الگوریتم کلونی زنبور مصنوعی آشوب‌گونه مبتنی بر حافظه برای حل مسائل بهینه‌سازی پویا

مجید محمدپور[۱،*]، حمید پروین[۲]

اطلاعات مقاله

دریافت مقاله: ۱۳۹۴/۰۳/۳۱
پذیرش مقاله: ۱۳۹۵/۰۱/۱۸

واژگان کلیدی:
بهینه‌سازی،
الگوریتم کلونی زنبور مصنوعی،
محیط‌های پویا،
آشوب،
حافظه،
محک قله‌های متحرک.

چکیده

الگوریتم کلونی زنبور مصنوعی، یکی از الگوریتم‌های بهینه‌سازی هوش جمعی است که از آن در اهداف و کاربردهای ایستا به‌صورت وسیعی استفاده می‌شود. بیشتر مسائل موجود در جهان واقعی با ماهیت پویا هستند، بنابراین به الگوریتم‌های بهینه‌سازی نیاز است که بتوانند مسائل را در محیط‌های پویا به‌خوبی حل کنند. مسائل بهینه‌سازی پویا مسائلی هستند که در طول زمان دچار تغییر می‌شوند. در این مقاله یک الگوریتم مبتنی بر کلونی زنبور مصنوعی آشوب‌گونه مبتنی بر حافظه برای مسائل بهینه‌سازی پویا ارائه شده است. یک سیستم آشوب‌گونه پیش‌بینی دقیق‌تری از آینده نسبت به یک سیستم تصادفی دارد. در این روش از حافظه صریح برای ذخیره راه‌حل‌های قدیمی خوب جهت نگهداری تنوع در جمعیت استفاده شده است. استفاده از راه‌حل‌های قدیمی خوب و تنوع در محیط به سرعت هم‌گرایی الگوریتم کمک می‌کند. روش پیشنهادی بر روی مسئله محک قله‌های متحرک آزمایش شده است. مسئله محک قله‌های متحرک، شبیه‌ساز مناسبی برای آزمایش کارایی الگوریتم‌های بهینه‌سازی در محیط‌های پویا است. نتایج آزمایش‌ها بر روی این تابع محک، نشان از کارایی مناسب روش پیشنهادی در مقایسه با سایر روش‌ها در حل مسائل بهینه‌سازی پویا دارد.

۱- مقدمه

امروزه هدف بسیاری از مسائل مهندسی، کم‌کردن هزینه (کمینه‌سازی) و در بعضی دیگر، حداکثرکردن سود (بیشینه‌سازی) است. کمینه‌سازی یا بیشنه‌سازی در این مسائل، نیازمند روش‌های بهینه‌سازی است. به‌طور کلی مسائل بهینه‌سازی را می‌توان به دو دسته مسائل بهینه‌سازی ایستا و مسائل بهینه‌سازی پویا تقسیم کرد. در حل مسائل بهینه‌سازی پویا الگوریتم‌های تکاملی به‌سادگی قادر به ردیابی بهینه مطلوب نیستند. در یک محیط ایستا به این دلیل که بهینه ثابت است و تغییر نمی‌کند، یافتن بهینه مطلوب برای الگوریتم‌های تکاملی، کار نسبتاً راحتی است. اگر محیط پویا باشد و دچار تغییر شود، یافتن بهینه مطلوب بعد از هر تغییر در محیط، کار آسانی است. بنابراین برای حل مسائل بهینه‌سازی پویا نیاز به روش‌های ابتکاری مناسب است. در بهینه‌سازی مسائل پیچیده، استفاده از روش‌های بهینه‌سازی دقیق غیرممکن است، بنابراین از روش‌های جست‌وجوی تصادفی برای رسیدن به یک پاسخ نزدیک به بهینه استفاده می‌شود. در واقع در این نوع از الگوریتم‌ها پاسخ‌های مناسب در یک بازه زمانی قابل‌قبول به دست خواهند آمد، اما هیچ تضمینی برای به‌دست‌آوردن بهترین پاسخ وجود ندارد.

یکی از عیوب الگوریتم‌های تکاملی که به‌تنهایی برای حل مسائل بهینه‌سازی پویا استفاده می‌شوند آن است که ممکن است پاسخ‌هایی که در اجراهای مختلف الگوریتم به دست می‌آیند تفاوت‌های زیادی داشته باشند و در بسیاری از موارد حتی غیرقابل‌اعتماد به نظر می‌رسند. بنابراین برای حل این مشکل از راهکارهایی از جمله روش‌های ترکیبی [۱] در الگوریتم‌های تکاملی برای حل مسائل بهینه‌سازی

*. پست الکترونیک نویسنده مسئول:
m.mohammadpour@iauyasooj.ac.ir
۱. باشگاه پژوهشگران جوان و نخبگان واحد یاسوج، دانشگاه آزاد اسلامی یاسوج
۲. استادیار مهندسی کامپیوتر، دانشگاه آزاد اسلامی واحد علوم تحقیقات یاسوج

پویا استفاده می‌شود. از جمله این راهکارها در [۱] آمده است.

در اینجا سعی شده با ترکیب یک راهکار مناسب با الگوریتم کلونی زنبور مصنوعی مدلی ارائه شود که به یک موازنه در جست‌وجو برسد، به‌طوری که دچار هم‌گرایی زودرس نشود و همچنین تنوع لازم را برای اکتشاف در حین اجرای الگوریتم داشته باشد. برای جلوگیری از هم‌گرایی زودرس در این روش از یک حافظه مناسب استفاده شده که می‌تواند تنوع را در محیط به‌خوبی به وجود بیاورد.

این مقاله شامل ۵ بخش است. در بخش ۱ مقدمه و کلیات بیان شده است. در بخش ۲ مروری بر کارهای گذشته صورت گرفته و در بخش ۳ الگوریتم کلونی زنبور مصنوعی تشریح شده است. در بخش ۴ نظریه آشوب بیان شده است. بخش ۵ حافظه و انواع حافظه را تشرح می‌کند. بخش ۶ محیط‌های پویا را به‌طور خلاصه بیان می‌کند. بخش ۷ نحوه محاسبه خطا را در محیط پویا تشریح می‌کند. بخش ۸ روش پیشنهادی را به‌طور مفصل بیان می‌کند. بخش ۹ نتایج تجربی را برای روش پیشنهادی تشریح می‌کند. بخش ۱۰ نتیجه‌گیری کلی و پیشنهادهای آتی را بیان می‌کند.

۲- مروری بر کارهای گذشته

برای حل مسائل بهینه‌سازی پویا روش‌های مختلفی ارائه شده است که هرکدام از این روش‌ها به‌نوعی چالش‌های موجود در محیط‌های پویا را برطرف می‌کنند. برخی از این چالش‌ها می‌تواند شامل مشکل به‌روزکردن حافظه بعد از هر تغییر در محیط، حفظ تنوع[1] در محیط بعد از هم‌گرایی ذرات به بهینه و ظرفیت محدود حافظه باشد.

روش MIGA[2] ارائه‌شده در مرجع [۲]، ترکیبی از الگوریتم ژنتیک با مهاجرت تصادفی و حافظه است. در این روش از یک حافظه از نوع مهاجرتی[3] استفاده شده است. در این روش زمانی‌که محیط تغییر می‌کند از روش ارزیابی مجدد استفاده می‌شود و بهترین فرد از حافظه نامزد مهاجرت به جمعیت اصلی می‌شود. تعداد مهاجرت‌ها در این روش یک درصدی (r_i) از کل افراد جمعیت (n) است. مهاجرت‌ها جایگزین بدترین فرد از جمعیت اصلی $((r_i) * n)$ می‌شود. در این روش برای حفظ تنوع در جمعیت از روشی به نام مهاجران تصادفی استفاده می‌شود. در حقیقت با مهاجرت تصادفی کروموزوم‌ها از منطقه‌ای به منطقه دیگر، می‌توان تنوع را در نسل‌های به‌وجودآمده افزایش داد.

الگوریتم چندجمعیتی مبتنی بر ازدحام ذرات با عنوان MPSO[4] توسط کاموسی و همکارانش در مرجع [۳] ارائه شده است. در این الگوریتم حداکثر تعداد گروه‌های فعال در محیط محدود شده است و در هر مرحله، آن گروه‌هایی که دارای میزان برازندگی بهتری هستند فعال می‌شوند. این کار سبب می‌شود الگوریتم پیشنهادی سریع‌تر بتواند جواب‌های مطلوب را در فضای جست‌وجو کشف کند. این الگوریتم دارای عملگرهای مختلفی از جمله عملگر ضدهم‌گرایی، گروه غیرفعال، شعاع دفع و گروه فعال است. در این الگوریتم کارهایی که پس از مشاهده تغییر در محیط صورت می‌گیرد به شرح زیر است:

ابتدا همه گروها غیرفعال می‌شوند، سپس بهترین موقعیت در هریک از گروه‌ها یافت می‌شود و بقیه ذرات مربوط به آن گروه در داخل ابر کره‌ای به مرکزیت بهترین موقعیت یافت‌شده این گروه و به شعاع r_{cloud} بر اساس توزیع یکنواخت قرار می‌گیرند. سپس میزان برازندگی موقعیت جدید محاسبه می‌شود و بهترین تجربه شخصی هریک از ذره‌ها برابر این موقعیت قرار می‌گیرد و بهترین تجربه جمعی هریک از گروه‌ها به‌روزرسانی می‌شود. سپس به تعداد گروه‌های فعال از گروه‌هایی که میزان برازندگی بهترین تجربه جمعی آن‌ها بهتر است، فعال می‌شود[۳].

