم کمتر از شعاع تعریف‌شده در [۱۸] بود، در آن صورت گروه فرزند با شایستگی کمتر حذف می‌شود. در این مقاله می‌توان به این موضوع اشاره کرد که الگوریتم پیشنهادی ارائه‌شده با معیار خطای برون‌خطی برای محیط‌هایی با تغییرات خیلی زیاد مفید واقع شده و با این معیار توانسته قله‌های با ارتفاع زیاد را ردیابی کند. نویسندگان [۲۰] از مفهوم چندجمعیتی در الگوریتم PSO استفاده کرده‌اند. در اینجا از رویکرد وزن تطبیقی[7]، خوشه‌بندی فازی[8] و اجرای جست‌وجوی محلی بر روی بهترین گروه استفاده شده است. برای دنبال کردن بهینه هم از ذرات موج‌گونه بهره برده‌اند. در اینجا دو نوع گروه وجود دارد: آزاد و همگرا. در ابتدا گروه‌ها به‌طور تصادفی ایجاد شده که هر گروه شامل تعدادی ذرات PSO است. در هر مرحله، موقعیت و سرعت ذرات به‌روزرسانی می‌شود. در این الگوریتم از همسایگی کوچک ذرات برای افزایش تنوع و کاهش سرعت همگرایی استفاده می‌شود. نویسندگان [۲۱] نسخهٔ جدیدی از الگوریتم گروه ماهی‌های مصنوعی برای بهینه‌سازی محیط پویا ارائه کردند. در این الگوریتم که به نام DMAFSA[9] می‌باشد، پارامترها، رفتارها و روال الگوریتم گروه ماهی‌های مصنوعی برای سازگاری با محیط

1. Yang
2. Elitism-based immigration Genetic Algorithm
3. Bit-wise mutation
4. Multi-swarm
5. Fast Evolutionary Programming (FEP)
6. Fast Particle Swarm Optimization (FPSO)
7. Adaptive weight
8. Fuzzy clustering
9. Dynamic Modified Artificial Fish Swarm Algorithm

پویا اصلاح شده است. از جمله رفتارهای ماهی‌ها جست‌وجوی غذا، حرکت گروهی و دنباله‌روی است. در گام نخست برای پویاسازی، این سه رفتار ماهی‌ها نسبت به الگوریتم AFSA استاندارد به‌طور متمایز مورد استفاده قرار گرفته و تحت عنوان الگوریتم MAFSA[1] مطرح می‌شوند. در مقالهٔ دیگری، نویسندگان [۲۲] از یک حافظهٔ صریح برای افزایش کارایی الگوریتم ژنتیک در حل مسائل بهینه‌سازی پویا استفاده کردند. آن‌ها در این روش توانستند با به‌کارگیری یک حافظهٔ صریح و همچنین روش مناسبی برای ذخیره و بازیابی اطلاعات از این حافظه به مدل کارایی در حل مسائل بهینه‌سازی پویا دست یابند.

۳. روش پیشنهادی

در این بخش، روش پیشنهادی با جزئیات تشریح خواهد شد. همان گونه که قبلاً نیز اشاره شد، روش پیشنهادی در این مقاله ترکیبی از الگوریتم **SSPCO** آشوب‌گونه، چندجمعیتی و حافظه است. هر کدام از این موارد در بخش قبل مفصل تشریح شد. حال در این بخش نحوهٔ ترکیب الگوریتم **SSPCO** آشوب‌گونه، چندجمعیتی و حافظه تشریح می‌شود. روش پیشنهادی شامل ۶ مرحلهٔ اساسی است که در ادامهٔ این مراحل تشریح شده‌اند.

مرحلهٔ ۱، تولید جمعیت اولیهٔ ذرات بر مبنای تئوری آشوب

الگوریتم‌های تکاملی الهام‌گرفته از پدیده‌های طبیعی هستند؛ این الگوریتم‌ها از مولدهای تصادفی برای تولید جمعیت اولیه استفاده می‌کنند ولی در طبیعت پدیده‌ها تصادفی نیستند و از نظم خاصی پیروی می‌کنند. در این روش از پدیدهٔ آشوب به‌منظور تولید اعداد شبه‌تصادفی در الگوریتم پیشنهادی استفاده شده است. آشوب نوعی بی‌نظمی منظم است؛ بی‌نظمی از آن رو که نتایج آن غیرقابل پیش‌بینی است و منظم بدان سبب که از نوعی قطعیت برخوردار است. در روش پیشنهادی، ابتدا جمعیت اولیه (هر پرندهٔ تیهو یک ذره از جمعیت در نظر گرفته می‌شود) مطابق تئوری آشوب و بر اساس روابط ذکرشده در جدول (۱) تولید می‌شوند.

مرحلهٔ ۲، چندجمعیتی نمودن ذرات

در این مرحله، ذرات به‌جای یک جمعیت، درون چندین زیرجمعیت تقسیم می‌شوند. همان گونه که در بخش‌های قبلی نیز بیان شد، یکی از چالش‌های اساسی در مسائل بهینه‌سازی پویا ایجاد تنوع بعداز همگرا شدن ذرات است. بنابراین همان گونه که در بخش‌های قبل نیز ذکر شد، روش چندجمعیتی برای ایجاد تنوع در محیط بسیار مناسب است. باید خاطرنشان کرد که در این روش همانند روش پیشنهادی در مرجع [۲۴] ذرات به دو دسته ذرات خنثی و ذرات کوانتومی تقسیم می‌شوند. ذرات خنثی وظیفهٔ همگرایی سریع به بهینهٔ مورد نظر را بر عهده دارند. درحالی‌که ذرات کوانتوم بنا به ماهیت کوانتومی که دارند، وظیفهٔ ایجاد تنوع (پراکندگی) در فضای مسئله را بر عهده دارند. در این روش ابتدا جمعیت اولیه به تعداد از پیش تعیین‌شده‌ای زیرجمعیت تقسیم می‌شود. درون هر زیرجمعیت چندین ذره خنثی و ذرات کوانتومی همانند [۲۴] وجود دارد. یکی از زیرجمعیت‌ها به‌عنوان زیرجمعیت پیشاهنگ در نظر گرفته می‌شود و این زیرجمعیت ابتدا در فضای مسئله غیرفعال است. در روش پیشنهادی ابتدا ذرات خنثی موجود در هر زیرجمعیت موقعیت خود را به‌صورت روابط ذکرشده در ۹ و ۱۰ به‌روز می‌کنند و سپس موقعیت $gbest$ هر زیرجمعیت با استفاده از ذرات کوانتوم در چندین مرحله بهبود داده می‌شود. هر ذره کوانتوم به تعداد £ بار می‌تواند موقعیت $gbest$ را بهبود دهد. فرض شود موقعیت هر ذره کوانتوم در اطراف $gbest$ از زیرجمعیت j با $x_{i,j}$ نشان داده می‌شود. حال موقعیت هر ذره کوانتوم از رابطهٔ (۱۲) به دست می‌آید.

$$x_{i,j} = G(t)_i + (R_j \times r_{cloud}) \qquad (۱۲)$$

در رابطهٔ (۱۲)، $x_{i,j}$ موقعیت ذرهٔ کوانتوم j از زیرجمعیت i می‌باشد. R_j یک عدد تصادفی از توزیع یکنواخت در بازه $[0,1]$ می‌باشد. بنابراین موقعیت هر ذرهٔکوانتوم در هریک از ابعاد فضای جست‌وجو می‌تواند در شعاع کوانتوم از اطراف موقعیت $gbest$ باشد. بعد از محاسبهٔ موقعیت ذرات کوانتوم مقدار شایستگی برای هر ذره محاسبه می‌شود. حال اگر مقدار شایستگی ذرهٔ کوانتومی از مقدار بهینهٔ سراسری $gbest$ بیشتر

1. Modified Artificial Fish Swarm Algorithm

شود، در این صورت موقعیت ذرهٔ کوانتوم به‌عنوان بهینهٔ سراسری در نظر گرفته می‌شود، در غیر این صورت موقعیت تغییری نمی‌کند. رابطهٔ (۴) به‌اندازهٔ £ بار تکرار می‌شود تا بهینهٔ سراسری بهبود یابد. شبه کد مربوطه به‌صورت شکل (۴) است.

Improve gbest Algorithm

```
//update G(t) with Quantum particles
foreach swarm i
 for c=1 to £ do
  Update Qi,j based on equation 1
  if f(Q_{i,j}) > f(G(t)_i) then
    G(t)_i= Q_{i,j}
 endfor
endfor
```

شکل (۴): شبه کد مربوط به بهبود بهینهٔ سراسری توسط ذرات کوانتوم

چندجمعیتی به‌صورت تطبیقی تعدادی از زیرگروه‌های مورد نیاز را تنظیم کرده و به‌صورت خودکار نواحی را برای هر زیرگروه جست‌وجو می‌کند. در حقیقت در این روش برخلاف روش‌های مشابه که تعداد زیرجمعیت‌ها ثابت می‌باشند (تعداد زیرجمعیت‌ها از ابتدا مشخص شده و ثابت هستند)، زیرجمعیت‌ها طبق یک سازوکار تطبیقی تولید می‌شوند. از آنجایی که تعداد قله‌ها از قبل معلوم نیست، تعداد زیرجمعیت‌ها (M) نمی‌تواند به‌راحتی قابل تعیین باشد. به این دلیل یک سازوکار تطبیقی برای تعداد زیرجمعیت‌ها ارائه شده است. در این روش، ابتدا جمعیت اولیه به‌صورت آشوب‌گونه ایجاد می‌شود، سپس این جمعیت به تعدادی زیرجمعیت با تعداد مشخصی تقسیم می‌شود. حال اگر تمام زیرجمعیت‌های فعلی به قلهٔ مورد نظر خود همگرا شده باشند و در صورتی که بعضی از قله‌ها هنوز بدون پوشش باقی مانده باشند، دستهٔ پیشاهنگ در فضای مسئله فعال خواهد شد. اگر فقط یک قله بدون پوشش وجود داشته باشد، دستهٔ پیشاهنگ به این قله همگرا می‌شود و مشکل حل شده و نیاز به جست‌وجوی بیشتری نیست. اما اگر بیش از یک قله بدون پوشش وجود داشته باشد، دستهٔ پیشاهنگ باید به‌عنوان یک دستهٔ یابنده عمل کرده و بتواند مکان این قله‌ها را کشف کرده و به دسته‌های ایجادشدهٔ جدید اطلاع‌رسانی کند. باید روشی وجود داشته باشد که دستهٔ پیشاهنگ بتواند در فضای مسئله فعال شود. این روش در ادامه شرح داده شده است.

فرض کنید n تعداد کل قله‌های موجود در فضای مسئله باشد و c قله توسط ذرات پوشش داده شده‌اند. حال دستهٔ پیشاهنگ مطابق رابطهٔ (۱۳) عمل می‌کند:

$$\begin{cases} \forall i = 1 \dots n \;\; if \; n - c > 0 \; then \; Scout_{swarm} = 1 \\ \forall i = 1 \dots c \;\; if \; n - c \leq 0 \; then \; Scout_{swarm} = 0 \end{cases} \quad (۱۳)$$

در رابطهٔ (۱۳)، $Scout_{swarm} = 1$ بدین معنی است که دستهٔ پیشاهنگ در فضای مسئله فعال است و $Scout_{swarm} = 0$ بدین معنی است که دستهٔ پیشاهنگ در فضای مسئله غیرفعال است. باید خاطرنشان کرد که دسته‌های جدید بر اساس مقدار شایستگی دستهٔ پیشاهنگ درمی‌یابند که این دسته یک قله را پیدا کرده یا خیر. برای مقدار شایستگی یک مقدار آستانه[1] برای دستهٔ پیشاهنگ در نظر گرفته می‌شود. اگر میانگین شایستگی اعضای دستهٔ پیشاهنگ بزرگ‌تر یا مساوی این مقدار آستانه گردد، بدین معنی است که باید یک دستهٔ جدید ایجاد شود (دستهٔ پیشاهنگ یک قله را پیدا کرده است). این موضوع به‌صورت رابطهٔ (۱۴) بیان می‌شود.

$$\text{if } mean\ FE_s\ for\ Scout_{Swarm} \geq threshold \quad (۱۴)$$
$$\text{then } creat\ new\ Swarm$$

پیدا کردن یک آستانهٔ مناسب برای مقدار شایستگی می‌تواند به بهبود کارایی الگوریتم کمک شایانی کند. راهکار مورد نظر برای دستهٔ پیشاهنگ بدین صورت است که این دسته طبق شعاع دفع [۲۲] دیگر نمی‌تواند در محدودهٔ دسته‌های دیگر قرار گیرد. بنابراین این دسته دیگر فضاهای مسئله را که خارج از شعاع دفع دیگر دسته‌هاست ردیابی می‌کند. این موضوع می‌تواند به افزایش سرعت ردیابی دستهٔ پیشاهنگ کمک شایانی کند. در صورتی که این دسته یک قله را پیدا کند، اعضای دستهٔ جدید را به‌سمت آن سوق داده و هنگامی که این دسته همگرا شدند، دستهٔ پیشاهنگ دوباره به جست‌وجوی خود در خارج از محدودهٔ دیگر دسته‌ها ادامه می‌دهد. زیرجمعیت پیشاهنگ زمانی که یک قله را ردیابی کرد و زیرجمعیت جدید را به‌سمت آن سوق داد، به‌صورت تصادفی (خارج از شعاع دفع دیگر دسته‌ها) در فضای مسئله قرار می‌گیرد. این کار تا زمانی ادامه می‌یابد که تمام محدودهٔ فضای مسئله مورد کاوش این دسته قرار گرفته باشد. برای تعیین

1. Threshold

همگرایی یک دسته، فاصلهٔ اقلیدسی تمام ذرات یک دسته از یکدیگر محاسبه شده و در صورتی که فاصلهٔ دورترین ذرات دسته کوچک‌تر از مقدار پارامتر r_{conv} [۲۰] (که بر اساس آزمایشات برابر ۱۰ است) باشد، یعنی دستهٔ همگرا شده است. فاصلهٔ اقلیدسی از رابطهٔ (۱۵) به دست می‌آید:

$$Dist = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N}(particle_i - particle_j)^2} \qquad (۱۵)$$

که $\boldsymbol{N}$ تعداد ذرات خنثی در هر دسته هستند.