الگوریتم mQSO[5] [۴] به الگوریتم ازدحام ذرات مبتنی بر ذرات کوانتوم معروف است. ذرات کوانتوم در [۵] به‌عنوان ابزاری برای حفظ سطح خاصی از تنوع در یک گروه ارائه شده‌اند. در این الگوریتم کل جمعیت به چند گروه تقسیم می‌شوند و شامل سه عملگر تنوع با نام‌های ذرات کوانتوم، دفع و ضدهم‌گرایی است. ذرات کوانتوم در موقعیت‌های تصادفی قرار می‌گیرند تا تنوع گروه‌ها را حفظ کنند. عملگر دفع هرگاه دو گروه هم‌پوشانی پیدا می‌کنند، به گروه بدتر، مقداردهی اولیه مجدد می‌کند. عملگر ضدهم‌گرایی زمانی‌که تمام گروه‌ها هم‌گرا می‌شوند، به گروه بدتر، مقداردهی اولیه مجدد می‌کند. Blackwell و همکارانش الگوریتم Adaptive mQSO[6] [۶] را مطرح کردند که در آن تعداد گروه‌ها از ابتدا معین است و با پیداکردن قله‌های

[1] Diversity
[2] Memory Immigrants Genetic Algorithm
[3] Memory-Immigrants
[4] Multi Swarm PSO
[5] Multi Swarm Quantum Exclusion PSO
[6] Adaptive Multi Swarm Quantum Exclusion PSO

جدید افزایش می‌یابد، به این صورت که در آن هرگاه همه گروه‌ها هم‌گرا شدند، عملگر ضدهم‌گرایی یک گروه آزاد جدید ایجاد می‌کند که به یافتن بهینه محلی جدید کمک می‌کند. در مرجع [۷] روشی ارائه شده است که همه ذرات پس از مقداردهی به دو بخش تقسیم می‌شوند که آن‌ها را به‌ترتیب بخش یک و بخش دو می‌نامیم. این دو بخش نقش‌های متفاوتی برای جست‌وجو در محیط‌های پویا با به‌کاربردن راهبردهای متفاوت بر عهده می‌گیرند که آن‌ها در اینجا معرفی می‌کنیم. نقش بخش اول جست‌وجوی سریع برای یافتن بهینه سراسری در محیط جاری است. بنابراین عملیاتی مشابه PSO استاندارد برای تنظیم هم‌گرایی به کار رفته است. علاوه بر آن برای افزایش توانایی جست‌وجوی محلی در بخش یک جست‌وجوگر محلی گوسی معرفی شده است. نقش قسمت دوم گسترش ناحیه جست‌وجوی الگوریتم و پیداکردن نواحی جدید برای جست‌وجو است. برای دستیابی به این هدف در بخش دوم هر ذره با احتمال ۰,۵ به‌سمت بهترین موقعیت یافت‌شده یکی از ذرات گروه اول که به‌طور تصادفی انتخاب می‌شود حرکت می‌کند و با احتمال ۰,۵ از این نقطه دورتر می‌شود. در الگوریتم CPSO[1] [۸] ذرات را به خوشه‌هایی تقسیم کرده که ذرات موجود در هر خوشه به جست‌وجوی محلی در آن خوشه می‌پردازند. هر ذره به‌سمت میانگین بهترین موقعیت ملاقات‌شده توسط ذرات در خوشه‌ای که در آن قرار دارد حرکت می‌کند. در این الگوریتم از معیاری به نام درجه روی‌هم‌افتادگی[2] ذرات استفاده می‌شود که اگر در ناحیه‌ای از فضای جست‌وجو شلوغی ذرات به وجود آمد به‌صورت تصادفی بعضی از ذرات که در این ناحیه ایجاد ازدحام کرده‌اند، به ناحیه خلوت‌تری از فضای جست‌وجو مهاجرت تصادفی[3] کنند. دستیابی به جواب بهینه برای توابع چندقله‌ای در محیط‌های پویا با استفاده از ایجاد تغییرات در الگوریتم تکاملی بهینه‌سازی گروه ذرات است. برای دستیابی به این امر، از یک مدل مبتنی بر اجزا که اجازه توسعه به زیرجمعیت‌های موازی را می‌دهد، استفاده شده و جهت خوشه‌بندی این جمعیت‌ها روش k-means به کار رفته است. در FMSO[4] [۹] یک گروه والد به‌عنوان یک گروه پایه برای شناسایی نواحی امیدبخش استفاده شده است و یک دسته از گروه‌های فرزند به‌منظور جست‌وجوی محلی در زیر فضای مربوط به خود آن‌ها به کار رفته است. هریک از گروه‌های فرزند ناحیه جست‌وجوی شخصی خود را دارد که این ناحیه به شکل یک کره با شعاع r و به مرکز بهترین ذره آن گروه در نظر گرفته می‌شود. از این رو یک ذره $\vec{x}$ با فاصله کمتر از r از بهترین ذره ($\vec{s}$) آن گروه در نظر گرفته می‌شود. در این صورت یک ذره $\vec{x}$ با فاصله کمتر از r از $\vec{s}$ به آن گروه فرزند تعلق دارد. یک الگوریتم بهبودیافته برای بهینه‌سازی در محیط‌های پویا در مرجع [۲۶] ارائه شده است. این الگوریتم مبتنی بر ازدحام ذرات است. در مرجع [۲۷] نصیری و میبدی یک روش مبتنی بر الگوریتم کرم شب‌تاب را برای بهینه‌سازی در محیط‌های پویا ارائه کرده‌اند. در این روش از مدل مبتنی بر اجزا برای حفظ تنوع استفاده شده است. در مرجع [۲۸] نوذری و همکارانش یک الگوریتم بهینه‌سازی مبتنی بر تکاملات تفاضلی سلولی برای بهینه‌سازی در محیط‌های پویا ارائه کرده‌اند.

۳- الگوریتم کلونی زنبور مصنوعی

در سال‌های اخیر علاقه رو به رشدی در حوزه هوش ازدحامی در مسائل بهینه‌سازی پویا بنابر اهمیت آن‌ها در دنیای واقعی ایجاد شده است. هوش دسته‌جمعی یک حوزه جدید تحقیقاتی است که بر روی مطالعه و مدل‌سازی رفتار اجتماعی حشراتی از جمله زنبورها تمرکز دارد. الگوریتم کلونی زنبور مصنوعی [۱۰]، رفتار کاوشگرانه و هوشمندانه مجموعه‌ای از زنبورها را شبیه‌سازی می‌کند. الگوریتم زنبور مصنوعی شامل شش مرحله اساسی است. در ادامه، این مراحل به‌صورت خلاصه توضیح داده شده است.

۳-۱- مراحل الگوریتم کلونی زنبور مصنوعی

مرحله ۱: ایجاد پارامترهای الگوریتم کلونی زنبور مصنوعی و پارامترهای مسئله بهینه‌سازی

به‌طور عمومی، مسئله بهینه‌سازی می‌تواند به‌صورت رابطه (۱) فرمول‌بندی شود.

$$min\, f(x) = \{\boldsymbol{X}|\boldsymbol{x} \in \boldsymbol{X}\},$$
$$g(x) < 0\ and\ h(x) = 0 \qquad (۱)$$

که در رابطه (۱)، $f(x)$ تابع هدف است که باید کمینه شود. x مجموعه متغیرهای تصمیم (زنبورها) است که $\{x_i|i = 1, \dots, N\}$.

[1] Clustering PSO
[2] Overlap Degree
[3] Random Immigrant
[4] Fast Multi Swarm PSO

X محدوده ممکن برای هر متغیر تصمیم است که:
$X = \{X_1, X_2, \dots, X_N\}$ و $X_i \in (LB_i, UB_i)$ است که LB_i و UB_i به‌ترتیب کران پایین و کران بالای متغیر تصمیم x_i است. N تعداد متغیرهای تصمیم (تعداد زنبورها) است و $g(x)$ و $h(x)$ به ترتیب محدودیت‌های کیفیت و عدم کیفیت راه‌حل‌ها هستند.

الگوریتم کلونی زنبور مصنوعی شامل سه پارامتر دیگر نیز است. این پارامترها عبارت‌اند از:

الف) پارامتر (SN)، که تعداد منابع غذایی (راه‌حل‌ها) در جمعیت است. تعداد زنبورهای کارگر با SN برابر است.

ب) پارامتر ماکزیمم سیکل برای الگوریتم (MCN)، که شامل ماکزیمم تعداد نسل‌ها می‌شود.

ج) پارامتر حد (Limit)، که تعداد دفعات حرکت یک زنبور به‌سمت یک موقعیت با عدم بهبود را مشخص می‌کند. از این پارامتر برای تنوع در جست‌وجوی منابع غذایی (راه‌حل‌ها) استفاده می‌شود. در صورتی‌که تعداد دفعات حرکت یک زنبور به‌سمت یک موقعیت با عدم بهبود بیشتر از پارامتر حد شود، زنبور آن موقعیت را به‌عنوان موقعیت متروک شناسایی می‌کند.