این روش می‌تواند جانشین مناسبی برای عملگر ضد همگرایی باشد که در مرجع [۲۰] آورده شده است. در عملگر ضد همگرایی کل فضای مسئله باید توسط بدترین دسته به‌صورت کاملاً تصادفی مجدداً جست‌وجو شود؛ این موضوع زمان جست‌وجو و کارایی را تا حدودی تحت‌تأثیر خود قرار می‌دهد. اما در روش تطبیقی پیشنهادی، فقط فضاهایی از مسئله جست‌وجو می‌شوند که تابه‌حال مورد جست‌وجو قرار نگرفته‌اند و در نتیجه زمان کمتری برای جست‌وجو مورد نیاز است. شکل (۵) شبه کد مربوط به نحوهٔ چندجمعیتی را در روش پیشنهادی نشان می‌دهد.

Algorithm: multi-swarm function

$\boldsymbol{p}$: *is peak*
$\boldsymbol{n}$: *number of peaks*
$\boldsymbol{c}$: *covered peaks by* sub_{swarm}
$\boldsymbol{r_{excl}}$: *is radius exclusion for each sub_swarm*
$\boldsymbol{t}$: *threshold*
$\boldsymbol{FEs}$: *Fitness Evaluations*

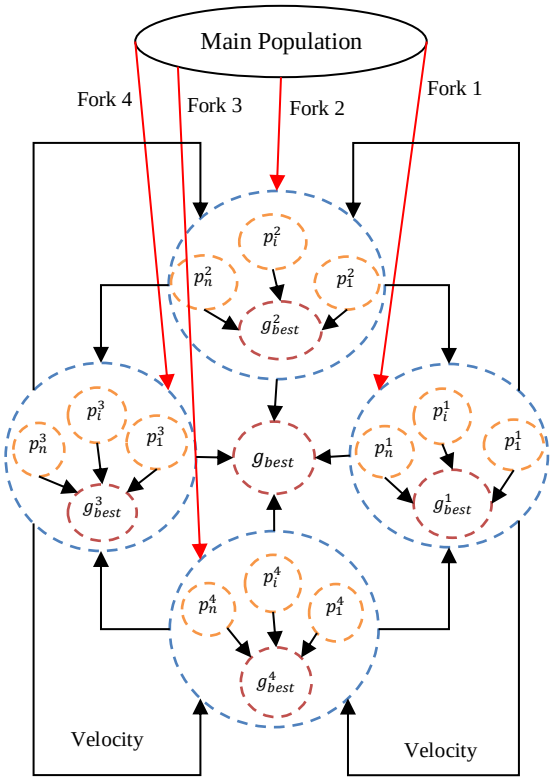
1. Initialization the *parent* $_{swarm}$
2. Initialization the sub_{swarm}
3. Initialization the $Scout_{swarm}$
4. $Scout_{swarm}$ *is inactive*
5. sub_{swarm} *is randomly in the search space*
6. $\forall\, i = 1 \dots n\,, k = 1 \dots n,$
$r_{excl}(sub_{swarm_i}) \neq r_{excl}(sub_{swarm_k})$
7. $\begin{cases} \forall i = 1 \dots n & \text{IF } n - c > 0 \text{ THEN} \\ & Scout_{swarm} = 1 \;\%\; n > sub_{swarm} \\ \forall j = 1 \dots c & \text{IF } n - c \leq 0 \text{ THEN} \\ & Scout_{swarm} = 0 \;\%\; n \leq sub_{swarm} \end{cases}$
8. IF *mean FEs for* $Scout_{swarm} \geq t$ THEN *create new* $_{swarm}$

شکل (۵): شبه کد نحوهٔ ایجاد چندجمعیتی در روش پیشنهادی

چالش دیگری که در بهینه‌سازی محیط پویا وجود دارد، همگرایی دو دسته به یک قله است. در این حالت، وجود یکی از این دو دسته غیرمفید و حتی مضر می‌باشد؛ زیرا دستهٔ اضافی با مصرف منابع محاسباتی و ارزیابی شایستگی، موجب کند شدن الگوریتم می‌شود. برای این منظور از روش انحصار [۲۰] استفاده می‌شود. در این روش، فاصلهٔ اقلیدسی دو موقعیت بهترین ذره تمام دسته‌های موجود در فضای مسئله از یکدیگر محاسبه شده و اگر فاصلهٔ بهترین ذرهٔ دو دسته کمتر از یک مقدار مشخص r_{excl} [۲۰] باشد، دسته‌ای که بهترین ذره آن از موقعیت بدتری نسبت به موقعیت بهترین ذره دستهٔ دیگر برخوردار باشد، به‌طور کلی از بین می‌رود. فاصلهٔ انحصاری از رابطهٔ زیر به دست می‌آید:

$$r_{excl} = 0.3 \times (100/(Peak_{number}{}^{1/Dimension})) \qquad (۱۶)$$

در آن $\boldsymbol{Peak\_number}$ تعداد قله‌ها و $\boldsymbol{Dimension}$ تعداد ابعاد مسئله‌اند. نحوهٔ تولید زیرجمعیت‌ها در این روش پیشنهادی در شکل (۶) آورده شده است.



شکل (۶): روش چندجمعیتی و تولید زیرجمعیت‌ها

مرحلهٔ ۳، کشف تغییر در محیط

در این مرحله، باید روشی وجود داشته باشد که بتواند تغییر در محیط را تشخیص دهد. اگر لحظهٔ تغییر در محیط برای الگوریتم مشخص نباشد، الگوریتم نمی‌تواند به‌سرعت به تغییرات رخ‌داده واکنش نشان داده و بهینهٔ مورد نظر را ردیابی

کند. روش‌های مختلفی برای کشف تغییر در محیط وجود دارد که کاملاً وابسته به نوع تغییرات در محیط‌اند. در این مقاله، هدف حل مسائلی است که تغییر در محیط آن‌ها به‌صورت سراسری است؛ یعنی مقدار شایستگی تمام نقاط موجود در محیط تغییر می‌کند. از آنجایی که شرط اجرای تمام رفتارها بر اساس مقدار شایستگی ذرات است، باید مجدداً پس از تغییر محیط مقدار شایستگی تمام ذرات محاسبه و به‌روز شود تا بتوانند به‌درستی رفتارها را پس از تغییر محیط انجام دهند. بر همین اساس می‌توان تغییر در محیط را کشف کرد. در واقع برای آزمایش کردن اینکه تغییر محیط رخ داده یا خیر، می‌توان شایستگی یکی از ذرات را مجدداً ارزیابی کرد. در صورتی که مقدار شایستگی به‌دست‌آمده برابر مقدار شایستگی ثبت‌شده برای ذرهٔ مورد نظر باشد، یعنی محیط تغییر نکرده است؛ در غیر این صورت محیط تغییر کرده است. شبه کد مربوط به کشف تغییر در محیط به‌صورت شکل (۷) آمده است.

Algorithm: Change Detecting

$Evaluate\ f(p_{ng})$
if *new value is different from last iteration* **then**
Re_evaluate each particle
$ChangFlag;$
Evaluate a particle from each swarm
if *new value is different from last iteration* **then**
$Chang\_Flag = 1;$
else
$ChangFlag = 0;$

شکل (۷): شبه کد شناسایی لحظهٔ تغییر در محیط

مرحلهٔ ۴، حل مسئلهٔ حافظهٔ نامعتبر ذرات

در این مرحله، بعد از تغییر در محیط لازم است تا حافظهٔ ذرات به‌روزرسانی شود. اطلاعات ذرات بعد از تغییر در محیط ممکن است نامعتبر باشند. بنابراین بعد از تغییر در محیط باید حتماً حافظهٔ نامعتبر ذرات به‌روزرسانی شود. پس از کشف تغییر در محیط، دسته‌ها با دو چالش مهم برخورد می‌کنند که اولی حافظهٔ نامعتبر و دومی تنوع ازدست‌رفته در دسته است. چالش اول از آنجا به وجود می‌آید که مقادیر ذخیره‌شده در حافظه به‌عنوان مقدار شایستگی هر ذره، در محیط جدید تغییر کرده و باید به‌روز شوند. همچنین مشکل دوم یعنی از دست دادن تنوع در دسته از آنجا ناشی می‌شود که هدف ذرات الگوریتم‌های فرامکاشفه‌ای، همگرا شدن به‌سمت یک هدف است. در واقع با کاهش گستردگی دسته و جمع شدن ذرات در اطراف هدف، تنوع در دسته کاهش یافته و این امر باعث افزایش توانایی جست‌وجوی محلی و رسیدن به موقعیت‌های بهتر می‌شود. با وجود این پس از تغییر محیط، هدف جابه‌جا شده و دسته با توجه به کاهش شدید تنوع در پیش از تغییر محیط، توانایی حرکت به‌سمت موقعیت جدید هدف را از دست داده و یا اینکه حرکت به‌سمت آن بسیار کند انجام می‌شود. برای مقابله با این دو چالش، در الگوریتم پیشنهادی پس از کشف تغییر در محیط ابتدا موقعیت بهترین ذره تمام دسته‌هایی که همگرا شده‌اند (قله‌هایی را تحت پوشش دارند) تعیین می‌شود. سپس تمام ذرات دسته در شعاعی در اطراف این موقعیت به‌صورت تصادفی پخش می‌شوند. اندازهٔ این شعاع برابر رابطهٔ (۱۷) است:

$$Radius = 0.5 \times Shift\_severity \quad (۱۷)$$

که در آن، $Shift\_Severity$ همان طول گام حرکتی قله‌هاست. مقدار این شعاع بر اساس مقدار گام حرکت قله‌ها مشخص شده که می‌توان آن را به‌راحتی بر اساس موقعیت‌های به‌دست‌آمده در قله‌ها در دو محیط متوالی به دست آورد. بدین ترتیب تنوع ازدست‌رفته به‌اندازه‌ای که لازم است به دسته‌ها بازگردد. پس از آن، مقدار شایستگی ذرات در محیط جدید محاسبه شده و در حافظه جایگزین می‌شوند. بنابراین مشکل حافظهٔ نامعتبر نیز از بین می‌رود.

مرحلهٔ ۵، به‌کارگیری حافظهٔ پیشنهادی

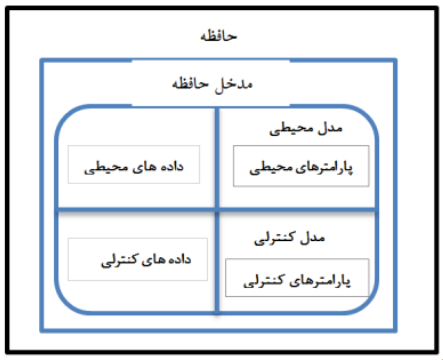
در این مرحله برای افزایش کارایی الگوریتم از یک حافظهٔ صریح برای نگهداری اطلاعات گذشته استفاده می‌شود. همگرایی جمعیت می‌تواند باعث کاهش سازگاری الگوریتم در مسائل پویا گردد که در این صورت حافظه به ایجاد تنوع در جمعیت کمک می‌کند. از زمانی که یک حافظه اقدام به حفظ تنوع جمعیت می‌کند، جمعیت‌های همگرا نیز اقدام به قبول و وفق‌پذیری با این تنوع می‌کنند. برخی از فنون افزایش تنوع، مقادیر منحصربه‌فرد غیرقابل استفاده‌ای را تولید می‌کنند، درحالی‌که تنوع ایجادشده به‌وسیلهٔ حافظه می‌تواند به‌صورت بالقوه مورد استفاده قرار گیرد. حافظه‌های صریح با میزان سربار حافظهٔ کمتری ارائه می‌شوند. این نوع حافظه در جست‌وجو

بسیار غنی هستند به‌طوری که اگر قسمتی از زمان محاسبات برای جست‌وجو را به حافظه اختصاص دهیم، حافظه با بهینه‌سازی زمان، بهره‌وری را افزایش می‌دهد. یکی از اساسی‌ترین چالش‌ها در استفاده از حافظهٔ استاندارد، ظرفیت محدود آن‌هاست. برای رفع این چالش باید راهکاری برگزید که بتواند به‌طور مناسب و کارا عملیات ذخیره و بازیابی از حافظه را انجام دهد. برای برخورد با این چالش از یک سازوکار مناسب استفاده شده است. حافظهٔ ارائه‌شده در این مقاله، حافظه‌ای با عنوان حافظهٔ تخمین تراکم با خوشه‌بندی گاوسی است. این حافظه مدل‌های احتمالی درون حافظه را با ساختن تخمین تراکم از نواحی امیدبخش فضای جست‌وجو در طول زمان ایجاد و حفظ می‌کند. حافظهٔ تخمین تراکم با خوشه‌بندی گاوسی اجازه می‌دهد که بسیاری از راه‌حل‌ها در حافظه ذخیره شوند، و مدل‌های طولانی‌مدت از فضای جست‌وجوی پویا به وجود می‌آورد و اجازه می‌دهد تا زمانی که سربار کم است مدخل‌های حافظه به‌راحتی تصحیح شوند. در واقع حافظهٔ استفاده‌شده در این روش نقاط ضعف یک حافظهٔ استاندارد را به‌صورتی که در ادامه تشریح شده، از بین می‌برد:

- ترکیب مدل‌های احتمالی از طریق راه‌حل‌های قبلی داخل حافظه به‌جای ذخیره کردن راه‌حل‌های تکی مجزا؛
- ذخیرهٔ بسیاری از راه‌حل‌های قبلی در حافظه تا زمانی که سربار کم است، در صورتی‌که استفاده از مدل‌های احتمالی این اجازه را می‌دهد تا راه‌حل‌های بیشتری در حافظه ذخیره شود. استفاده از تعداد محدودی از مدل‌های احتمالی این اجازه را می‌دهد که الگوریتم بهینه‌سازی و یادگیری تنها با آن مدل‌ها تعامل داشته باشند و نه با آن راه‌حل‌هایی که آن مدل‌ها را ایجاد کرده است. به‌وسیلهٔ خوشه‌بندی کردن راه‌حل‌های ذخیره‌شده در حافظه تنها تعدادی از مدل‌های احتمالی در حافظه نیاز به بازیابی دارند وقتی که راه‌حل‌های جدید ذخیره می‌شوند.
- ایجاد مدل‌های غنی، طولانی‌مدت از فضای جست‌وجوی پویا در هر زمان با استفاده از ذخیرهٔ راه‌حل‌ها در محیط‌های مختلف و تخمین تراکم از فضای جست‌وجو در هر لحظه. این تخمین تراکم یک مدل طولانی‌مدت از ناحیه‌های امیدبخش از فضای جست‌وجو را معرفی می‌کند.
- تصحیح آسان مدخل‌های حافظه با اضافه شدن راه‌حل‌های جدید به حافظهٔ مدخل‌های حافظه، به‌روزرسانی و جایگزین مدخل‌های قدیمی می‌شوند. حافظه می‌تواند تصحیح شود و مدل‌های غنی‌تری از راه‌حل‌های خوبی را که ممکن است در فضای جست‌وجوی پویا وجود داشته باشد فراهم می‌کند.
- نگاشت راه‌حل‌های قبلی به محیط جاری برای مسائلی که راه‌حل‌های ممکن است منسوخ شده باشند، راه‌حل‌های قبلی می‌تواند به ایجاد راه‌حل‌های جدید در محیط جاری کمک کند حتی اگر راه‌حل‌های قبلی منسوخ و عملی نباشند.
- این الگوریتم با خوشه‌بندی افزایشی راه‌حل‌ها را درون مدخل‌های حافظه مجزا می‌کند. مراکز خوشه به‌عنوان مدل‌ها در حافظه استفاده می‌شوند.
- به هر حال حافظهٔ تخمین تراکم اطلاعات بسیاری را از راه‌حل‌های بیشتری از حافظهٔ استاندارد جمع‌آوری می‌کند، سربار استفاده از حافظه کمتر می‌شود. به‌جای تعامل با هر راه‌حل ذخیره‌شده در حافظه نیز حافظهٔ تخمین تراکم اجازه می‌دهد الگوریتم جست‌وجو تنها با مدل‌های ذخیره در حافظه تعامل داشته باشد. حافظهٔ تخمین تراکم اجازه می‌دهد که مدخل‌های حافظه آسان‌تر از مدخل‌های حافظه در حافظهٔ استاندارد اصلاح شوند.

حافظهٔ تخمین تراکم از الگوریتم توزیع‌شده الهام گرفته شده است [۲۳]. تخمین الگوریتم توزیع توسط یادگیری و نمونه‌برداری احتمالات توزیع از بهترین افراد در آن جمعیت در هر تکرار جست‌وجو می‌کند. تکرار یادگیری و نمونه‌برداری به الگوریتم توزیع اجازه می‌دهد که مدل‌ها را از راه‌حل‌های خوب اصلاح کند. برخلاف حافظهٔ استاندارد که می‌تواند فقط یک مدخل حافظه به‌وسیلهٔ جایگزین کردن آن تغییر بدهد، حافظهٔ تخمین تراکم اجازه می‌دهد آن مدل اصلاح شود. حافظهٔ ترکیبی از نقاط جدید برای اصلاح یک حافظه در مدل احتمالی و ایجاد

یک حلقهٔ بازخوردی بین حافظه و اصول یادگیری یا الگوریتم بهینه‌سازی است؛ چنان‌که نوع (کیفیتی) بازیابی راه‌حل از تصحیح پیشرفت حافظه و اصول الگوریتم می‌تواند زمان زیادی در یافتن بهترین راه‌حل‌ها صرف کند که منجر به ذخیرهٔ بهترین راه‌حل موجود در حافظه می‌شود. از آنجایی که حافظهٔ تخمین تراکم می‌تواند بسیاری از بهترین راه‌حل‌ها به‌وسیلهٔ اصول الگوریتم جست‌وجو را تولید و ذخیره کند. یک حافظهٔ تخمین تراکم نقاط را در تعداد محدودی از مدخل‌های حافظه ذخیره‌سازی می‌کند. هر نقطه از اطلاعات محیطی و اطلاعات کنترل تشکیل شده است. اطلاعات کنترلی عمدتاً راه‌حل‌های یک مسئله را در بر می‌گیرد، درحالی‌که اطلاعات محیطی وضعیت محیط‌های پویا را در زمانی که نقطه ذخیره شده است، شامل می‌شود. در حافظهٔ استاندارد، هر مدخل حافظه می‌تواند یک نقطهٔ تکی را ذخیره کند ولی در حافظهٔ تخمین تراکم، هر مدخل ممکن است تعدادی نقاط را ذخیره کند. یک حافظه ممکن است تا |M| مدخل داشته باشد. هر مدخل اصولاً از مجموعه‌ای نقاط، یک مدل از اطلاعات محیطی در آن نقاط و همچنین، یک مدل از اطلاعات کنترلی در آن نقاط تشکیل شده است. شکل (۶) ساختار یک حافظهٔ تخمین تراکم را نشان می‌دهد. در این مقاله آن مدلی که استفاده شده تا یک مدخل حافظه را نشان دهد، مدل خوشه‌بندی گاوسی افزایشی می‌باشد. این الگوریتم از خوشه‌بندی افزایشی به شکل مدخل‌های حافظه استفاده می‌کند، سپس مدل‌های گاوسی برای هر مدخل حافظه ایجاد می‌کند. این اجرا تخمین تراکم خوبی از راه‌حل‌های قبلی فراهم می‌کند. در نسخهٔ گاوسی، متوسط و کوواریانس همهٔ نقاط موجود در مدخل حافظه به‌جهت توصیف مدل مورد استفاده قرار گرفته می‌شوند. زمانی که مدخل‌ها برای نخستین بار ساخته می‌شوند، ممکن است برای محاسبهٔ کوواریانس معتبر نقاط کافی نباشد. در این حالت، نقاط تصادفی اطراف میانگین به‌صورت موقت برای تکمیل مدل اضافه می‌شوند. در نسخهٔ خوشه‌بندی ساده، فقط میانگین مورد استفاده قرار گرفته می‌شود. از آنجایی که مدخل حافظه شامل نقاط متعددی می‌باشد، فقط مدل محیطی و مدل کنترلی برای برقراری تعامل با آن مدخل لازم است. همچنین به‌سبب اینکه مدخل‌ها از هزاران نقطه تشکیل شده‌اند، این مسئله میزان سربار حافظهٔ تخمین تراکم را پایین نگه می‌دارد. شکل (۸) یک حافظهٔ تخمین تراکم را نشان می‌دهد.



شکل (۸): حافظهٔ تخمین تراکم گوسی افزایشی

حافظهٔ تخمین تراکم از تعداد محدودی از مدخل‌ها ایجاد شده است. هر مدخل شامل مجموعه‌ای از نقاط است که هر نقطه شامل دادهٔ محیطی و دادهٔ کنترلی از زمانی است که آن نقطه ذخیره شده بوده. شکل (۹) نحوهٔ عملکرد متفاوت حافظهٔ تخمین تراکم را از حافظهٔ استاندارد نشان می‌دهد. در این مثال، یک فضای جست‌وجوی یک‌بعدی وجود دارد که شامل چهار قله با شکل‌ها و شایستگی مختلف است. فرض کنید که الگوریتم جست‌وجو سعی دارد تا نقاط علامت‌گذاری شده با خطوط ممتد را طی مدت‌زمانی مشخص درون حافظه ثبت کند. حافظهٔ استاندارد نقطه با بهترین شایستگی را انتخاب می‌کند؛ ازاین‌رو همیشه نقطهٔ مزبور نزدیک‌ترین نقطه به مرکز حافظه است. حافظهٔ تخمین تراکم نقاط را بر اساس قله خوشه‌بندی می‌کند و آنگاه به‌ازای هر خوشه یک مدل می‌سازد. در بالای شکل یک مدل گاوسی به‌ازای هر قله نشان داده شده است. پیرامون این مثال، مقدار میانگین محاسبه‌شده به‌ازای هر خوشه شایستگی بیشتری دارد و نزدیک‌ترین نقطه به قلهٔ بیشینه است. فراتر از استفاده از هندسهٔ اقلیدسی، مدل‌های گاوسی اطلاعات بیشتری را به هنگام افزودن یک نقطهٔ جدید به حافظه ارائه می‌کنند. قله در قسمت چپ به حد کافی بزرگ است و هرچه نقاط از حد متوسط دورتر باشند، احتمال پذیرفته شدن آن‌ها نیز بیشتر است. خوشه‌هایی که تابه‌حال مدل دقیقی برای آن‌ها ارائه نشده است، مدل می‌بایست ضمن افزودن نقاط به حافظه اصلاح

گردند. چنان‌چه مدل‌ها دقیق‌تر از قبل شوند، راه‌حل‌های بازیابی‌شده از حافظه نیز بهتر خواهند شد. الگوریتم جست‌وجو در راستای بهبود راه‌حل‌های بازیابی‌شده از حافظه عمل می‌کند، این حلقه بازخورد نیز در جهت بهبود مدل‌های ذخیره‌شده در حافظه گام برمی‌دارد.

در واقع حافظه‌های تخمین تراکم با خوشه‌بندی گاوسی، مدل‌های تخمین تراکم غنی‌تری برای استفادهٔ حافظه فراهم می‌کند. در این روش، هر مدخل حافظه یک مدل گاوسی از نقاط در آن مدخل را محاسبه می‌کند. در این نوع از حافظه، ماتریس میانگین و کوواریانس برای محاسبهٔ اینکه نقطهٔ جدید به خوشهٔ موجود متعلق است یا نه و همچنین برای محاسبهٔ احتمال اینکه دو خوشه می‌بایست ادغام شوند، استفاده می‌شود.

- **ذخیره‌سازی راه‌حل‌ها در حافظهٔ تخمین تراکم**

در حافظهٔ استاندارد، یک نقطه اصولاً با استفاده از راهکار جایگزینی در حافظه ذخیره می‌شود. شایان ذکر است که این رویکرد وظیفهٔ تصمیم‌گیری راجع به نحوهٔ جایگزینی یک نقطه را درون یکی از مدخل‌های حافظهٔ موجود بر عهده دارد. اکثر راهکارهای جایگزینی تنوع در حافظه را حفظ می‌کنند. برای مثال، راهکار جایگزینی ابتدا یک نقطهٔ جدید به حافظه اضافه می‌کند، سپس دو مدخل حافظه نزدیک به یکدیگر پیدا کرده و آن را که از شایستگی کمتری برخوردار است، حذف می‌کند. به غیر از مدخل‌های ازبین‌رفته، حافظهٔ تخمین تراکم مدخل‌ها را با هم ادغام می‌کند. در وهلهٔ نخست، نقطهٔ جدید اضافه‌شده به حافظه به‌دلیل ایجاد مدخل حافظهٔ جدید مورد استفاده قرار گرفته می‌شود. پس از اینکه مدل‌ها برای این مدخل جدید ساخته شد، دو مدخل که در حافظه بیشتر از همه به یکدیگر شبیه‌اند پیدا شده و ادغام می‌گردند. زمانی که فضای جست‌وجوی جدیدی برای نخستین بار ساخته می‌شود، نقطهٔ جدید نیز برای آغاز مدل‌سازی این ناحیه از حافظه مورد استفاده قرار گرفته خواهد شد. به‌واسطهٔ افزایش نقاط خوشه‌بندی، ناحیه‌های جدید فضای جست‌وجو را می‌توان بسته به تغییرات محیط پویا مدل‌سازی کرد. به‌محضی که خوشه‌ها ادغام شدند، خوشه‌ها متشکل از چندین نقطه خواهند شد. همچنین، چنانچه تراکم آن نقاط افزایش یابد، مدل مناسبی نیز برای برآورد مناسبی از مکان راه‌حل‌های خوب در آن ناحیه ارائه خواهد شد. شایان ذکر است که حافظهٔ تخمین تراکم همانند حافظهٔ استاندارد به‌شدت به الگوریتم بهینه‌سازی یا یادگیری وابسته است. حافظهٔ تخمین تراکم اصولاً مدل‌ها را بر مبنای نقاط ذخیره‌شده در حافظه می‌سازد، ولیکن اگر الگوریتم مزبور راه‌حل‌های ضعیفی را ذخیره کند، مدل‌های موجود در حافظه راه‌حل خوبی را نمایش نخواهند داد. در صورتی که راه‌حل‌های خوبی از سوی الگوریتم معرفی گردد، حافظهٔ تخمین تراکم به الگوریتم کمک خواهد کرد تا ناحیهٔ امیدبخش مورد نظر را کشف کند، ولیکن قبل از آن الگوریتم باید بتواند به‌خوبی عملیات جست‌وجو را در آن ناحیه انجام دهد. زمانی‌که نقاط جدید در حافظه ذخیره می‌شوند تا حد زیادی به الگوریتم‌های بهینه‌سازی یا یادگیری و نوع مسئله بستگی دارد. زمانی که از الگوریتم تکاملی استفاده می‌شود، نقطهٔ جدید ممکن است در هرچند نسل ذخیره گردد. همچنین، موقعی که از الگوریتم یادگیری تقویتی بهره گرفته می‌شود، بهترین راه این است که نقاط جدید در زمان یاد گرفتن راه‌حل‌های مطلوب ذخیره شوند (در ذخیرهٔ نقاط جدید یک بار راه‌حل‌های خوب آموزش ببینند). در بعضی از مسائل هنگامی که تغییرات به‌وضوح قابل رؤیت باشد، بهترین حالت این است که راه‌حل‌ها قبل از اعمال تغییر ثبت شوند. در رابطه با مسائل دیگری که تغییرات در آن به‌صورت تدریجی رخ می‌دهد، بهتر است که نقاط به‌طور دوره‌ای ذخیره شوند. ذخیره‌سازی یک راه‌حل، درست بعد از اعمال تغییر اصلاً کار مطلوبی نیست؛ زیرا هیچ فرصتی برای تطبیق آن وجود ندارد. سربار محاسباتی استفاده از حافظهٔ تخمین تراکم به مسئله و مدل‌های احتمالاتی به‌کارگرفته‌شده در حافظه وابسته است. زمانی‌که نقاط خوشه‌بندی افزایش می‌یابد، انتخاب دو مدخل برای ادغام می‌تواند در $\mathbf{O(m^2)}$ انجام داد. در این حالت، $\mathbf{m}$ تعداد مدخل‌های حافظه تلقی می‌شود. وقتی از مدل‌های گاوسی درون حافظهٔ تخمین تراکم استفاده می‌شود، کواریانس در $\mathbf{O(nd^2)}$ محاسبه می‌گردد؛ که $\mathbf{n}$ تعداد نقاط موجود در مدخل حافظه و $\mathbf{d}$ تعداد ابعاد موجود در داده‌های کنترلی یا محیطی می‌باشد. از