مرحله ۲: ایجاد حافظه منابع غذایی

حافظه منبع غذایی (FSM) شامل یک ماتریس $SN \times N$ بعدی است که در هر سطر، یک منبع غذایی را به‌صورت رابطه (۲) شامل می‌شود. بردارها با توجه به مقدار تابع هزینه مجاورت به‌صورت صعودی مرتب می‌شوند.

$$FSM = \begin{bmatrix} x_1(1) & \cdots & x_1(2) & \cdots & x_1(N) \\ \vdots & & \vdots & \ddots & \vdots \\ x_{SN}(1) & & x_{SN}(2) & \cdots & x_{SN}(N) \end{bmatrix} \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_{SN}) \end{bmatrix} \quad (۲)$$

به‌صورت عمومی هر بردار $x_j(i)$ (موقعیت زنبور i ام در بعد j ام) به‌صورت رابطه (۳) تولید می‌شود.

$$x_j(i) = LB_i + (UB_i - LB_i) \times r$$
$$\forall\, j \in (1,2,\dots,SN) \quad (۳)$$
$$\forall\, i \in (1,2,\dots,N)$$

در این رابطه r یک عدد تصادفی با توزیع یکنواخت در بازه $[0,1]$ است.

مرحله ۳: تخصیص زنبورهای کارگر به منابع غذایی

در این مرحله هر زنبور کارگر به‌صورت تصادفی به‌سمت منابع غذایی حرکت و میزان شایستگی را برای آن موقعیتی که در آن واقع شده است محاسبه می‌کند. هر زنبور یک همسایه را به‌تصادف انتخاب کرده، به‌سمت آن حرکت می‌کند. زنبور کارگر موقعیت جدید خود را طبق رابطه (۴) اصلاح می‌کند.

$$x'(i) = x_j(i) \pm r\left(x_j(i) - x_k(i)\right)$$
$$\forall k \in (1,2,\dots,SN), \quad (۴)$$
$$k \neq j \;\; and \;\; r \in [0,1]$$

که در رابطه (۴)، $x_j(i)$ موقعیت زنبور i ام در بعد j ام است و $x_k(i)$ موقعیت زنبور همسایه را مشخص می‌کند. r یک عدد تصادفی با توزیع یکنواخت در بازه $[0,1]$ است. شبه‌کد مربوط به فاز زنبور کارگر به‌صورت شکل (۱) است.

مرحله ۴: ارسال زنبورهای تماشاچی

فاز زنبور تماشاچی شامل سه مرحله به شرح زیر است:

۱) تخصیص احتمال انتخاب‌شدن به هر زنبور کارگر طبق رابطه (۵).

$$P_j = \frac{f(x_j)}{\sum_{j=1}^{SN} f(x_j)} \quad (۵)$$

۲) احتمال اینکه زنبور کارگر توسط زنبور تماشاچی طبق رابطه (۵) انتخاب شود. این موضوع در واقع همان انتخاب بر اساس احتمال در چرخ رولت است؛ یعنی هر قدر یک موقعیت احتمال بیشتری داشته باشد، شانس انتخاب‌شدن آن موقعیت نیز بیشتر خواهد شد.

۳) هر زنبور تماشاچی، یک زنبور را به‌تصادف انتخاب می‌کند و به‌سمت آن حرکت می‌کند. اگر کارایی موقعیت جدید از موقعیت قبلی بیشتر باشد، زنبور موقعیت جدید را برمی‌گزیند، در غیر این صورت به موقعیت قبلی خود بازمی‌گردد و به شاخص محاکمه[1] خود یک واحد اضافه می‌کند. زنبور موقعیت جدید خود را طبق رابطه (۴) اصلاح می‌کند، شاخص محاکمه شمارنده تعداد دفعات متوالی حرکت زنبور با عدم بهبود است. اگر زنبوری مقدار شاخص محاکمه‌اش از حد[2] معین‌شده بیشتر شود، به این معنی است که آن منطقه غذایی دیگر شهدی ندارد و آن منطقه را باید ترک کرد. شبه‌کد مربوط به فاز زنبور تماشاچی به‌صورت شکل (۲) است. در این شبه‌کد sum_prob احتمال تجمعی انتخاب‌شدن زنبورهای کارگر توسط

[1] Trial

[2] Limit

زنبورهای تماشاچی است.

Algorithm 1: Employed bee Phase

1: for j=1…SN do
2: for i=1…N do
3: $x'(i) = x_j(i) \pm r(x_j(i) - x_k(i))$
$\forall k \in (1,2,\dots,SN), k \neq j$
4: end for
5: calculate $f(x_i)$
6: $if\ (f(x') \leq f(x_i))\ then\ x_i = x'\ and\ f(x_i) = f(x)'$
7: end of if
8: end for

شکل ۱: شبه‌کد مربوط به فاز زنبور کارگر

Algorithm 2: Onlooker bee Phase

1: for j =1…SN do
2: sum_prob = 0 , $r \in [0,1]$
3: j =0
4: while (sum_prob ≤ r) do
5: sum_prob= sum_prob+P_j
6: $j = j + 1$
7: for k=1…N do
8: $x'(j) = x_j(k) \pm r(x_j(k) - x_j(m))$
$\forall m \in (1,2,\dots,SN)$
9: end for
10: calculate $f(x_j)$
11: $if\ \left(f(x') \leq f(x_j)\right) then\ x_j = x'\ and\ f(x_j) = f(x)'$
12: end of if
13: end for

شکل ۲: شبه‌کد مربوط به فاز زنبور تماشاچی

Algorithm 3: Scout bee Phase

1: for i =1…SN do
2: if (scout(i) = Limit) then
3: generate x_j using equation (3)
4: end if
5: end for

شکل ۳: شبه‌کد مربوط به فاز زنبور پیشاهنگ

مرحله ۵: ارسال زنبورهای پیشاهنگ برای جست‌وجوی منابع غذایی جدید

زنبوری که در فضای جست‌وجو وظیفه اکتشاف راه‌حل‌های جدید را برعهده دارد، زنبور پیشاهنگ است. این زنبور در فضای جست‌وجو به‌صورت تصادفی حرکت می‌کند تا مناطق جدید را اکتشاف کند. این زنبور موظف است منابع غذایی متروک پیداشده را با منابع غذایی جدید با استفاده از جست‌وجوی تصادفی طبق رابطه (۳) جایگزین کند. شبه‌کد مربوط به فاز زنبور پیشاهنگ به‌صورت شکل (۳) است.

مرحله ۶: به‌خاطرسپاری بهترین منبع غذایی

در این مرحله شایستگی و موقعیت بهترین منبع غذایی، x^{best} پیداشده تا کنون در حافظه منبع غذایی، ذخیره می‌شود.

شبه‌کد مربوط به الگوریتم کلونی زنبور مصنوعی در حالت کلی به‌صورت شکل (۴) است.

۴- نظریه آشوب

تئوری آشوب[1] شاخه‌ای از ریاضیات است که به مطالعه رفتار دینامیکی سیستم‌هایی می‌پردازد که به انتخاب‌های اولیه بسیار حساس هستند. تغییرات کوچک در انتخاب‌های اولیه (مثل خطای گردکردن در محاسبات عددی) باعث واگرایی نتایج سیستم‌های دینامیکی می‌شود و در حالت کلی پیش‌بینی‌های بلندمدت را غیرممکن می‌سازد [۱۱]. حتی اگر سیستم‌ها قطعی[2] باشند این اتفاق رخ می‌دهد؛ به این معنا که رفتار آینده سیستم به‌طور کامل به شرایط اولیه بستگی دارد [۱۱]. به عبارت دیگر، ماهیت قطعی این سیستم‌ها آن‌ها را غیرقابل‌پیش‌بینی می‌سازد. این رفتار به‌عنوان آشوب قطعی[3] یا به‌صورت خلاصه آشوب شناخته شده است. پدیده‌های تصادفی[4] که تاکنون با عجز و ناتوانی، دلیلی برای آن‌ها نمی‌یافتیم، به‌کمک نظریه آشوب، توجیه می‌شوند. نظریه آشوب، بر پایه‌های ریاضی، فیزیک و حتی فلسفه استوار است. هریک از این علوم، با ابزارهای خود این نظریه را بررسی و ثابت کرده‌اند. نظریه آشوب پدیده جدیدی نیست، قانون علت و معلول در آن پا برجاست و فقط با ابزارهای متفاوتی، علت‌های بیشتری را برای یک معلول بررسی می‌کند.

نمونه مشهور یک سیستم آشوب‌گونه، مدل جمعیتی نگاشت لجستیک[5] است. از ویژگی‌های تئوری آشوب می‌توان به خودسازماندهی[6] (وفق‌دادن خود با شرایط

[1] Chaos
[2] Deterministic
[3] Deterministic Chaos
[4] Random
[5] Logistic Map
[6] Self Organization

محیطی) در محیط‌های پویا و خودمانایی[1](هر جزئی از سیستم دارای ویژگی کل بوده و مشابه آن است) و حساسیت به شرایط اولیه اشاره کرد. شکل (۵) حساس‌بودن سیستم آشوب‌گونه را به شرایط اولیه نشان می‌دهد. در این شکل مشخص است که در سیستم‌های آشوب‌گونه یک واگرایی اولیه کوچک دل‌خواه نه‌تنها کوچک باقی نمی‌ماند بلکه به‌طور نمایی رشد می‌کند.