آنجایی که حافظه به چندین خوشه تقسیم می‌شود، **n** عمدتاً کمتر از تعداد کل نقاط موجود در حافظه انتخاب می‌گردد. اگرچه استفاده از حافظه مقداری سربار محاسباتی دارد، زمان لازم برای ارزیابی راه‌حل‌ها به‌طور معمول بر زمان مورد نیاز برای نگهداری حافظه غالب است. از آنجایی که حافظهٔ تخمین تراکم اساساً به ارزیابی راه‌حل‌های اضافی احتیاجی ندارد، در نتیجه سربار حافظهٔ تخمین تراکم تنها به‌واسطهٔ ساخت و نگهداری مدل‌های احتمالاتی موجود در حافظه حاصل می‌شود. در بعضی از مسائلی که داده‌های کنترلی یا محیطی ابعاد بسیار زیادی دارند، ممکن است زمان زیادی برای ساخت حافظه برده شود و قبل از بازیابی از حافظه می‌تواند در جهت بهبود کارایی مؤثر باشد. هرچه ابعاد داده بیشتر باشد، نقاط بیشتری برای ارائهٔ تخمین تراکم خوب مورد نیاز است. در رابطه با این مسائل، بهتر است که ابتدا حافظه ساخته شود تا اینکه به‌صورت خالی باشد.

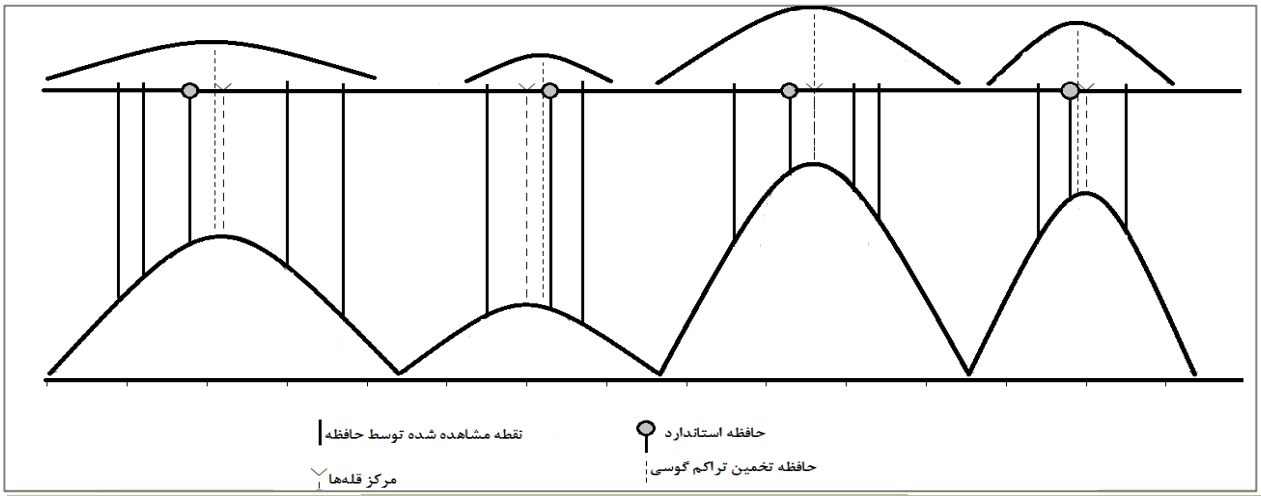
- **بازیابی راه‌حل‌ها از حافظهٔ تخمین تراکم**

راه‌حل‌ها در حافظهٔ تخمین تراکم شبیه به حافظهٔ استاندارد بازیابی می‌شوند. در خصوص الگوریتم‌های جست‌وجوی مبتنی بر جمعیت، راه‌حل‌ها ممکن است از حافظه بازیابی شوند یا در کل اجرا و یا بلافاصله بعد از تغییر شناسایی شوند. زمانی‌که مدخل‌های حافظه تنها به هنگام ذخیره‌سازی نقاط جدید به حافظه تغییر پیدا می‌کنند، راه‌حل‌های ارائه‌شده به‌واسطهٔ بازیابی از حافظه می‌تواند در هر نسل به‌وسیلهٔ الگوریتم جست‌وجو مورد استفاده قرار گیرد. برای الگوریتم‌های یادگیری که تنها یک نقطه را جست‌وجو می‌کنند، راه‌حل‌ها می‌بایست از حافظه با تکرار کمتر و انتخاب بیشتر بازیابی شوند. از آنجایی که بازیابی از حافظه در هر مرحله باعث می‌شود که الگوریتم راه‌حل مربوطه را اصلاح کند، راه‌حل‌ها به‌صورت دوره‌ای یا در هر نسل بازیابی می‌شوند. در مقالهٔ حاضر، راه‌حل حاصل از حافظهٔ زمانی بازیابی می‌گردد که راه‌حل جاری ضعیف عمل کرده و محیط کنونی نیز کاملاً با یکی از مدخل‌های حافظه منطبق است. برخلاف حافظهٔ استاندارد، راه‌حل‌های بازیابی‌شده از حافظه چندان شباهتی به تک‌تک راه‌حل‌های ذخیره‌شده ندارند. به‌واسطهٔ تجمع بسیاری از نقاط موجود در مدخل حافظه، راه‌حل بازیابی‌شده از حافظه تحت‌تأثیر راه‌حل‌های قبلی که در محیط مشابه به‌خوبی عمل می‌کرده‌اند، قرار خواهد گرفت.

شبه کد روش پیشنهادی در شکل (۱۰) آورده شده است. روند نمای روش پیشنهادی در شکل (۱۱) آورده شده است.

مرحلهٔ ۶، شرط خاتمهٔ الگوریتم

در روش پیشنهادی تعداد ارزیابی کارایی برای ذرات، به‌عنوان شرط خاتمه در نظر گرفته شده است.



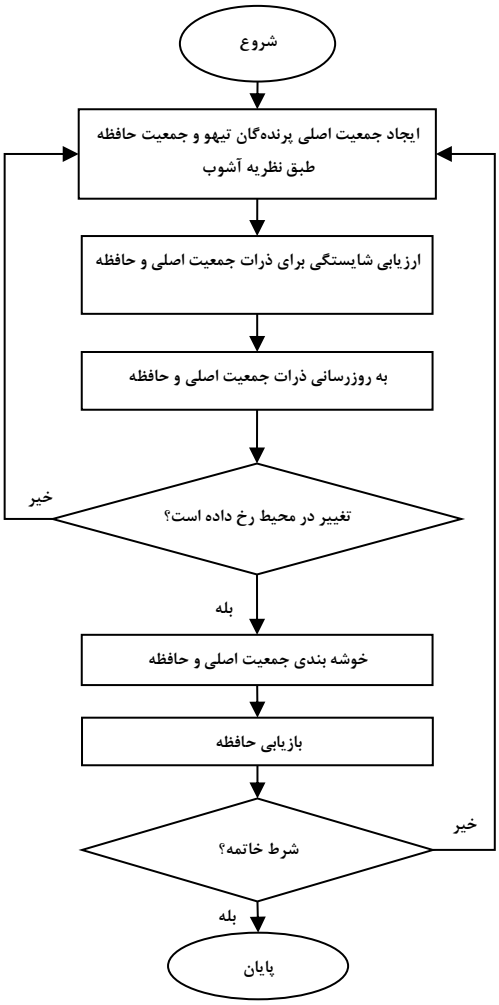
شکل (۹): مقایسهٔ حافظهٔ تخمین تراکم گوسی با حافظهٔ استاندارد

۴. آزمایش‌ها و نتایج تجربی

برای انجام آزمایش‌ها و بررسی کارایی روش پیشنهادی باید از تابع محکی استفاده شود که به‌خوبی بتواند یک محیط پویا را شبیه‌سازی کند. برای آزمایش بر روی روش پیشنهادی و مقایسهٔ آن با سایر روش‌ها از تابع محک قله‌های متحرک برای شبیه‌سازی این الگوریتم‌ها در محیط‌های پویا استفاده می‌شود. در ادامه، تنظیمات مربوط به پارامترهای تابع محک قله‌های متحرک برای ارزیابی الگوریتم پیشنهادی در محیط‌های پویا ذکر می‌شوند.

1. **Input**:
2. MCN: Number of Cycles that algorithm is permitted to go
3. Mem_size: Size of memory
4. N:number of sub-swarm
5. **Output**:
6. BEST Solution, BEST Fitness, Offline Error
7. **Begin**
8. Initialize $Swarm_{\mathrm{N}}$ with chaos map
9. $Swarm_{\mathrm{i}}$ divided to the $Sub_{swarm_{\mathrm{i}}}$with equations (12 and 13)
   % create sub swarm with adaptive strategy
10. $F_i = Fitness(Swarm_i)$
    % F_i is fitness for i-th Particle of Swarm
11. $M_i = Fitness(Mem_i)$
    % M_i is fitness for i-th Particle of memory
12. $UFlag_{UpdateMem} = 0$
    % flag for Update memory
13. $Popflag = 0$
    % flag for Update particles
14. $Cycle = 1$
15. **repeat**
    % TEST FOR COVERD PEAKS
16. **if** all peaks is covered by particles **then**
17. $Scout_{Swarm_i} = 0$
18. **else**
19. $Scout_{Swarm_{\mathrm{i}}} = 1$
    %TEST FOR CHANGE DETECTED
20. **if** (ChangeFlag==1) **then**
21. **if**$\left(\exists i: \left((Fitness(Mem_i) \neq M_i) \vee (Fitness(Pop_i) \neq F_i)\right)\right)$ **then**
22. $ChangeFlag = 1$ *%Change detected*
23. $Update\_Time = rand(5,10) + Cycle$
24. $PopFlag = 1$
25. Update particles into $Swarm_{\mathrm{i}}$ by **SSPCO** Algorithm
26. **if** (ChangeFlag==1) **then** *% We should reuse memory*
27. $UFlag\_UpdateMem = 1$
28. $Mem = Update\_Memory\ (Mem, Swarm, Mem\_size)$
29. **if** $(Update\_Time \geq Cycle)$ **then**
30. $Cycle = Cycle + 1$
31. **Until** $Cycle = MCN$

شکل (۱۰): شبه کد مربوط به روش پیشنهادی



شکل (۱۱): نمودار گردش کار برای روش پیشنهادی

جدول (۲): تنظیمات استاندارد تابع محک قله‌های متحرک

پارامتر	مقدار
تعداد قله‌ها (P)	10
فرکانس تغییرات (f)	5000
Height severity (h_{sev})	7.0
Width severity (w_{sev})	1.0
شکل قله (P_{sh})	Con
Basic function	No
Shift length S_l	1.0
Number of dimensions (D)	5
Correlation coefficient (λ)	0
D_s	[0 ,100]
h_s	[70.0,30.0]
W_s	[12 ,1]
I	50.0

جدول (۳): پارامترهای الگوریتم پیشنهادی

پارامتر	مقدار
کران پایین	۰
کران بالا	۱۰۰
تعداد ذرات جمعیت والد	۱۰۰
اندازهٔ حافظه	۱۰
تعداد اولیهٔ زیرجمعیت‌ها	۱۰
تعداد ذرات هر زیرجمعیت	۱۰
شعاع دفع [۲۰]	۳۱/۵
شعاع انحصار [۲۰]	$0.3 \times (100/(Peak\_number1/Dimension))$
شعاع همگرایی [۲۰]	۱۰

الگوریتم پیشنهادی در محیط نرم‌افزار متلب ۳۰ بار اجرا شده و در تمام آزمایش‌ها تعداد تکرارها برابر $100 \times Change\ Frequency$ در نظر گرفته شده است (تعداد ارزیابی‌ها با تعداد تکرارها در تمام آزمایش‌ها برابرند). جدول (۲) تنظیمات پیش‌فرض مربوط به تابع قله‌های متحرک را نشان می‌دهد که مقادیر آن مبتنی بر سناریوی دوم برانک [۱۳] می‌باشد. جدول (۳) پارمترهای مربوط به الگوریتم پیشنهادی را نشان می‌دهد. نتایج به‌دست‌آمده از سایر الگوریتم‌ها از مرجع مربوط به آن‌ها استخراج شده است. تمامی آزمایش‌ها در محیط نرم‌افزار متلب در ۱۰۰ بار اجرا صورت گرفته است.

جدول (۴) روش‌های مورد مقایسه با روش پیشنهادی را به‌همراه تنظیمات پارامترهایشان نشان می‌دهد. سعی شده در شرایط یکسانی مقایسه انجام بگیرد و اکثریت مقالات به‌صورت پیش‌فرض، پارامترها را به همین اندازه در نظر می‌گیرند.

جدول (۵) روش پیشنهادی را با سایر روش‌ها در شرایط استاندارد سناریوی دوم برانک (جدول ۲) و تعداد قله‌های مختلف مورد مقایسه قرار می‌دهد. نتایج حاصل از این جدول نشان می‌دهد روش پیشنهادی در تمامی حالات نتایج مناسب‌تری را تولید کرده است. این موضوع به‌دلیل حداکثر تنوعی است که در محیط توسط راهکارهای اضافه‌شده به الگوریتم SSPCO به وجود آمده است.