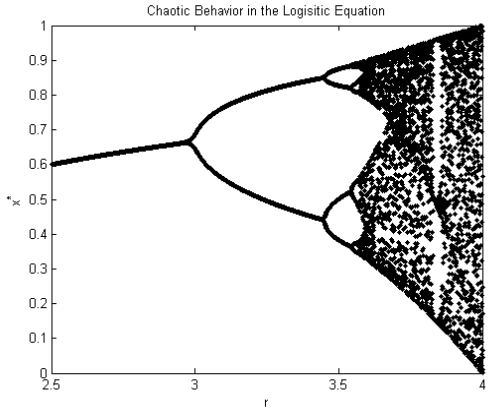
همچنین با توجه شکل، به‌وضوح می‌توان گفت اگر از هرکدام از نقاط انتهایی شروع به حرکت کنیم، در نهایت به نقطه شروع هم‌گرا خواهیم شد که این از خواص یک سیستم آشوبناک است.

Algorithm4: Artificial bee colony algorithm

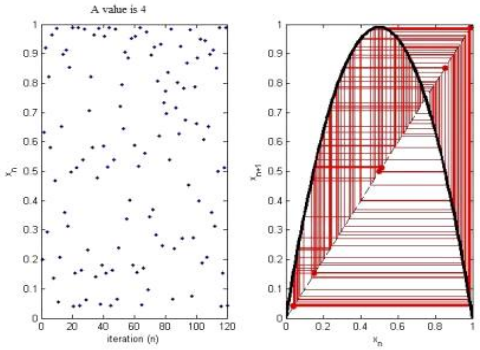
Input:
Algorithm parameters: $N, SN, D, MCN, Limit, LB, UB$
Output:
BEST Solution, BEST Fitness
1. Begin
% initialization%
2. Initialize solution population by equation (3)
% Evaluate population%
3. For i $= 1$ to SN
4. $f_i = f(x_i)$
5. End
6. $cycle = 1$
$Repeat$:
% employ bee phase%
7. $For\ i = 1\ to\ SN$
8. $Produce\ new\ solutions\ x_i^{new}\ for\ employ\ bee\ by\ (4)$
9. $Apply\ the\ greedy\ selection\ process$
10. $End\ For$
11. $Apply\ the\ ranking\ evaluation\ bees$
% $end\ of\ employ\ bee\ phase$%
% $Onlooker\ bee\ phase$%
12. $For\ i = 1\ to\ SN$
13. $sharing\ the\ entire\ solutions\ between\ employ\ bees\ and\ onlooker\ bees$
14. $select\ the\ best\ solution\ with\ roulette\ wheel\ strategy\ by\ (5)$
15. $Produce\ new\ solutions\ x_k^{new}(i)\ for\ onlooker\ bees\ by\ (4)$
16. $Apply\ the\ greedy\ selection\ process$
17. $End\ For$
% $end\ of\ onlooker\ bee\ phase$%
% $scout\ bee\ phase$%
18. $if\ Trial > Limit\ then$
19. $replace\ the\ solution\ with\ a\ new\ randomly\ produced\ solution\ \boldsymbol{x}_j\ by\ (3)$
% $end\ of\ scout\ bee\ phase$%
20. $Save\ in\ memory\ the\ best\ solution\ so\ far$
21. $cycle = cycle + 1$
22. $Until\ cycle = MCN$
23. End

شکل ۴: شبه‌کد الگوریتم کلونی زنبور مصنوعی

[1] Self Similarity



شکل ۵: حساسیت سیستم آشوب‌گونه به شرایط اولیه



شکل ۶: نمودار تابع لجستیک با مقدار $A = 4$

در مدل‌های آشوب‌گونه خطای پیش‌بینی به‌سرعت رشد می‌کند و در نتیجه پیش‌بینی می‌تواند فقط برای مدت زمان محدود و با یک خطای پیش‌بینی مجاز صورت پذیرد؛ زیرا وضعیت فعلی سیستم به وضعیت قبلی آن وابسته است، در حالی‌که در یک سیستم تصادفی، وضعیت جاری سیستم از وضعیت پیشین آن مستقل است. در مدل تصادفی، حتی برای یک مدت زمان کوتاه هم نمی‌توان پیش‌بینی دقیق از خروجی مدل داشت.

نمونه مشهور نگاشت لجیستیک به‌صورت فرمال به‌صورت رابطه (۶) است.

$$r_{n+1} = Ar_n(1 - r_n) \qquad (۶)$$

در رابطه (۶)، r_n یک عدد حقیقی در بازه $[۰,۱]$ است و پارامتر A که به ضریب لجستیک معروف است، موجب بروز ویژگی‌های منحصربه‌فردی در این تابع می‌شود. در این مقاله مقدار پارامتر A برابر ۴ در نظر گرفته شده است. فرض کنید پارامتر $A = 2.5$ و نقطه شروع عددی نزدیک به ۰,۰۲۵۰۰۰ باشد، در این صورت مقادیر مختلف برای تابع لجستیک به‌صورت توالی اعداد از سمت چپ به راست به‌صورت زیر به دست می‌آید:

۰,۰۶۰۹۳-۰,۱۴۳۰-۰,۳۱۲۵-۰,۵۳۰۶۳-۰,۶۱۲۵۲۷-
۰,۵۹۳۳۴-۰,۶۰۳۲۱۶-۰,۵۹۸۳۶-۰,۶۰۰۸۱۱

در این توالی از اعداد می‌بینیم که در توالی پنجم به‌بعد اعداد به‌سمت عدد ۰,۶ هم‌گرایی پیدا می‌کنند. نمونه‌های مختلف دیگری از سیستم‌های آشوبناک نیز وجود دارند که در اینجا به معرفی آن‌ها نمی‌پردازیم. شکل (۶) نمودار مربوط به تابع لجستیک با پارامتر $A = 4$ را نشان می‌دهد.

۵- حافظه

ایده حفظ افراد نخبه (راه‌حل‌های خوب) در یک حافظه صریح [۱۲] برای الگوریتم به نظر جذاب است. بدین ترتیب اگر بهینه دوباره در مکان قبلی خود ظاهر شود یک حافظه می‌تواند به‌راحتی آن را به خاطر آورد و خیلی سریع جمعیت را به‌سمت آن حرکت دهد. از این حافظه به‌عنوان یک حافظه صریح برای نگهداری یک کپی از بهترین راه‌حل‌ها استفاده می‌شود. یکی از خواص محیط‌های پویا، چرخه‌ای‌بودن است. اگر یک محیط چرخه‌ای باشد، به‌معنای آن است که نقاط بهینه ممکن است به نقاط قبلی خود بازگردند. بنابراین این حافظه کمک می‌کند اگر بهینه‌ای به نقطه قبل از خود برگشت، بتوان به‌راحتی آن را ردیابی کرد و در نتیجه سرعت هم‌گرایی الگوریتم را افزایش داد. راه‌حل‌هایی در حافظه ذخیره می‌شوند که خیلی قدیمی نباشند و در مکان‌های مختلف فضای جست‌وجو قرار گرفته باشند. پس به نظر می‌رسد در هر زمان باید بهترین راه‌حل‌ها در حافظه قرار گیرند.

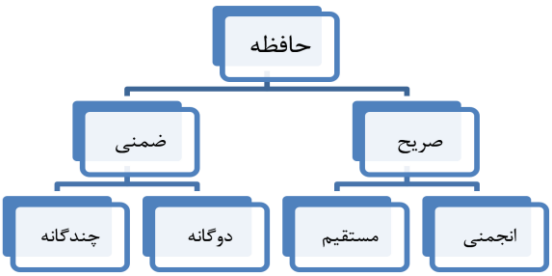
به‌طور کلی حافظه به دو دسته تقسیم می‌شود، یکی حافظه صریح[۱] و دیگری حافظه ضمنی[۲]. از حافظه ضمنی برای ذخیره تمام اطلاعات (اطلاعات اضافی را نیز شامل می‌شود) استفاده می‌شود. در حقیقت از این حافظه برای ذخیره تمامی اطلاعات یک کروموزم (هر کروموزم دو یا بیشتر الل[۳] دارند) استفاده می‌شود. اطلاعات هم‌گرایی یعنی توزیع الل در جمعیت می‌تواند به‌عنوان نمایش طبیعی محیط جاری به کار رود. از جمله کاربردهای حافظه ضمنی چندگانه در مرجع [۱۳] آمده است. حافظه ضمنی به دو دسته تقسیم می‌شود، یکی حافظه دوگانه[۴] و دیگری حافظه چندگانه[۵].

[1] Explicit Memory
[2] Implicit Memories
[3] Alleles
[4] Dualism
[5] Diploid

حافظه صریح برای ذخیره اطلاعات مفید از محیط استفاده می‌شود و برخلاف حافظه ضمنی که حتی اطلاعات اضافی را نیز ذخیره می‌کند، فقط اطلاعات مفید را ذخیره می‌کند. حافظه صریح شامل دو نوع حافظه مستقیم[1] و حافظه انجمنی[2] است[۱۴].