جدول (۶) روش پیشنهادی را با برخی دیگر از روش‌ها در فرکانس‌های تغییرات مختلف، از جمله فرکانس‌های ۵۰۰، ۱۰۰۰، ۲۵۰۰ و ۱۰۰۰۰ مقایسه کرده است. نتایج حاصل‌شده از این جدول، حاکی از برتری روش پیشنهادی نسبت به سایر روش‌هاست.

جدول (۷) روش پیشنهادی را با سایر روش‌ها، در شدت تغییرات مختلف مورد مقایسه قرار داده است. افزایش شدت تغییرات باعث می‌شود طول گام حرکتی الگوریتم برای جست‌وجو افزایش یافته و در نتیجه کار الگوریتم برای جست‌وجو مشکل‌تر شود. نتایج حاصل‌شده از جدول (۷) نشان می‌دهد که با افزایش شدت تغییرات، باز هم الگوریتم پیشنهادی کارایی بهتری نسبت به سایر رقبای خود دارد. شکل (۱۲) حرکت ذرات به‌سمت بهینه را در فضای مسئله در ارزیابی‌های[1] مختلف (eval) نشان می‌دهد. همان گونه که از شکل (۱۲) مشخص است، تنوع حداکثری ایجادشده برای الگوریتم باعث شده تا ذرات به‌سرعت مکان قله‌های متحرک را ردیابی کرده و همگرا شوند. در محیط‌های پویا پوشش اکثریتی قله‌ها می‌تواند باعث شود کارایی برای الگوریتم بالا رفته و در حقیقت ذرات به‌سرعت مکان قلهٔ تغییریافته را پیدا کنند. یکی از معیارهای مناسب برای مقایسهٔ الگوریتم‌ها در محیط‌های پویا درصد پوشش قله‌ها توسط ذرات در فضای مسئله است. هرچه ذرات قله‌های بیشتری را پوشش دهند، به‌معنای افزایش تنوع در فضای مسئله بوده و در نهایت منجر به افزایش کارایی الگوریتم می‌شود.

جدول (۸) روش پیشنهادی را با تعداد قله‌های مختلف و همچنین در ابعاد مختلف از نظر درصد پوشش قله‌ها در فضای مسئله با سه روش CPSO [۲۵] و [۱۶] مقایسه کرده است. باید خاطرنشان کرد که با افزایش ابعاد و افزایش تعداد قله‌ها پیچیدگی فضا به‌شدت افزایش می‌یابد. نتایج جدول (۸) نشان از برتری روش پیشنهادی از نظر پوشش قله‌ها در فضای مسئله دارد. این پوشش بالای برای قله‌ها، به‌دلیل ایجاد تنوع حداکثری است که روش چندجمعیتی و حافظه در فضای مسئله به وجود آورده‌اند.

1. Evaluation

جدول (۴): معرفی الگوریتم‌های مورد مقایسه با روش پیشنهادی

روش‌ها	مرجع	تنظیم پارامترها
Mohamadpour et. al	[۱۶]	اندازهٔ جمعیت=۱۰۰، اندازهٔ حافظه=۱۰، ضریب لجستیک آشوب=۰/۴، احتمال تقاطع=۰/۲، احتمال جهش=۰/۶
mQSO	[۲۴]	از پیکربندی (۵+۵q)۱۰ استفاده شده است که در آن ۱۰ گروه ایجاد می‌شود و در آن هریک از گروه‌ها دارای ۵ ذرهٔ خنثی و دارای ۵ ذرهٔ کوانتوم هستند. همچنین برای این الگوریتم شعاع کوانتوم برابر ۰/۵ و شعاع دفع و شعاع همگرایی برابر ۳۱/۵ تعیین شده است.
FMSO	[۱۸]	بیشینهٔ تعداد گروه‌های فرزند برابر ۱۰، شعاع دفع مابین گروه‌های فرزند برابر ۲۵، تعداد ذرات در گروه والد و گروه‌های فرزند به‌ترتیب برابر ۱۰۰ و ۱۰ در نظر گرفته شده است.
Cellular-PSO	[۳]	در این روش، یک آتاماتای سلولی ۵ بعدی با ۱۰۵ سلول و همسایگی مور[1] با شعاع ۲ سلول در فضای جست‌وجو به کار رفته است. بیشینهٔ سرعت ذرات برابر شعاع همسایگی و بیشینهٔ تعداد ذرات برای هر سلول برابر ۱۰ و شعاع جست‌وجوی محلی برابر ۰/۵ تعیین شده است. همچنین جست‌وجوی محلی را همهٔ ذرات پس از مشاهدهٔ تغییر در محیط برای یک مرحله اجرا می‌کنند.
CPSO	[۲۵]	اندازهٔ جمعیت اصلی برابر ۱۰۰ و اندازهٔ بزرگ‌ترین زیرگروه ۳۰ در نظر گرفته شده است. در روش CPSO ضرایب مؤلفه‌های اجتماعی و شناختی c_1 و c_2 در سرعت ذره به‌ترتیب برابر ۲/۸ و ۱/۳ و وزن اینرسی w میانگین c_1 و c_2 فرض شده است (۲/۰۵). پارامتر هم‌پوشانی در این روش برابر ۰/۷ در نظر گرفته شده است.
memory/search	[۲۶]	اندازهٔ جمعیت=۱۰۰، اندازهٔ حافظه=۱۰، احتمال تقاطع=۰/۲، احتمال جهش=۰/۶
SEAm	[۲۷]	اندازهٔ جمعیت=۱۰۰، اندازهٔ حافظه=۱۰، احتمال تقاطع=۰/۲، احتمال جهش=۰/۶
RIm	[۲۷]	اندازهٔ جمعیت=۱۰۰، اندازهٔ حافظه=۱۰، احتمال تقاطع=۰/۲، احتمال جهش=۰/۶
FTMPSO	[۵]	در این روش اندازه کل جمعیت برابر ۱۰۰، اندازهٔ جمعیت ردیاب برابر ۵، شعاع دفع برابر ۳۱/۵، ضرایب مؤلفه‌های اجتماعی و شناختی c_1 و c_2 در سرعت ذره به‌ترتیب برابر ۲/۸ و ۱/۳ و وزن اینرسی w میانگین c_1 و c_2 فرض شده است (۲/۰۵).
CPSOR	[۲۵]	ضرایب مؤلفه‌های اجتماعی و شناختی c_1 و c_2 در سرعت ذره به‌ترتیب برابر ۲/۸ و ۱/۳ و وزن اینرسی w میانگین c_1 و c_2 فرض شده است (۲/۰۵). پارامتر هم‌پوشانی در این روش برابر ۰/۷ در نظر گرفته شده است.
CGAR	[۲۵]	اندازهٔ جمعیت=۱۰۰، احتمال تقاطع=۰/۲، احتمال جهش=۰/۶
mCPSO	[۲۴]	در اینجا برای mCPSO از پیکربندی (۵+۵q)۱۰ استفاده شده است. تعداد کل ذرات برابر ۱۰۰ و برای این الگوریتم شعاع کوانتوم برابر ۰/۵، شعاع دفع و شعاع همگرایی برابر ۳۱/۵ تعیین شده است.
AmQSO	[۲۸]	تعداد کل ذرات برابر ۱۰۰ و برای این الگوریتم، تعداد ذرات در هر زیرجمعیت برابر ۱۰، شعاع کوانتوم برابر ۰/۵، شعاع دفع و شعاع همگرایی برابر ۳۱/۵ تعیین شده است.

جدول (۵): مقایسهٔ خطای برون‌خطی و خطای استاندارد روش پیشنهادی با سایر روش‌ها با تعداد قله‌های مختلف و پارمترهای استاندارد جدول ۲ و به‌کارگیری تابع نگاشت آشوب لجستیک در تولید جمعیت و جست‌وجو

روش	تعداد قله									
	1	2	5	10	20	30	40	50	100	200
MMSSPCO	**0.025 (0.009)**	**0.033 (0.008)**	**0.050 (0.009)**	**0.079(0.008)**	**0.096(0.005)**	**0.15(0.050)**	**0.15(0.070)**	**0.28 (0.067)**	**0.36(0.044)**	**0.69(0.053)**
Mohamadpour	1.09(-)	1.11(-)	1.16(-)	**1.17(-)**	**1.19(-)**	2.09(-)	2.50(-)	2.65(-)	2.41(-)	2.34(-)
FMSO	2.36(0.05)	3.82(0.35)	1.90(0.08)	1.91(0.08)	2.56(0.10)	2.68(0.10)	2.65(0.08)	2.63(0.08)	2.52(0.06)	2.36(0.05)
mQSO	1.24(0.06)	0.14(0.11)	0.72(0.30)	1.05(0.24)	1.59(0.22)	1.58(0.17)	1.51(0.12)	1.54(0.12)	1.41(0.08)	1.24(0.06)
CPSO	2.62(0.10)	0.51(0.04)	1.01(0.09)	1.51(0.10)	2.00(0.15)	2.19(0.17)	2.28(0.12)	2.43(0.13)	2.68(0.12)	2.62(0.10)
AmQSO	3.40(0.06)	2.55(0.12)	1.68(0.11)	1.78(0.05)	2.61(0.07)	2.93(0.08)	3.14(0.08)	3.26(0.08)	3.41(0.07)	3.40(0.06)
Cellular PSO	4.46(0.7)	5.63(0.49)	3.42(0.21)	2.56(0.05)	3.96(0.08)	3.93(0.10)	4.15(0.10)	3.26(0.08)	4.41(0.07)	3.40(0.06)
CPSOR	0.0356(0.008)	0.0535(0.005)	0.549(0.049)	0.599(0.048)	0.796(0.05)	1.05(0.06)	-	0.986(0.05)	1.06(0.04)	0.949(0.04)
CGAR	2.02(0.05)	1.88(0.10)	2.56(0.1)	2.6(0.13)	3.66(0.14)	3.12(0.1)	-	3.26(0.11)	2.68(0.07)	2.39(0.07)
mCPSO	2.44(0.04)	4.93(0.17)	2.07(0.08)	2.08(0.07)	2.64(0.07)	2.63(0.08)	2.67(0.07)	2.65(0.06)	2.49(0.04)	2.44(0.04)

1. Moore

جدول (۶): مقایسهٔ خطای برون‌خطی و خطای استاندارد روش پیشنهادی با سایر روش‌ها در فرکانس‌های تغییرات ۵۰۰، ۱۰۰۰، ۲۵۰۰ و ۱۰۰۰۰ و با تعداد قله‌های مختلف و به‌کارگیری تابع نگاشت آشوب لجستیک در تولید جمعیت و جست‌وجو

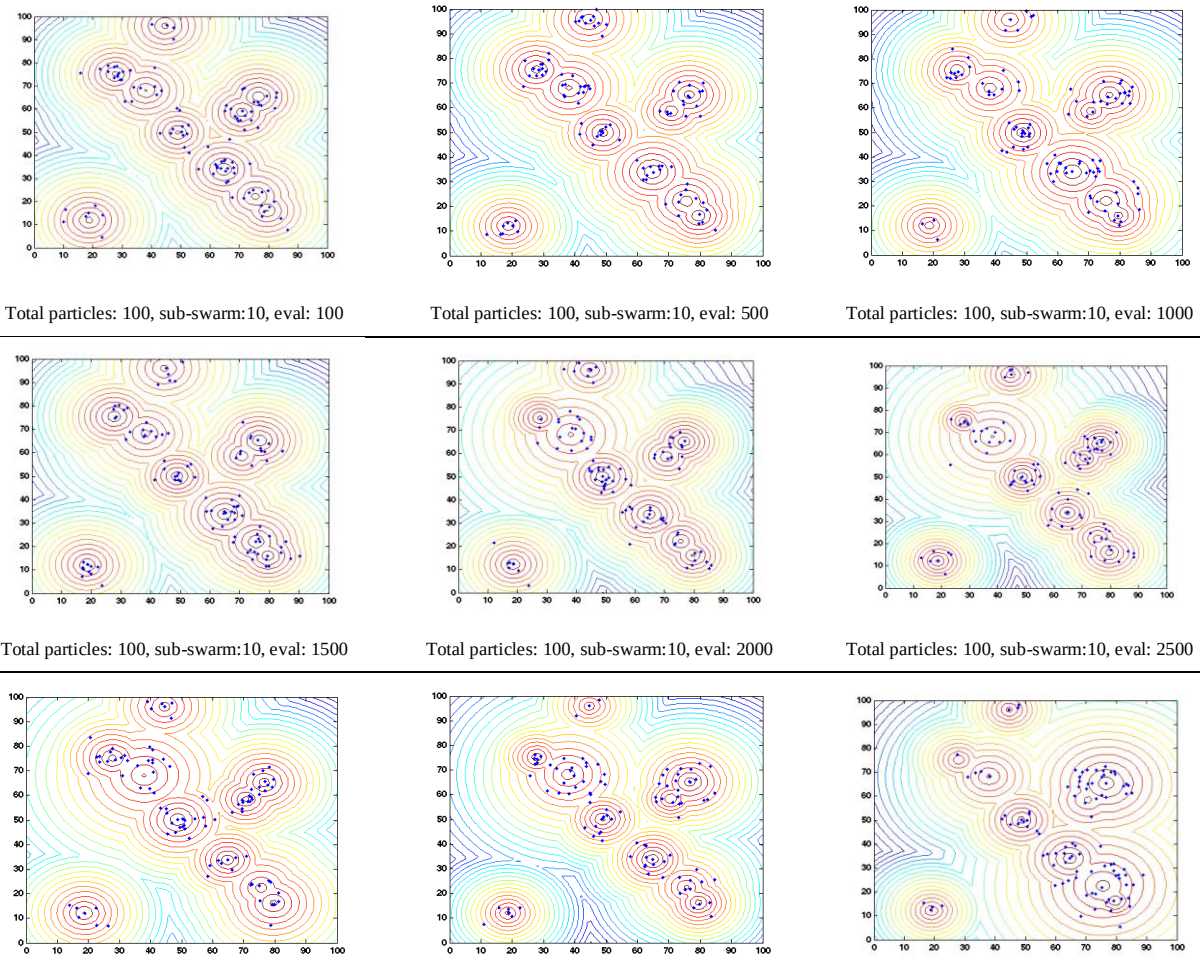
Frequency	روش	تعداد قله								
		1	5	10	20	30	40	50	100	200
500	**MMSSPCO**	**1.02(0.12)**	**1.11(0.10)**	**1.21(0.15)**	**1.32(0.19)**	**1.45(0.20)**	**1.58(0.21)**	**1.68(0.31)**	**1.74(0.35)**	**1.88(0.29)**
	FMSO	27.58(0.94)	19.45(0.45)	18.26(0.32)	17.34(0.30)	16.39(0.48)	15.34(0.45)	15.54(0.26)	12.87(0.60)	11.52(0.61)
	mQSO	33.67(3.42)	11.91(0.76)	9.62(0.34)	9.07(0.25)	8.80(0.21)	8.55(0.21)	8.72(0.20)	8.54(0.16)	8.19(0.17)
	CPSO	14.2521(-)	36.4094(-)	20.9129(-)	13.1155(-)	10.8307(-)	10.1239(-)	9.2871(-)	7.7726(-)	6.8333(-)
	AmQSO	3.02(0.32)	5.77(0.56)	5.37(0.42)	6.82(0.34)	7.10(0.39)	7.05(0.41)	8.97(0.32)	7.34(0.31)	7.48(0.19)
	CellularPSO	13.46(0.7)	9.63(0.49)	9.42(0.21)	8.84(0.28)	8.81(0.24)	8.94(0.24)	8.62(0.23)	8.54(0.21)	8.28(0.18)
1000	**MMSSPCO**	**0.45(0.10)**	**0.52(0.11)**	**0.56(0.20)**	**0.66(0.14)**	**0.70(0.16)**	**0.85(0.19)**	**0.96(0.14)**	**0.98(0.25)**	**1.05(0.20)**
	FMSO	14.42(0.48)	10.59(0.24)	10.40(0.17)	10.33(0.13)	10.06(0.14)	9.85(0.11)	9.54(0.11)	8.77(0.09)	8.06(0.07)
	mQSO	18.60(1.63)	6.56(0.38)	5.71(0.22)	5.85(0.15)	5.81(0.15)	5.70(0.14)	5.87(0.13)	5.83(0.13)	5.54(0.11)
	CPSO	8.935(-)	8.622(-)	7.481(-)	6.103(-)	5.448(-)	5.571(-)	5.171(-)	4.262(-)	3.742(-)
	AmQSO	2.33(0.31)	2.90(0.32)	4.56(0.40)	5.36(0.47)	5.20(0.38)	5.25(0.37)	6.06(0.14)	4.77(0.45)	5.75(0.26)
	CellularPSO	6.77(0.38)	5.30(0.32)	5.15(0.13)	5.23(0.18)	5.33(0.16)	5.61(0.16)	5.55(0.14)	5.57(0.12)	5.50(0.12)
2500	**MMSSPCO**	**0.09(0.01)**	**0.12(0.05)**	**0.14(0.08)**	**0.16(0.10)**	**0.18(0.09)**	**0.20(0.11)**	**0.28(0.13)**	**0.33(0.09)**	**0.31(0.09)**
	FMSO	-	-	-	-	-	-	-	-	-
	mQSO	7.64(0.64)	3.26(0.21)	3.12(0.14)	3.58(0.13)	3.63(0.10)	3.55(0.10)	3.63(0.10)	3.58(0.08)	3.30(0.06)
	CPSO	-	-	-	-	-	-	-	-	-
	AmQSO	0.87(0.11)	2.16(0.19)	2.49(0.10)	2.73(0.11)	3.24(0.18)	3.27(0.23)	3.68(0.15)	3.53(0.14)	3.07(0.12)
	CellularPSO	4.15(0.25)	2.85(0.24)	2.80(0.10)	3.41(0.14)	3.62(0.12)	3.84(0.12)	3.86(0.10)	4.10(0.11)	3.97(0.10)
10000	**MMSSPCO**	**0.001(0.003)**	**0.002(0.002)**	**0.007(0.003)**	**0.008(0.002)**	**0.009(0.001)**	**0.010(0.001)**	**0.012(0.007)**	**0.016(0.004)**	**0.019(0.003)**
	FMSO	1.90(0.06)	1.75(0.06)	1.91(0.04)	2.16(0.04)	2.18(0.04)	2.21(0.03)	2.60(0.08)	2.20(0.03)	2.00(0.02)
	mQSO	1.90(0.18)	1.03(0.06)	1.10(0.07)	1.84(0.08)	2.00(0.09)	1.99(0.07)	1.99(0.07)	1.85(0.05)	1.71(0.04)
	CPSO	-	-	-	-	-	-	-	-	-
	AmQSO	0.19(0.02)	0.45(0.04)	0.76(0.06)	1.28(0.12)	1.78(0.09)	1.68(0.06)	1.55(0.08)	1.89(0.14)	2.52(0.10)
	CellularPSO	1.53(0.12)	0.92(0.10)	1.19(0.07)	2.20(0.10)	2.60(0.13)	2.73(0.11)	2.84(0.12)	2.93(0.09)	2.88(0.07)

جدول (۷): مقایسهٔ خطای برون‌خطی و خطای استاندارد روش پیشنهادی با سایر روش‌ها با شدت تغییرات مختلف، در شرایط یکسان برای همهٔ روش‌ها و به‌کارگیری تابع نگاشت آشوب لجستیک در تولید جمعیت و جست‌وجو

روش	شدت تغییرات در قله			
	1	2	3	5
mQSO10 (5+5q)	1.85(0.08)	2.40(0.06)	3.00(0.06)	4.24(0.10)
AmQSO	1.51(0.10)	2.09(0.08)	2.72(0.09)	3.71(0.11)
MCPSO (Blackwell and Branke, 2006)	4.89(0.11)	3.57(0.08)	2.80(0.05)	2.08(0.07)
Mohamadpour et. al	1.19(-)	1.24(-)	1.35(-)	2.20(-)
FTMPSO	1.31(0.06)	1.20(0.06)	1.40(0.09)	1.69(0.07)
MMSSPCO	**0.079(0.008)**	**0.19(0.010)**	**0.67 (0.008)**	**1.07 (0.028)**

جدول (۸): مقایسهٔ روش پیشنهادی با سایر روش‌ها از نظر درصد پوشش قله‌ها و به‌کارگیری تابع نگاشت آشوب لجستیک در تولید جمعیت و جست‌وجو

تعداد قله	ابعاد	CPSO	[۱۶]	MMSSPCO
5	**2**	5	**5**	**5**
	5	3.11	**5**	**5**
	10	2.84	**4.92**	4.82
10	**2**	10	**10**	**10**
	5	7.15	9.74	**9.89**
	10	5.94	8.41	**8.60**
15	**2**	13.24	**15**	**15**
	5	9.21	14.06	**14.09**
	10	7.55	**12.65**	**12.45**
20	**2**	16.24	**20**	**20**
	5	11.25	18.17	**18.26**
	10	9.89	15.44	**15.57**



شکل (۱۲): نحوهٔ حرکت ذرات به‌سمت قله‌های متحرک

همان گونه که در بخش‌های گذشته نیز بیان شد، در روش پیشنهادی به‌جای اینکه ذرات جمعیت به‌صورت تصادفی تولید شوند، از نگاشت آشوب برای تولید ذرات استفاده شده است. توابع مختلف مربوط به نگاشت آشوب در بخش ۲ تشریح شده است. برای اینکه بتوان نتایج تولیدشده توسط هریک از توابع آشوب بر روی روش پیشنهادی را مشاهده کرد، در این بخش این توابع مورد آزمایش قرار گرفته‌اند. جدول (۹) مقدار خطای برون‌خطی حاصل از روش پیشنهادی با پارامترهای استاندارد جدول (۲) با تعداد قله‌های مختلف و نگاشت‌های مختلف آشوب را نشان می‌دهد. در جدول (۹) از روابط آشوب برای ایجاد جمعیت اولیهٔ ذرات استفاده شده و نتایج مطلوبی تولید شده است. نتایج حاصل‌شده نشان می‌دهد که تابع نگاشت لجستیک تا حدودی نتایج بهتری را نسبت به سایر توابع تولید کرده است. جدول (۱۰) نتایج مختلفی را نشان می‌دهد که الگوریتم پیشنهادی برای تولید ذرات و جست‌وجوی ذرات به‌صورت آشوب‌گونه عمل کرده است. نتایج حاصل از این جدول در مقایسه با جدول (۹) نشان داده که اگر هم در جست‌وجو و هم در تولید ذرات از آشوب استفاده شود، نتایج مطلوب‌تری به همراه خواهد داشت. در جدول (۱۰) همان گونه که مشاهده می‌شود، نتایجی که از ترکیب نگاشت لجستیک و تکراری به دست آمده، تا حدودی نسبت به دیگر ترکیب‌ها مناسب‌تر است. در روش پیشنهادی برای بهبود کارایی الگوریتم از حافظهٔ تخمین تراکم گوسی به‌جای حافظهٔ استاندارد استفاده شده است. همان گونه که قبلاً

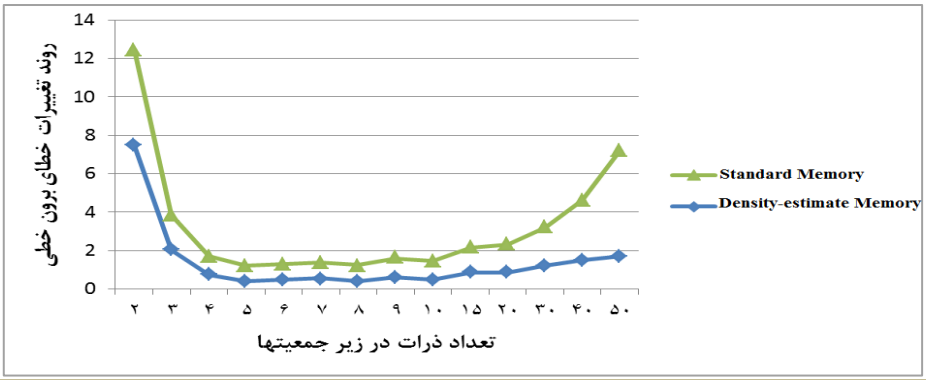
نیز گفته شد، حافظهٔ تخمین تراکم گوسی می‌تواند نقاط ضعف حافظهٔ استاندارد را از بین برده و کارایی را برای الگوریتم افزایش دهد. شکل (۱۳) روش پیشنهادی را با دو حالتی که از حافظهٔ استاندارد و حافظهٔ تخمین تراکم استفاده شده، مورد مقایسه قرار داده است. در شکل (۱۳) محور افقی بر اساس تعداد ذرات در هر زیرجمعیت در نظر گرفته شده تا تأثیر این پارامتر مهم نیز بر روی روش پیشنهادی بیشتر نمایان شود. نتایج این شکل نشان داده که حافظهٔ تخمین تراکم کارایی الگوریتم را به‌صورت چشمگیری افزایش داده است. با افزایش تعداد زیرجمعیت‌ها نیز کارایی برای الگوریتم کاهش یافته است. بنابراین به‌دست‌آوردن مقدار مناسب برای این پارامتر بسیار حائز اهمیت است (طبق آزمایش‌ها مقدار ۵ زیرجمعیت مناسب است). تعداد زیرجمعیت‌های فعال تأثیر فراوانی بر کارایی الگوریتم دارد. بنابراین شکل (۱۴)، روند تغییرات خطای برون‌خطی را بر اساس تعداد زیرجمعیت‌های فعال در محیط، در فرکانس تغییرات ۵۰۰۰ و به‌ازای تعداد قله‌های مختلف (p) نشان می‌دهد.

جدول (۹): مقدار خطای برون‌خطی حاصل از روش پیشنهادی با پارمترهای استاندارد جدول ۲، با تعداد قله‌های مختلف و نگاشت‌های مختلف آشوب

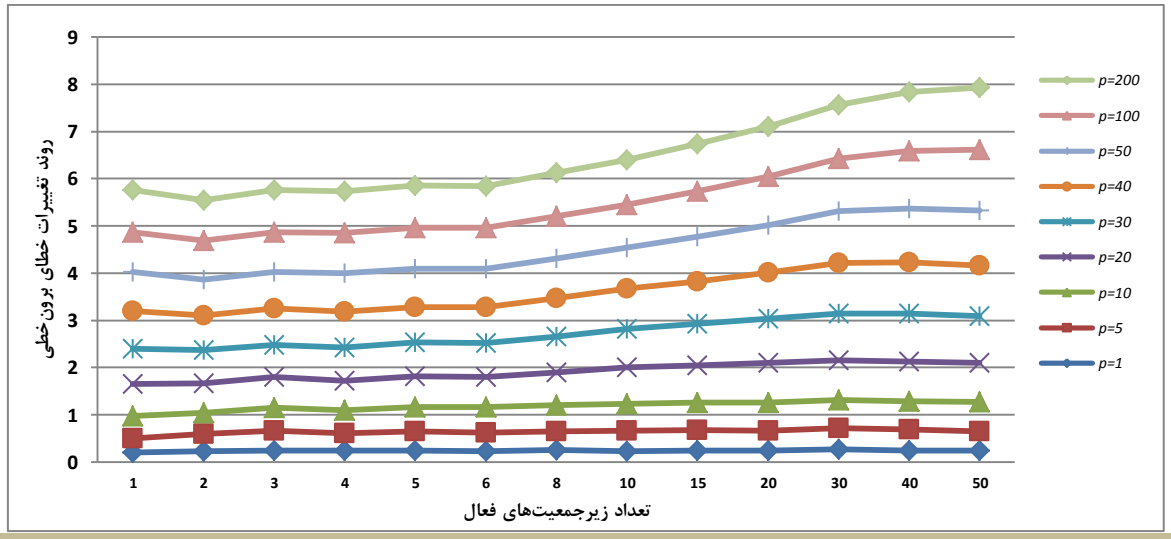
تعداد قله	نگاشت				
	دایره	تکراری	لیبویتچ	لجیستیک	چبیشف
1	0.060(0.000)	0.065(0.000)	0.071(0.010)	**0.054(0.001)**	0.060(0.001)
5	0.081(0.010)	**0.076(0.002)**	0.086(0.030)	0.078(0.001)	0.083(0.004)
10	0.092(0.010)	**0.088(0.007)**	0.096(0.010)	0.098(0.008)	0.099(0.009)
20	0.101(0.010)	0.097(0.009)	0.099(0.007)	**0.095(0.008)**	0.119(0.011)
30	0.193(0.014)	0.189(0.017)	0.197(0.012)	**0.182(0.011)**	0.195(0.015)
40	0.250(0.025)	0.198(0.017)	0.230(0.015)	**0.190(0.015)**	0.240(0.020)
50	0.300(0.045)	0.291(0.028)	0.298(0.025)	**0.230(0.021)**	0.310(0.043)
100	0.541(0.066)	0.510(0.048)	0.601(0.023)	**0.441(0.042)**	0.642(0.067)
200	0.901(0.080)	0.781(0.093)	0.804(0.081)	**0.741(0.063)**	0.902(0.082)