در روش حافظه مستقیم راه‌حل‌های خوب به‌دست‌آمده توسط هر فرد (اطلاعات محلی) و یا راه‌حل‌های به‌دست‌آمده توسط همه افراد جمعیت (اطلاعات عمومی) به‌صورت مستقیم در حافظه ذخیره می‌شوند و در محیط‌های جدید مورد استفاده مجدد قرار می‌گیرند[۱۵].

در حافظه انجمنی اطلاعات محیطی همانند راه‌حل‌های خوب ذخیره می‌شوند. از جمله اطلاعات ذخیره‌شده در این نوع حافظه می‌توان، ذخیره لیستی از حالات فضای مسئله یا احتمال رخداد یک جواب خوب در فضای مسئله را نام برد [۱۶] که در محیط جدید مورد استفاده مجدد قرار می‌گیرند. شکل (۷) تقسیم‌بندی حافظه را برای استفاده در محیط‌های پویا نشان می‌دهد.



شکل ۷: تقسیم‌بندی حافظه برای محیط‌های پویا

۵-۱- راهکارهای جایگزینی در حافظه

برای جایگزینی افراد جمعیت در حافظه، استراتژی‌های مختلفی ارائه شده است که چند نمونه از این استراتژی‌های جایگزینی که در مرجع [۷] آمده، بیان می‌شود:

راهکار ۱. در این راهکار دو فرد در حافظه با کمترین فاصله (بیشترین شباهت) انتخاب و فردی که کارایی کمتری داشته باشد کاندیدای جایگزینی می‌شود. برای مثال اگر fit(i) را به‌عنوان کارایی فرد i ام و fit(j) را به‌عنوان کارایی فرد j ام در نظر بگیریم، اگر $fit(i)<fit(j)$ بهترین فرد جمعیت جایگزین $fit(i)$ می‌شود و برعکس.

راهکار ۲. اگر $fit(j) \times \frac{d_{ij}}{d_{max}} \leq fit(new)$ باشد در این صورت فرد j ام با بهترین فرد فعلی جایگزین می‌شود.

که در این رابطه $fit(j)$ را کارایی فرد j ام و d_{ij} فاصله بین دو فرد i و j، d_{max} ماکزیمم فاصله ممکن بین دو فرد i و j است.

راهکار ۳. این راهکار به روش مشابهت معروف است. در این راهکار بهترین فرد فعلی از جمعیت، جایگزین شبیه‌ترین فرد در حافظه می‌شود، به شرطی‌که این فرد دارای کارایی بیشتری نسبت به فرد حافظه باشد. برای اندازه‌گیری شباهت می‌توان از فاصله اقلیدسی استفاده کرد. فاصله اقلیدسی میان دو فرد i و j طبق رابطه (۷) محاسبه می‌شود.

$$d(i,j) = \sqrt{\sum_{d=1}^{D} (x_i^d - x_j^d)^2} \qquad (۷)$$

۵-۲- بازیابی از حافظه

از اطلاعات ذخیره‌شده در حافظه باید برای ردیابی بهینه جدید استفاده کرد. بنابراین بهترین زمان برای بازیابی اطلاعات از حافظه، لحظه‌ای است که محیط دچار تغییر می‌شود. برای بازیابی از حافظه می‌توان استراتژی‌های مختلفی اتخاذ کرد. یکی از استراتژی‌های بازیابی از حافظه بدین صورت است که بهترین فرد از حافظه را مشخص و آن را جایگزین بدترین فرد از جمعیت اصلی کرد [۱۸و۱۷].

۶- محیط‌های پویا

محیط پویا محیطی است که در گذر زمان دچار تغییر می‌شود. در بسیاری از مسائل بهینه‌سازی دنیای واقعی، تابع هدف[3] یا محدودیت‌ها[4] می‌توانند در طول زمان تغییر یابند، بنابراین بهینه در این مسائل نیز می‌تواند تغییر یابد. در دنیای واقعی محیط در حال تغییر و تحول است، بنابراین بر اثر تغییرات محیطی رسیدن به هدف مطلوب نیز دست‌خوش تغییر می‌شود و برای رسیدن به بهینه مطلوب باید الگوریتم موردنظر بتواند به‌صورت کارا و مناسب تغییرات محیطی را دنبال کرده، بهینه مناسب را بعد از هر تغییر در محیط مشخص کند. اگر هریک از این رویدادهای نامعین در فرآیند بهینه‌سازی موردتوجه قرار گیرند، این مسئله پویا نامیده می‌شود.

۶-۱- لحظه تغییر در محیط

به چرخه‌ای[5] که در آن محیط (تابع بهینه‌سازی) تغییر

[1] Direct Memory
[2] Associative Memory
[3] Objective Function
[4] Constraints
[5] Cycle

می‌کند، لحظه تغییر در محیط گویند.
یکی از چالش‌های محیط‌های پویا شناسایی لحظه تغییر در محیط است. برای شناسایی زمان تغییر در محیط، یک ارزیابی مجدد برای محاسبه کارایی انجام می‌گیرد، اگر مقدار کارایی زنبور $\vec{L}$ در ارزیابی مجدد با مقدار قبلی‌اش برابر نباشد، در این صورت می‌توان گفت محیط دچار تغییر شده است و از این روش برای شناسایی لحظه تغییر استفاده می‌شود. رابطه (۸) بیانگر این موضوع است. اگر f_i میزان کارایی برای زنبور i ام قبل از تغییر محیط و $f_i^{'}$ میزان کارایی زنبور بعد از تغییر باشد، در این صورت رابطه (۸) برقرار است.

$$\begin{aligned} &if\ f_i \neq f_i^{'} \\ &then\ change\ detected\ in \\ &the\ environmen \end{aligned} \quad (۸)$$

۷- خطای یک کلونی

خطای یک کلونی x با تابع کارایی $f(\vec{L})$ به‌شکل رابطه (۹) تعریف می‌شود که در این رابطه x^* موقعیت زنبور بهینه (که آن را نداریم) است.

$$E\left(\boldsymbol{x}, f\left(\vec{L}\right)\right) = \min\left(f(\vec{x}_{:i})\right) - f(x^*) \quad (۹)$$

۷-۱- خطای آفلاین

برای اندازه‌گیری کارایی الگوریتم‌های تکاملی از نظر کمی از رابطه‌ای به نام خطای آفلاین [19] استفاده می‌شود. خطای آفلاین از رابطه (۱۰) به دست می‌آید.

$$Offlin\ Error = \frac{1}{FE_s}\sum_{t=1}^{FE_s}(h(t) - f(t)) \quad (۱۰)$$

که در رابطه (۱۰)، FE_s برابر بیشینه تعداد ارزیابی کارایی برای زنبورها، $h(t)$ مقدار بهینه سراسری قبل از تغییر و $f(t)$ بهترین موقعیت پیداشده توسط زنبورها در زمان t است.

۸- روش پیشنهادی

در این مقاله یک الگوریتم مبتنی بر کلونی زنبور مصنوعی آشوب‌گونه[1] مبتنی بر حافظه برای محیط‌های پویا پیشنهاد داده شده است. در این الگوریتم برخلاف الگوریتم کلونی زنبور مصنوعی که حرکات زنبورها برای اکتشاف مناطق جدید، تصادفی است از رفتار آشوب‌گونه برای حرکت زنبورها استفاده می‌شود. در طبیعت رفتار بسیاری از سیستم‌ها به‌صورت آشوب‌گونه است و در جهان واقعی سیستم‌ها به‌جای رفتارها و حرکات تصادفی از خود رفتارهای آشوب‌گونه نشان می‌دهند. از جمله این رفتارهای آشوب‌گونه را می‌توان در سیگنال‌های مغز انسان، موج دریا، حرکت بال پروانه، گردباد و غیره مشاهده کرد. در الگوریتم کلونی زنبور مصنوعی استاندارد جمعیت اولیه به‌صورت تصادفی ایجاد شده، ولی در این الگوریتم جمعیت اولیه بر اساس تئوری آشوب ایجاد می‌شود و در این روش از تابع دینامیکی غیرخطی نگاشت لجستیک که یکی از نگاشت‌های معروف تئوری آشوب است استفاده شده است. همچنین در این روش پیشنهادی از یک حافظه صریح برای ذخیره بهترین راه‌حل‌های خوب، جهت استفاده مجدد در محیط جدید استفاده می‌شود. در یک محیط پویا به‌دلیل چرخه‌ای‌بودن این محیط، ممکن است بهینه که در چند نسل قبل ظاهر شده، مجدداً در همان نقطه ظاهر شود و در این صورت یک حافظه می‌تواند به‌راحتی این بهینه را شناسایی کند و سرعت هم‌گرایی را در الگوریتم افزایش دهد. از استراتژی جایگزینی مناسبی که در ادامه آن را توضیح خواهیم داد برای به‌روزرسانی حافظه استفاده شده تا تنوع در حین اجرای الگوریتم حفظ شود. یکی از چالش‌های اصلی در محیط‌های پویا حفظ تنوع در حین اجرای الگوریتم است و در این روش با استفاده از حافظه صریح تنوع بعد از تغییر در محیط حفظ شده است. در این روش حافظه و جمعیت در مرحله اول طبق رابطه ۱۱ مقداردهی اولیه می‌شوند. استفاده از یک حافظه صریح مستقیم، می‌تواند به‌سرعت زنبورها را به بهینه تغییر یافته هم‌گرا کند و سرعت هم‌گرایی را به‌صورت مناسبی افزایش دهد. اگر بعد از یک هم‌گرایی زنبورها، محیط دچار تغییر شود، یک حافظه صریح می‌تواند به‌سرعت مکان بهینه جدید را به خاطر آورد و جمعیت را به‌سمت آن سوق دهد. اگر حافظه بعد از مدتی نیاز به به‌روزرسانی داشته باشد، ابتدا بدترین زنبور از جمعیت حافظه را بر اساس میزان کارایی زنبورها درون حافظه مشخص و آن را از حافظه خارج کرده، سپس بهترین زنبور از جمعیت اصلی (زنبوری که به مرکز بهینه جدید نزدیک‌تر است) جایگزین آن می‌شود. بدترین زنبور از جمعیت حافظه زنبوری است که بر اساس رتبه‌بندی جمعیت به‌صورت صعودی، در ابتدای این رتبه‌بندی قرار

[1] Chaotic Artificial Bee Colony Algorithm

می‌گیرد. در صورت تغییر در محیط باید از اطلاعات مفیدی که در حافظه ذخیره شده استفاده شود تا بهینه تغییریافته به‌سرعت ردیابی شود. جهت بازیابی اطلاعات از حافظه، بهترین زنبور از حافظه، جایگزین بدترین زنبور جمعیت اصلی می‌شود. بهترین زنبور همیشه بالاترین کارایی را دارد، پس زنبوری با کارایی بالا در ردیابی بهینه جدید می‌تواند بسیار مناسب واقع شود.

شبه‌کد مربوط به روش پیشنهادی به‌صورت شکل (۸) است.

Algorithm5: Proposed Algorithm

Input: *N, SN, D, MCN, Limit,* x_j^{min}, x_j^{max}, *mem_size*
Output: *BEST Solution, BEST Fitness, offline error*
1: Begin
2: initialize x and mem based logistic map
% x is population and mem is memory population
3: $f_i = fit(x_{:i})$ % $f_i$is fitness for population
4: $m_i = fit(mem_i)$ % m_i is fitness for memory population%
5: update_mem=1
update_time=rand (5,10)
6: $cycle = 1$
Repeat
7: update_mem:
update_mem=0
mem=update_memfunction(*mem, x*)
update_time=rand (5,10) + cycle
8: cx=update_popfunction(x) % cx is current *x*
9: $cf_i = fit(x_{:i})$ % fitness for current *x*
11: if $cf_i \neq f_i$ then chang_flag=1
% change detected
11: chang_flag: % reuse memory
1: $m_i = fit(mem_{:i})$, $f_i = fit(x_{:i})$
2: *select* $j_r \in m_r$
% j_r is best bees in memory
3: select $d_r \in x_r$
% d_r is worst bees in population
4: $x_{d_r} = mem_{j_r}$
12: if update_time≥cycle
then update_mem=1
13: $cycle = cycle + 1$
14: *Until cycle = MCN*
15: End

شکل ۸: شبه کد الگوریتم پیشنهادی

Algorithm 6: popfunction(x)

function {
Repeat
%employ bee phase
1: for j=1…SN do
2: for i=1…N do
3: $x'(i) = x_j(i) \pm Ar_n(1 - r_n)(x_j(i) - x_k(i)$
$\forall k \in (1,2,\dots,SN)\,, k \neq j$
4: end for
5: calculate $f(x_i)$
6: $if\ (f(x') \leq f(x_i))\ then\ x_i = x'\ and\ f(x_i) = f(x)'$
7: end of if
8: end for
%end of employ bee phase

%onlooker bee phase
9: for j=1…SN do
10: sum_prob = 0 , $r \in [0,1]$
11: *j=0*
12: while (sum_prob ≤ *r*) do
13: sum_prob= sum_prob+P_j
14: *j=j+1*
15: for k=1…N do
16: $x'(j) = x_j(k) \pm Ar_n(1 - r_n)(x_j(k) - x_j(m))$
$\forall m \in (1,2,\dots,SN)$
17: end for
18: calculate $f(x_j)$
19: $if\ \big(f(x') \leq f(x_j)\big) then\ x_j = x'\ and\ f(x_j) = f(x)'$
20: end of if
21: end for
%end of onlooker bee phase

%scout bee phase
22: for i=1…SN do
23: if (scout(i) = Limit) then
24: generate x_j using equation (3)
25: end if
26: end for
%end of scout bee phase
}

شکل ۹: شبه کد مربوط به تابع به‌روزرسانی زنبورها

شبه‌کد مربوط به فازهای الگوریتم کلونی زنبور مصنوعی برای به‌روزرسانی موقعیت خود در شکل (۹) آمده است. همان‌گونه که در بخش‌های گذشته بیان شد، زنبورها به‌جای حرکت تصادفی به‌سمت یکی از همسایگان خود از

حرکت آشوب‌گونه استفاده می‌کنند و موقعیت جدید خود را به‌روزرسانی می‌کنند. در حقیقت همان فازهای زنبور کارگر و تماشاچی و پیشاهنگ الگوریتم زنبور است، با این تفاوت که از حرکت آشوب‌گونه به‌جای حرکت تصادفی استفاده شده است. نحوه به‌روزرسانی موقعیت زنبورها در قالب یک تابع نوشته شده است که شبه‌کد این تابع به‌صورت شکل (۹) است. زنبورها طبق رابطه (۱۱) موقعیت جدید خود را به‌روزرسانی می‌کنند.

$$x'(i) = x_j(i) \pm Ar_n(1-r_n)(x_j(i) - x_k(i)$$
$$\forall k \in (1,2,\dots,SN)\,, k \neq j \qquad (۱۱)$$

در رابطه (۱۱)، $x'(i)$ موقعیت جدید زنبور i ام، $x_j(i)$ موقعیت قبلی زنبور i ام در jامین بعد و $x_k(i)$ موقعیت زنبور k در j امین بعد (زنبور همسایه) است. همان‌گونه که در رابطه (۱۱) مشخص است زنبورها به‌جای حرکت تصادفی از حرکت آشوب‌گونه برای جابه‌جایی استفاده می‌کنند. در این رابطه به‌جای مقدار تصادفی توزیع یکنواخت λ از تابع نگاشت لجستیک (تابع آشوب) استفاده شده است. همان‌گونه که در بخش‌های قبلی بیان شد رفتارهای آشوب‌گونه برای رسیدن به هم‌گرایی سریع‌تر الگوریتم کمک شایانی می‌کند که این موضوع نیز در شکل مربوط به تابع نگاشت لجستیک که در بخش‌های قبل آمد، مشاهده شد. شکل (۹) فازهای مربوط به الگوریتم کلونی زنبورها را نشان می‌دهد ولی با این تفاوت که در این روش به‌جای تصادفی‌سازی از آشوب استفاده کرده‌ایم. شبه‌کد نوشته‌شده در شکل (۹) همان‌گونه که مشخص است از سه مرحله مربوط به کلونی زنبورها تشکیل شده است. سه مرحله مربوط به الگوریتم کلونی زنبور مصنوعی شامل فاز زنبور کارگر، فاز زنبور تماشاچی و فاز زنبور پیشاهنگ است. در فاز زنبور کارگر، زنبورها به‌جای اینکه تصادفی به‌سمت یکی از همسایگان خود حرکت کنند، با حرکت آشوب‌گونه به‌سمت همسایه‌های خود می‌روند. در فاز زنبور تماشاچی نیز به همین شکل عمل می‌شود. جمعیت اولیه نیز بر اساس نظریه آشوب ایجاد می‌شود.

همان‌طور که قبلا بیان شد، برای به‌روزرسانی حافظه باید از راهکار جایگزینی استفاده شود. شبه‌کد مربوط به تابع به‌روزرسانی حافظه به‌صورت شکل (۱۰) آمده است. در این شبه‌کد نشان داده شده است که زنبورها در حافظه رتبه‌بندی شده و زنبوری که دارای کمترین کارایی است از حافظه خارج می‌شود.