جدول (۱۰): نتایج حاصل از پارمترهای استاندارد تابع محک قله‌های متحرک با استفاده از جفت‌نگاشت‌های آشوب در تولید ذرات و جست‌وجوی فضای مسئله

تعداد قله	جفت نگاشت‌های مورد استفاده شده در قسمت $\frac{\text{تولید ذرات}}{\text{جست‌وجو}}$					
	دایره	تکراری	لجستیک	تکراری	چبیشف	چبیشف
	تکراری	لیبویتچ	تکراری	لیبویتچ	چبیشف	لیبویتچ
1	0.041(0.000)	0.046(0.000)	**0.022(0.000)**	0.036(0.000)	0.042(0.000)	0.048(0.001)
5	0.061(0.000)	0.056(0.000)	**0.046(0.000)**	0.058(0.000)	0.063(0.001)	0.0611(0.002)
10	0.082(0.010)	0.067(0.002)	**0.056(0.000)**	0.068(0.001)	0.089(0.005)	0.092(0.001)
20	0.091(0.010)	0.081(0.003)	**0.059(0.001)**	0.075(0.003)	0.091(0.001)	0.101(0.001)
30	0.093(0.014)	0.091(0.007)	**0.067(0.002)**	0.080(0.001)	0.095(0.005)	0.123(0.004)
40	0.150(0.015)	0.098(0.006)	**0.070(0.005)**	0.085(0.005)	0.099(0.009)	0.140(0.005)
50	0.200(0.035)	0.099(0.008)	**0.078(0.005)**	0.090(0.005)	0.101(0.003)	0.1650(0.005)
100	0.441(0.066)	0.112(0.008)	**0.080(0.003)**	0.099(0.002)	0.122(0.007)	0.181(0.006)
200	0.801(0.080)	0.280(0.009)	**0.091(0.001)**	0.101(0.003)	0.152(0.008)	0.198 (0.008)



شکل (۱۳): روند تغییر در خطای برون‌خطی برای روش پیشنهادی به‌ازای تعداد ذرات در هر زیرجمعیت

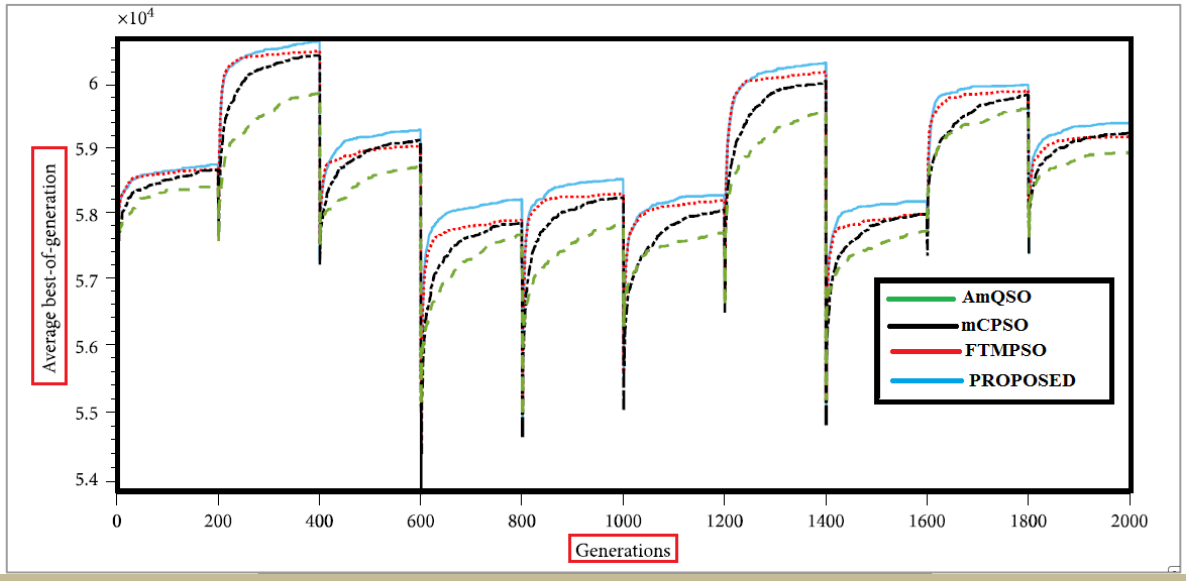


شکل (۱۴): روند تغییرات خطای برون‌خطی بر اساس تعداد زیرجمعیت‌های فعال در محیط در فرکانس ۵۰۰۰ و به‌ازای تعداد قله‌های مختلف

جدول (۱۱): نتایج آزمون *t*

t-test results	Moving Peaks Bnechmark														
فرکانس تغییرات	500	1000	1500	5000	10000	500	1000	1500	5000	10000	500	1000	1500	5000	10000
mQSO, AmQSO	+	-	-	-	~	+	-	-	+	-	~	+	-	-	~
mQSO, mCPSO	-	-	-	-	+	-	-	+	-	~	-	-	-	~	-
mQSO, FTMPSO	-	-	~	+	-	-	-	-	~	-	+	-	-	+	~
mQSO, CellularPSO	+	-	-	+	~	+	-	-	+	-	~	+	-	+	~
mQSO, FTMPSO	+	-	-	+	~	+	-	+	+	-	~	+	-	-	~
mQSO, [16]	-	-	-	-	-	-	-	+	-	~	-	-	-	~	-
AmQSO, mCPSO	+	-	-	-	~	+	-	-	+	-	~	+	-	-	~
AmQSO, FTMPSO	-	-	+	-	+	-	-	+	-	~	-	-	+	~	-
AmQSO, CellularPSO	-	-	~	+	-	-	-	-	~	-	+	-	-	+	~
AmQSO, FTMPSO	+	-	-	+	+	+	-	+	-	-	-	~	+	+	~
AmQSO, [16]	~	-	~	-	-	-	+	-	-	-	-	+	-	-	-
FTMPSO, CellularPSO	+	-	-	+	-	+	-	-	-	-	-	~	+	+	~
FTMPSO, [16]	-	-	-	-	~	+	-	-	+	-	~	+	-	-	~
MMSSPCO, AmQSO	-	-	-	-	+	-	-	+	-	~	-	-	-	~	-
MMSSPCO, mCPSO	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
MMSSPCO, TMPSO	+	+	+	+	+	+	+	+	+	+	+	+	+	~	+
MMSSPCO,CellularPSO	+	+	+	+	+	-	+	+	~	+	-	+	+	+	+
MMSSPCO, FTMPSO	+	-	+	+	+	+	-	+	+	+	+	+	+	+	~
MMSSPCO, [16]	+	+	+	+	-	+	+	+	+	+	+	-	+	+	~

جدول (۱۱) هر شش روش را از نظر آماری با هم مقایسه کرده است. برای مقایسهٔ آماری الگوریتم‌ها از آزمون *t* استفاده شده است. در این آزمون، ما از ۹۸ درجه آزادی با سطح معنی‌دار برابر با ۰/۵ استفاده کرده‌ایم. نوع آزمون را paired و one-tailed قرار داده‌ایم. بعد از انجام آزمایش، نتایج بدین صورت نمایش داده می‌شوند که اگر الگوریتم اول از الگوریتم دوم به‌صورت معناداری بهتر باشد، از علامت + استفاده شده است. اگر الگوریتم اول از الگوریتم دوم به‌صورت معناداری بدتر باشد، از علامت – استفاده شده است. اگر شواهد لازم برای معناداری هیچ کدام از الگوریتم‌ها پیدا نشود، از علامت ~ استفاده شده است. شایان ذکر است که برای آزمون آماری معناداری، روش پیشنهادی ۳۰ بار اجرا شده است. نمودارهای همگرایی چهار الگوریتم با شرایط یکسان برانک و در شکل (۳) ارائه شده‌اند. برای هر تغییر، روش پیشنهادی می‌تواند به بهترین نتیجه برسد. از نمودار مشخص است که الگوریتم پیشنهادی توانایی کارآمدتری در تنظیم محیط‌های پویا دارد.



شکل (۱۵): نمودار همگرایی روش پیشنهادی در مقایسه با سایر روش‌های رقیب

جدول (۱۲): مقایسهٔ خطای برون‌خطی و خطای استاندارد روش پیشنهادی با روش SSPCO با پارامترهای استاندارد جدول (۲)

روش	تعداد قله									
	1	2	5	10	20	30	40	50	100	200
MMSSPCO	**0.025 (0.009)**	**0.033 (0.008)**	**0.050 (0.009)**	**0.079 (0.008)**	**0.096 (0.005)**	**0.15 (0.050)**	**0.15 (0.070)**	**0.28 (0.067)**	**0.36 (0.044)**	**0.69 (0.053)**
SSPCO	3.00 (0.02)	3.90 (0.06)	3.75 (0.06)	2.91 (0.04)	2.88 (0.04)	2.85 (0.04)	2.80 (0.03)	3.60 (0.08)	4.20 (0.03)	4.00 (0.02)

جدول (۱۲) روش پیشنهادی را از نظر خطای برون‌خطی در شرایط استاندارد برانک [۱۳] و با تعداد قله‌های مختلف، با روش سادهٔ SSPCO مقایسه می‌کند. نتایج جدول (۱۲) حاکی از این است که روش سادهٔ SSPCO در مقابل روش پیشنهادی دارای خطای بالایی در محیط‌های پویاست.

مؤلفه‌های اصلی روش پیشنهادی شامل خوشه‌بندی و به‌روزرسانی حافظه می‌باشند. عملیات خوشه‌بندی برای روش پیشنهادی فقط یک بار در هر بار تغییر انجام می‌گیرد. این عملیات بعد از تغییر در محیط انجام می‌شود، پس تعداد کل اجراهای این الگوریتم، $\frac{ite}{f}$ بار می‌باشد که ite نشان‌دهندهٔ تعداد محاسبهٔ کارایی و f نشان‌دهندهٔ فرکانس تغییرات است. با توجه به این موضوع، پیچیدگی محاسباتی روش پیشنهادی از مرتبهٔ $O\left(\frac{ite}{f}\times(M\times CI\times k+M\times\log M)+ite\times FT\right)$ می‌باشد، که M، CI، k و FT به‌ترتیب اندازهٔ جمعیت، تعداد چرخه‌های الگوریتم خوشه‌بندی، تعداد خوشه‌ها در الگوریتم خوشه‌بندی و زمان اجرای تابع کارایی است. با توجه به آنکه اصل زمان مصرفی در مرتب‌سازی کارایی افراد صرف می‌شود، مرتبهٔ الگوریتم را می‌توانیم برابر با $O(M\times\log M)$ در نظر بگیریم. جدول (۱۳) روش پیشنهادی را از نظر پیچیدگی محاسباتی با دو روش CPSO، CPSOR و mQSO مقایسه می‌کند.

جدول (۹): مقایسهٔ روش پیشنهادی با دیگر روش‌ها از نظر پیچیدگی محاسباتی

الگوریتم	پیچیدگی محاسباتی
MMSSPCO	$\boldsymbol{O(M\log M)}$
FTMPSO	$O(M^3)$
mCPSO	$O(M^3)$
mQSO	$O(M^2)$

۵. نتیجه‌گیری و پیشنهادات آتی

الگوریتم‌های بهینه‌سازی هوشمند در محیط‌های پویا باید به‌گونه‌ای طراحی شوند که بتوانند به‌طور مناسب و کارا بهینهٔ مورد نظر را دنبال کنند. در این مقاله از ترکیب حافظهٔ تخمین تراکم گوسی و چندجمعیتی با الگوریتم SSPCO برای حل

مسئلهٔ قله‌های متحرک استفاده شده است. یک راه معمول برای حفظ اطلاعات گذشته، استفاده از حافظه است که راه‌حل‌ها را به‌صورت دوره‌ای ذخیره می‌کند و زمانی که محیط تغییر می‌کند می‌توان آن‌ها را بازیابی کرد. حافظه می‌تواند به جست‌وجوی سریع کمک کند و در تغییرات مسائل پویا کارآمد باشد. یکی از مشکلاتی که برای حافظهٔ استاندارد رخ می‌دهد، ظرفیت محدود این حافظه برای ذخیره‌سازی است. در این مقاله با استفاده از حافظهٔ تخمین تراکم گوسی نقاط ضعف حافظهٔ استاندارد برطرف شده و بررسی انجام‌گرفته در بخش آزمایش‌ها نشان داده است که روش پیشنهادی با حافظهٔ تخمین تراکم بهتر از حافظهٔ استاندارد عمل می‌کند. همچنین در این الگوریتم برای ایجاد جمعیت اولیه به‌جای تصادفی‌سازی جمعیت اولیه از توابع نگاشت آشوب استفاده کرده است. در این روش به‌جای جست‌وجوی تصادفی از جست‌وجوی آشوب‌گونه استفاده شده زیرا در رفتار حرکتی آشوب‌گونه نسبت به رفتار حرکتی تصادفی می‌توان پیش‌بینی مناسب‌تری از آینده داشت. در این روش در بخش مربوط به نتایج تجربی، نشان داده شده است که در حرکت آشوب‌گونه، میزان همگرایی جمعیت به بهینهٔ مورد نظر بیشتر بوده و در نتیجه میزان کارایی الگوریتم افزایش یافته است. در این روش نیز از روش چندجمعیتی به‌صورت تطبیقی استفاده شده و همان گونه که از نتایج برمی‌آید، در افزایش کارایی الگوریتم بسیار مفید واقع شده است. در بخش نتایج تجربی و آزمایش‌ها، روش پیشنهادی با برخی دیگر از روش‌های رقیب در فرکانس‌های تغییرات مختلف مقایسه شد. نتایج حاصل‌شده از جداول، حاکی از برتری روش پیشنهادی نسبت به سایر روش‌ها بوده است. همچنین در این بخش، روش پیشنهادی با سایر روش‌ها، در شدت تغییرات مختلف مورد مقایسه قرار گرفته است. افزایش شدت تغییرات باعث می‌شود طول گام حرکتی الگوریتم برای جست‌وجو افزایش یافته و در نتیجه کار الگوریتم برای جست‌وجو مشکل‌تر شود. نتایج حاصل‌شده نشان می‌دهند که با افزایش شدت تغییرات، باز هم الگوریتم پیشنهادی کارایی بهتری نسبت به سایر رقبای خود دارد. حرکت ذرات به‌سمت بهینه بر روی نمودارهای مختلف در فضای مسئله در ارزیابی‌های مختلف نشان داده شد. تنوع حداکثری ایجادشده برای الگوریتم باعث شده تا ذرات به‌سرعت مکان قله‌های متحرک را ردیابی کرده و همگرا شوند. جدول (۸) روش پیشنهادی را با تعداد قله‌های مختلف و همچنین در ابعاد مختلف از نظر درصد پوشش قله‌ها در فضای مسئله با سه روش CPSO [۲۵] و [۱۶] مقایسه کرده است. باید خاطرنشان کرد که با افزایش ابعاد و افزایش تعداد قله‌ها پیچیدگی فضا به‌شدت افزایش می‌یابد. روش پیشنهادی از نظر پوشش تعداد قله‌های مختلف نیز مورد ارزیابی قرار گرفت. نتایج نشان از برتری روش پیشنهادی از نظر پوشش قله‌ها در فضای مسئله دارد. این پوشش بالای برای قله‌ها، به‌دلیل ایجاد تنوع حداکثری است که روش چندجمعیتی و حافظه در فضای مسئله به وجود آورده‌اند. روش پیشنهادی، با استفاده از توابع مختلف نگاشت آشوب مورد آزمایش قرار گرفته و نتایج نشان داده‌اند که به‌کارگیری آشوب در تولید جمعیت اولیهٔ ذرات و همچنین در جست‌وجوی ذرات باعث می‌شود کارایی الگوریتم به‌طور چشمگیری افزایش یابد. از میان روابط آشوب استفاده‌شده، تابع نگاشت لجستیک توانسته کارایی الگوریتم را نسبت به سایر توابع بیشتر افزایش دهد. در حالتی که توابع به‌صورت جفتی برای تولید ذرات و جست‌وجو استفاده شده‌اند، ترکیب دو تابع نگاشت لجستیک و تکراری توانسته کارایی الگوریتم را نسبت به سایر جفت توابع دیگر، بیشتر افزایش دهد. نتایج حاصل‌شده از روش پیشنهادی نشان می‌دهند که این روش برای حل مسائل بهینه‌سازی پویا بسیار مناسب و کارا خواهد بود. روش پیشنهادی می‌تواند برای اکثر مسائل پویا از جمله زمان‌بندی فعالیت‌ها، بهینه‌سازی مسیریابی در شبکه‌های حسگر متحرک، داده‌کاوی در شرایطی که پایگاه داده به‌طور مداوم باید به‌روزرسانی شود و... مورد استفاده قرار گیرد. در حل مسائل بهینه‌سازی پویا با چالش‌های اساسی روبه‌رو هستیم که یک الگوریتم بهینه‌سازی باید به‌گونه‌ای طراحی شود که بتواند به‌نحو مطلوبی این چالش‌های موجود را حل کند. یکی از چالش‌های که ممکن است در یک محیط با ماهیت پویا رخ دهد، وجود نویز در محیط است. نویز می‌تواند مقدار کارایی را برای یک

الگوریتم به‌شدت تحت‌تأثیر قرار دهد. بنابراین یکی از پیشنهادات آتی می‌تواند بررسی این الگوریتم در شرایطِ محیطیِ پویا و نویزی باشد. از دیگر چالش‌های دیگری که یک محیط با ماهیت با آن مواجه است، گیر افتادن در بهینهٔ محلی است [۳۶، ۳۷ و ۳۸]. افزایش حداکثری تنوع منجر به کاهش این اتفاق می‌شود. روش پیشنهادی صرفاً برای مسئلهٔ معروف محک قله‌های متحرک مورد ارزیابی قرار گرفته و نویسنده صرفاً دربارهٔ این مسئله صحبت به میان آورده است. بنابراین برای سایر مسائل معروف دیگر ارزیابی صورت نگرفته است. اکثریت محققان که در زمینهٔ بهینه سازی تکاملی پویا تحقیق و پژوهش انجام می‌دهند، یا بر روی این مسئله محک الگوریتم خود را ارزیابی کرده‌اند و یا مسائلی از جمله مسائل محک معروف [۲۴] و برخی نیز به‌صورت کاربردی بر روی مسئلهٔ زمان‌بندی پویا. بنابراین این موضوع می‌تواند در بخش کارهای آتی واگذار مورد بررسی دیگر محققان قرار گیرد.

مراجع

[1] Cruz C., Gonzalez J.R., and Pelta D.A., "*Optimization in Dynamic Environments: A Survey on Problems, Methods and Measures*", Journal Soft Computing-A Fusion of Foundations, Methodologies and Applications, vol. 15, pp. 1427-1448, 2011.

[2] Yang S., "*Genetic Algorithms With Elitism-Based Immigrants For Changing Optimization Problems*", Applications of Evolutionary Computing, vol. 4448, pp. 627-636, 2007.

[3] Hashemi A. B. and Meybodi M. R., "*Cellular PSO: A PSO for Dynamic Environments*", Advances in Computation and Intelligence, pp. 422–433, 2009.

[4] Yang S., "*Explicit Memory Schemes For Evolutionary Algorithms In Dynamic Environments*", Evolutionary Computation in Dynamic and Uncertain Environments, pp. 3-28, 2007.

[5] Yazdani D. and Nasiri B., Sepas-Moghaddam A., Meybodi M.R., "*A Novel Multi-Swarm Algorithm For Optimization In Dynamic Environments Based On Particle Swarm Optimization*", Applied Soft Computing, vol. 13, pp. 2144- 2158, 2013.

[6] Omidvar R., Parvin H. and Rad F., "*SSPCO Optimization Algorithm (See-See Partridge Chicks Optimization)*", 14th-Mexican international conferences on artificial intelligence, IEEE, 2015.

[7] Arena P., Caponetto R., Fortuna L., Rizzo A. and La Rosa M., "*Self-organization in nonrecurrent complex systems*", International Journal of Bifurcation and Chaos, vol. 10, no. 5, pp. 1115-1125, 2000.

[8] Alatas B., Akin E. and Ozer A. B., "*Chaos embedded particle swarm optimization algorithms*", Chaos, Solitons & Fractals, vol. 4, no. 40, 1715-1734, 2009.

[9] Lorenz E. N., "*Deterministic nonperiodic flow*", Journal of the atmospheric sciences, vol. 2, no. 20, pp. 130-141, 1963.

[10] Gandomi A. H., and Yang X.-S., "*Chaotic bat algorithm*", Journal of Computational Science, vol. 2, no. 5, pp. 224-232, 2014.

[11] Brank J., "*Evolutionary Optimization in Dynamic Environments*", vol. 3 of Genetic algorithm and evolutionary computation, Kluwer Academic Publisher, Massachusettes, USA, 2001.

[12] Brank J., "*Memory enhanced evolutionary algorithms for changing optimization problems*", in Proceeding of the 1999 IEEE Congress on Evolutionary Computation (CEC-1999), vol. 3, pp. 1875–1882, 1999.

[13] Brank J., "*Evolutionary Optimization in Dynamic Environments*", Kluwer, 2002.

[14] Ngyen T. T., "*Continuous Dynamic Optimisation Using Evolutionary Algorithm*", Doctor of Philosophy Thesis, University of Birmingham, 2010.

[۱۵] محمدپور، پروین، «*الگوریتم ژنتیک آشوب‌گونه مبتنی بر حافظه و خوشه‌بندی برای حل مسائل بهینه‌سازی پویا*»، مجله مهندسی برق دانشگاه تبریز، جلد ۴۶، شمارهٔ ۳، ص ۲۹۹-۳۱۸، ۱۳۹۵.

[16] Mohammadpour M. and Parvin H., "*Genetic Algorithm Based on Explicit Memory for Solving Dynamic Problems*", Journal of Advances in Computer Research Sari Branch Islamic Azad University, vol. 7, no. 2, pp. 53-68, 2016.

[17] Yang S., "*Genetic Algorithms With Elitism-Based Immigrants For Changing Optimization Problems*", Applications of Evolutionary Computing, vol. 4448, pp. 627-636, 2007.

[18] Li C. and Yang S., "*Fast Multi-Swarm Optimization For Dynamic Optimization Problems*", Natural Computation, ICNC '08. Fourth International Conference, vol. 7, pp. 624-628, 2008.

[19] Li C. and Yang S., "*An Island Based Hybrid Evolutionary Algorithm For Optimization*", Simulated Evolution and Learning, pp. 180-189, 2008.

[20] Rezazadeh I., Meybodi M. R. and Naebi A., "*Adaptive Particle Swarm Optimization Algorithm For Dynamic Environments*", ICSI'11 Proceedings of the Second international conference on Advances in swarm intelligence, vol. I, pp. 120-129, 2011.

[21] Yazdani D., Akbarzadeh-Totonchi M. R., Nasiri B., and Meybodi M. R., "*A New Artificial Fish Swarm Algorithm For Dynamic Optimization Problems*", Evolutionary Computation (CEC), IEEE Congress on, pp. 1-8, 2012.

[22] Mohammadpour M., Parvin H. and Sina M., "*Chaotic Genetic Algorithm based on Explicit Memory with a new Strategy for Updating and Retrieval of Memory in Dynamic Environments*", Journal of AI and Data Mining, vol. 6, no. 1, pp. 191-205, 2018.

[23] Rastegar R. and Meibodi M. R., "*Study of global*

[۳۸] غلامشاهی، شبنم، هاشمی‌نژاد، سید محمدحسین، «*روشی برای تشخیص مؤلفه‌های نرم‌افزاری مبتنی بر الگوریتم ژنتیک مرتب‌سازی نامغلوب*» مجله محاسبات نرم، جلد ۷، شمارهٔ ۲، ص ۴۷ـ۶۴، ۱۳۹۷.

convergence time complexity of Estimation of Distribution Algorithm", Computer Engineering Department, Amirkabir University, Tehran, Iran, Sep 2007.

[24] Blackwell T. M. and Branke J., "*Multi-swarms, exclusion and anti-convergence in dynamic environments*", IEEE Transactions on Evolutionary Computation, vol. 10, pp. 459–472, 2006.

[25] Yang S. and Li C., "*A General Framework of Multipopulation Methods with Clustering in Undetectable Dynamic Environments*", IEEE Transactions on Evolutionary Computation, vol. 16, no. 4, pp. 556-577, 2012.

[26] Branke J., Kaußler T., Schmidt C. and Schmeck H., "*A multipopulation approach to dynamic optimization problems*", In Adaptive Computing in Design and Manufacturing, pp. 299–308, 2000.

[27] Grefenstette J., "*Genetic algorithms for changing environments*", In Parallel Problem Solving from Nature, pp. 137–144, 1992.

[28] Blackwell T.M. and Branke J., "*Particle Swarms for Dynamic Optimization Problems*", Swarm Intelligence, pp.193–217, 2008.

[29] Mavrovouniotis M., Li Ch. and Yang S., "*A survey of swarm intelligence for dynamic optimization: Algorithm and application*", Swarm and Evolutionary Computation, vol. 33, pp. 1-17, 2017.

[30] Mirjalili S., and Lewis A., "*Grey Wolf Optimizer*", Advances in Engineering Software, pp. 69: 46-61, 2014.

[31] Yazdani D., Nasiri B., Sepas-Moghaddam A., and Meybodi M. R., "*A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization*", Applied Soft Computing, vol. 13, no. 4, pp. 2144-2158, 2013.

[32] Lung R., and Dumitrescu D., "*A Collaborative Model for Tracking Optima in Dynamic Environments*," in IEEE Congress on Evolutionary Computation, 2007.

[33] Ozsoydan F., and Baykasoglu A., "*A multi-population firefly algorithm for dynamic optimization problems*", in Evolving and Adaptive Intelligent Systems (EAIS), 2015 IEEE International Conference, 2015.

[34] Parvin H., Nejatian S., and Mohammadpour M., "*Explicit memory based ABC with a clustering strategy for updating and retrieval of memory in dynamic environments*", Applied Intelligence, vol 48, pp. 4317-4337, 2018.

[35] Bakas N. P., "*Numerical solution for the extrapolation problem of analytic functions*", Research, vol. 6, pp. 1-10, 2019.

[۳۶] سلیمی سرتختی، جواد، گلی بیدگلی، سلمان، «*ارائهٔ یک الگوریتم ترکیبی با استفاده از الگوریتم کرم شب‌تاب، الگوریتم ژنتیک و جست‌وجوی محلی*»، مجلهٔ محاسبات نرم، جلد ۸، شمارهٔ ۱، ص ۱۴ـ۲۷، ۱۳۹۸.

[۳۷] طاهر، سید عباس، فاطمی، سعید، هنرفر، امید، «*تجدید آرایش بهینهٔ شبکه‌های توزیع به‌منظور کاهش تلفات و افزایش قابلیت اطمینان با استفاده از الگوریتم خفاش*»، مجله محاسبات نرم، جلد ۸، شماره ۱، ص ۲۹ـ۴۲، ۱۳۹۸.