Algorithm 7:memfunction(mem,x)

```
function
{
input:  mem: set of solutions in the memory;
s′: new good solution found; C: replacement
strategy
output:∅
1: select(C, mem, s′)
    % use the replacement strategy %
2: s ∈ mem|∀s′ ∈ mem,  if f(s′) ≥ f(s)
3: Remve s
4:   then mem ← s′ ∪ {mem}
5:  end of if
}
```

شکل ۱۰: شبه کد مربوط به تابع به‌روزرسانی حافظه

۸-۱- توضیحات مربوط به شبه‌کد الگوریتم پیشنهادی

در ابتدای الگوریتم پارامترهای لازم تعریف می‌شوند. در مرحله ۲ جمعیت باید ایجاد شود، در این الگوریتم برخلاف الگوریتم کلونی زنبور مصنوعی که از تصادفی‌سازی برای مقداردهی اولیه جمعیت استفاده می‌شود، جمعیت بر اساس تئوری آشوب و نگاشت لجستیک که یک سیستم دینامیکی غیرخطی گسسته آشوب‌گونه است مقداردهی اولیه می‌شوند. به‌جای مقدار تصادفی توزیع یکنواخت λ، از رابطه نگاشت لجستیک سیستم دینامیکی آشوب‌گونه که از رابطه (۶) محاسبه می‌شود استفاده می‌کند تا مقداردهی اولیه از تصادفی‌سازی اجتناب کند و همانند جهان واقعی مبتنی بر آشوب باشد. برای مقداردهی اولیه جمعیت از رابطه (۱۱) استفاده می‌شود. در این مقاله از یک حافظه نیز استفاده شده که این حافظه در ابتدا همانند جمعیت اصلی باید مقداردهی اولیه شود و طبق همان رابطه آشوب که جمعیت اصلی را مقداردهی اولیه می‌کند، جمعیت حافظه نیز مقداردهی اولیه می‌شود. مقداردهی جمعیت برای جمعیت اصلی و جمعیت حافظه به‌ترتیب طبق رابطه (۱۲) و (۱۳) صورت می‌گیرد.

$$x_j(i) = LB_i + Ar_n(1-r_n)(UB_i - LB_i)\,,$$
$$r_n \epsilon\, [0,1]$$
$$\forall\, j \in (1,2,\dots,SN),\quad \forall\, i \in (1,2,\dots,N) \qquad (۱۲)$$

$$mem_j(i) = LB_i + Ar_n(1 - r_n)(UB_i - LB_i)\,,\quad r_n \epsilon\,[0,1]$$
$$\forall\, j \in (1,2,\dots,SN),\quad \forall\, i \in (1,2,\dots,N) \qquad (۱۳)$$

در مرحله ۳ و ۴ طبق تابع کارایی، شایستگی برای هر زنبور از جمعیت اصلی و جمعیت حافظه محاسبه می‌شود. در مرحله ۵ مشخص می‌شود که به‌روزرسانی برای حافظه در یک بازه تصادفی به‌صورت $rand(5,10)$ انجام می‌گیرد. اگر زمان به‌روزرسانی حافظه در تکرار $cycle$ صورت بگیرد، زمان به‌روزرسانی بعدی برای حافظه در زمان $Rand(5,10) + cycle$ انجام می‌پذیرد. در مرحله ۶ چرخه الگوریتم آغاز می‌شود. مرحله ۷ و ۸ همان توابع مربوط به نحوه به‌روزرسانی را برای زنبورها و جمعیت حافظه بیان می‌کند. در مرحله ۹ ارزیابی مجدد برای محاسبه کارایی زنبورها صورت می‌گیرد که اگر کارایی برای حتی یک زنبور تغییر یافته باشد در آن صورت الگوریتم درمی‌یابد که محیط تغییر کرده است. مرحله ۱۱ در اثر تغییر در محیط باید از اطلاعات ذخیره‌شده در حافظه برای محیط جدید استفاده کرد که این مرحله خود در ۴ فاز مختلف صورت می‌گیرد که این ۴ مرحله به‌صورت زیر تشریح می‌شوند.

فاز ۱: کارایی برای جمعیت اصلی و جمعیت حافظه محاسبه می‌شود.

فاز ۲: بهترین فرد از جمعیت حافظه انتخاب می‌شود که در حقیقت بهترین فرد از این جمعیت فردی است که بیشترین کارایی را دارد.

فاز ۳: بدترین فرد از جمعیت اصلی انتخاب می‌شود (بدترین فرد از جمعیت اصلی فردی است که کمترین کارایی را داراست).

فاز ۴: بهترین فرد از حافظه جایگزین بدترین فرد از جمعیت اصلی می‌شود.

مرحله ۱۲ بیان می‌کند اگر زمان به‌روزرسانی از چرخه الگوریتم بزرگ‌تر شود، به‌روزرسانی برای حافظه فعال می‌شود، در غیر این‌صورت به چرخه یک واحد اضافه می‌شود. در نهایت اگر به پایان شرط توقف چرخه رسیدیم از چرخه خارج شده، به پایان الگوریتم می‌رسیم.

۹- آزمایش‌ها و نتایج

برای انجام آزمایش‌ها بر روی الگوریتم پیشنهادی و مقایسه آن با سایر الگوریتم‌ها در یک محیط پویا از تابع محک قله‌های متحرک[1] [20] برای تست کارایی الگوریتم پیشنهادی استفاده می‌کنیم. در تمام آزمایش‌ها الگوریتم‌ها در شرایط یکسان با هم مقایسه شده‌اند. تعداد اجراهای الگوریتم ۳۰ مرتبه است.

۹-۱- مسئله محک قله‌های متحرک

مسئله محک قله‌های متحرک[2] یک شبیه‌ساز مناسب برای شبیه‌سازی محیط‌های پویا است. این مسئله شامل n قله[3] ، در یک فضای m بعدی[4]، با پارامترهایی با مقادیر واقعی است و ارتفاع[5]، عرض[6] و موقعیت قله‌ها[7] در طول زمان می‌توانند تغییر یابند که این موضوع در حقیقت همان شبیه‌سازی یک مسئله پویا است. تابع محک قله‌های متحرک به‌صورت رابطه (۱۴) فرمول‌بندی می‌شود.

$$F(\vec{x},t) = \max(B(\vec{x}), \max_{i=1\dots m} P(\vec{x}, h_i(t), w_i(t), \vec{p}_i(t))) \qquad (۱۴)$$

که $B(\vec{x})$ مقدار پایه محیط است که مستقل از زمان و P تابعی است که شکل پیک را تعریف می‌کند که هریک از m پیک، پارامترهای متغیر زمانی، ارتفاع (h) و عرض (w) و موقعیت (p) خودشان را دارند. در هر ΔE ارزیابی، ارتفاع، عرض و محل هریک از پیک‌ها تغییر می‌کند. ارتفاع و عرض هریک از قله‌ها با اضافه‌کردن یک متغیر تصادفی گوسی به آن‌ها تغییر می‌کند. پارامتر فرکانس تغییرات[8] بیان می‌کند که در چه زمان‌هایی محیط تغییر پیدا می‌کند یا چه زمان الگوریتم باید به تغییرات محیط پاسخ دهد. تابع محک قله‌های متحرک دارای پارامترهای مختلفی است که با تغییر هریک از این پارامترها در نتایج نیز ممکن است تغییر حاصل شود. شکل (۱۱) نحوه تغییرات قله‌ها را در این مسئله با چند قله نشان می‌دهد.

پارامتر s میزان شدت تغییرات را کنترل می‌کند، ΔE فرکانس تغییرات را تعیین می‌کند. پارامتر λ تعیین می‌کند که چه مقدار تغییر موقعیت یک قله بستگی به حرکت قبلی خود دارد. در صورتی که مقدار $\lambda = 0$ باشد هر حرکت کاملاً تصادفی است، برای مقدار $\lambda = 1$ قله‌ها همیشه در

[1] Moving Peaks Benchmark
[2] Moving Peaks Benchmark
[3] Peaks
[4] Dimension
[5] Height
[6] Width
[7] Position of Peaks
[8] Change Crequency

یک مسیر مشخص حرکت می‌کنند. هر زمان تغییری در محیط رخ می‌دهد، این تغییر بر روی مکان، ارتفاع و عرض یک قله به‌صورت روابط ذکرشده در (۱۵) بیان می‌شود.

$$\begin{cases} h_i(t) &= h_i(t-1) + height_{severity}.\ \sigma \\ w_i(t) &= w_i(t-1) + width_{severity}.\sigma \\ \vec{p}_i(t) &= \vec{p}_i(t-1) + \vec{v}_i(t) \\ \sigma &\in N(0,1) \end{cases} \quad (۱۵)$$

جدول ۱: تنظیمات استاندارد پارامترها برای تابع قله‌های متحرک

پارامتر	مقدار
تعداد قله‌ها	۱۰
فرکانس تغییرات	۵۰۰۰
شدت تغییرات ارتفاع قله‌ها	۷
شدت تغییرات عرض قله‌ها	۱
شکل قله‌ها	مخروطی
میزان حرکت مکان قله ها	۱
ابعاد فضای جست‌وجو (A)	۵
محدوده ارتفاع قله‌ها (H)	[۳۰و۷۰]
محدوده عرض قله‌ها (W)	[۱و۱۲]
ارتفاع استاندارد قله‌ها (I)	۵۰
محدوده فضای جست‌وجو	[۰و۱۰۰]
طول تغییرات (s)	۱

بردار انتقال $\vec{v}_i(t)$ یک بردار تصادفی $\vec{r}$ را با بردار انتقال قبلی $\overrightarrow{v_i}(t-1)$ ترکیب می‌کند. بردار تصادفی $\vec{v}_i(t)$ به‌وسیله تولید اعداد تصادفی در بازه $[0,1]$ برای هر بعد و نرمال‌کردن آن به طول s ایجاد می‌شود. بردار $\vec{v}_i(t)$ را می‌توان وابسته به تغییر قبلی آن ایجاد کرد که در این صورت، تغییر موقعیت قله‌ها هم‌سو با تغییرات قبل آن می‌شود یا به صورت تصادفی آن را ایجاد کرد که موجب می‌شود موقعیت قله‌ها به‌صورت تصادفی تغییر کند و هیچ‌گونه وابستگی به تغییر قبلی نداشته باشد. بردار $\vec{v}_i(t)$ به‌صورت رابطه (۱۶) محاسبه می‌شود.

$$\vec{v}_i(t) = \frac{s}{|\vec{r}+\overrightarrow{v_i}(t-1)|}((1-\lambda)\vec{r} + \lambda\overrightarrow{v_i}(t-1)) \quad (۱۶)$$

تابع قله برای ارتفاع، عرض و موقعیت هر قله به‌صورت فرمول (۱۷) محاسبه می‌شود.

$$P(\vec{x}, h(t), w(t), \vec{p}(t)) = h(t) - w(t).\sqrt{\sum_{j=1\dots n}(x_j - p_j)^2} \quad (۱۷)$$

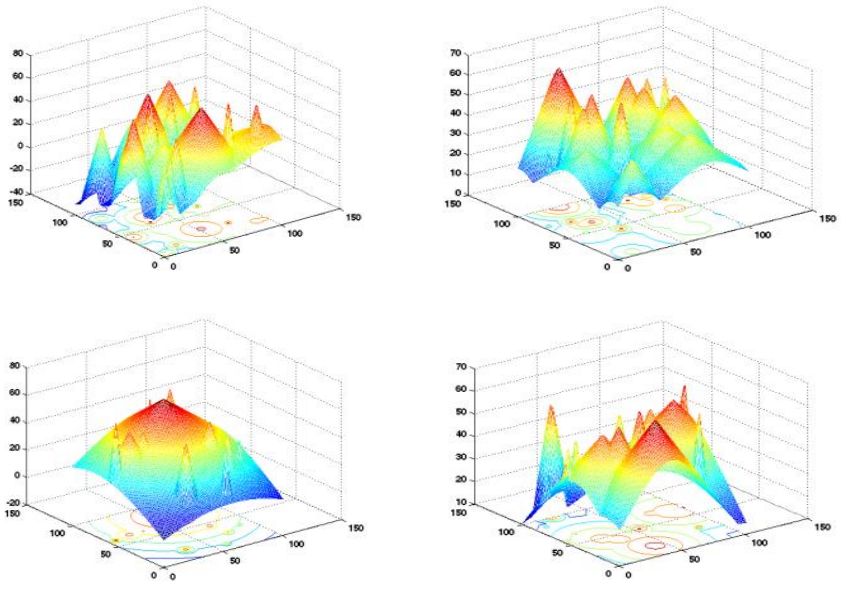
بخش مربوط به جذر، فاصله بین نقطه موجود و موقعیت هر قله را بیان می‌کند. تمامی شبیه‌سازی‌ها در محیط نرم‌افزار متلب صورت گرفته است. در ادامه تنظیمات مربوط به پارامترهای تابع محک قله‌های متحرک را برای ارزیابی الگوریتم پیشنهادی در محیط‌های پویا ذکر می‌کنیم. الگوریتم پیشنهادی با روش‌های *mQSO*، *FMSO*، *AmQSO*، *rPSO* [۲۱]، *CellulrPSO* [۲۲]، *CPSO* [۱]، *ESCA* [۲۳]، *rSPSO* [۲۴]، *CESO* [۲۵]، *SEAm* [۱۵] در جداول مختلف مقایسه می‌شوند. جدول ۱ تنظیمات پیش‌فرض مربوط به تابع محک قله‌های متحرک را نشان می‌دهد. پارامترهای ذکرشده در جدول ۱ مربوط به سناریوی دوم آقای برانک است که در مرجع [۲۰] آمده است. جدول ۲ پارامترهای مربوط به الگوریتم پیشنهادی را نشان می‌دهد و نتایج به‌دست‌آمده از سایر الگوریتم‌ها از مرجع مربوط به آن‌ها استخراج شده است.

جدول ۲: پارامترهای مربوط به الگوریتم پیشنهادی

پارامتر	مقدار
کران پایین	۰
کران بالا	۱۰۰
تعداد زنبورها	۱۰۰
سایز حافظه	۱۰
تعداد منابع غذایی	۵۰
limit	۰,۲
تعداد خوشه‌های جمعیت اصلی	۲
تعداد خوشه‌های جمعیت حافظه	۲

آزمایش‌های مختلف بر روی پارامترهای مختلفی برای الگوریتم انجام گرفته است که نتایج این آزمایش‌ها در ادامه در جداول مختلف موردارزیابی و مقایسه با روش‌های دیگر قرار گرفته است. جدول ۳ مقدار میانگین خطای آفلاین را برای روش پیشنهادی و سایر روش‌ها در فرکانس‌های تغییر ۵۰۰ و ۱۰۰۰ و ۵۰۰۰ و با تعداد قله‌های مختلف نشان می‌دهد. الگوریتم پیشنهادی را در ۳۰ بار اجرا مورد آزمایش قرار داده‌ایم. آزمایش‌ها در محیط نرم‌افزار متلب و سیستم

عامل ویندوز ۷ با پردازنده اینتل CORE i5 انجام گرفته است. نتایج سایر روش‌ها از مرجع مربوط به آن‌ها به دست آمده است (بعضی از روش‌ها برای مقایسه به‌صورت نموداری پیاده‌سازی شده‌اند). نتایج به‌دست‌آمده از جدول ۳ نشان می‌دهد که روش پیشنهادی در فرکانس تغییرات پایین دارای مقدار خطای آفلاین کمی است. این یکی از امتیازهای این الگوریتم است، همچنین در فرکانس تغییرات بالا الگوریتم زمان بیشتری را برای جست‌وجو پیدا کرده، کارایی مناسبی را در ردیابی بهینه تغییریافته از خود نشان می‌دهد. با افزایش تعداد قله‌ها همان‌طور که در جدول ۳ مشاهده می‌شود، با افزایش خطای آفلاین مواجه هستیم که این موضوع را می‌توان با افزایش تعداد جمعیت متناسب با افزایش تعداد قله‌ها حل کرد و خطا را به نسبت در الگوریتم کاهش داد. جدول ۴ میانگین خطای آفلاین را برای روش پیشنهادی و دیگر روش‌ها با طول تغییرات مختلف برای قله‌ها نشان می‌دهد. با افزایش طول تغییرات در قله‌ها، خطای آفلاین نیز افزایش می‌یابد که این موضوع را می‌توان در جدول ۴ در تمام روش‌ها به‌وضوح مشاهده کرد. روش پیشنهادی ما با افزایش طول تغییرات، خطای آفلاین زیادی را تولید نمی‌کند و خطا در آن ناچیز است که این یکی دیگر از امتیازات روش پیشنهادی است. شکل (۱۲) نمودار مربوط به خطای آفلاین را برای روش پیشنهادی در مقایسه با دو روش مبتنی بر حافظه *SEAm* و *MIGA* در فرکانس تغییر، ۱۰۰ و با ۱۰ قله نشان می‌دهد. نمودار رسم‌شده در شکل (۱۲) نشان می‌دهد روش پیشنهادی نسبت به دو روش دیگر، دارای خطای آفلاین کمتری است و کارایی مناسب این روش را در مقایسه با دو روش دیگر نشان می‌دهد. شکل رسم‌شده در (۱۳)، نمودار هم‌گرایی خطای آفلاین الگوریتم پیشنهادی در فرکانس ۱۰۰۰۰ با ۱۰ قله و ۱۰ تغییر در محیط را نشان می‌دهد. شکل (۱۴) نحوه هم‌گراشدن زنبورها را به مراکز قله‌های در حال نوسان در طول اجرای الگوریتم نشان می‌دهد. همان‌طور که در شکل (۱۴) نشان داده شده است هرچه الگوریتم روبه‌جلو حرکت می‌کند، میزان هم‌گرایی زنبورها به قله‌ها افزایش می‌یابد و حافظه صریح بعد از اولین هم‌گرایی زنبورها به قله‌ها به‌سرعت بعد از هر تغییر در محیط الگوریتم را در ردیابی مکان جدید قله‌ها یاری کرده، زنبورها را به‌سرعت به مراکز قله‌ها هم‌گرا می‌کند. نمودار رسم‌شده در شکل (۱۵) متوسط خطای آفلاین را برای روش پیشنهادی در هر تکرار در مقایسه با روش *mQSO* نشان می‌دهد و همان‌گونه که در این نمودار مشاهده می‌شود مقدار خطای آفلاین روش پیشنهادی در مقایسه با روش *mQSO* کمتر بوده، کارایی بیشتری نسبت به این روش داراست. استفاده از حافظه صریح در روش پیشنهادی باعث هم‌گرایی بیشتر زنبورها به مکان‌های بهینه می‌شود و همچنین استفاده از آشوب به‌جای تصادفی‌سازی سرعت هم‌گرایی را در الگوریتم افزایش می‌دهد که به‌وضوح می‌توان آن را در نتایج به‌دست‌آمده از آزمایش‌ها بر روی الگوریتم پیشنهادی مشاهده کرد.



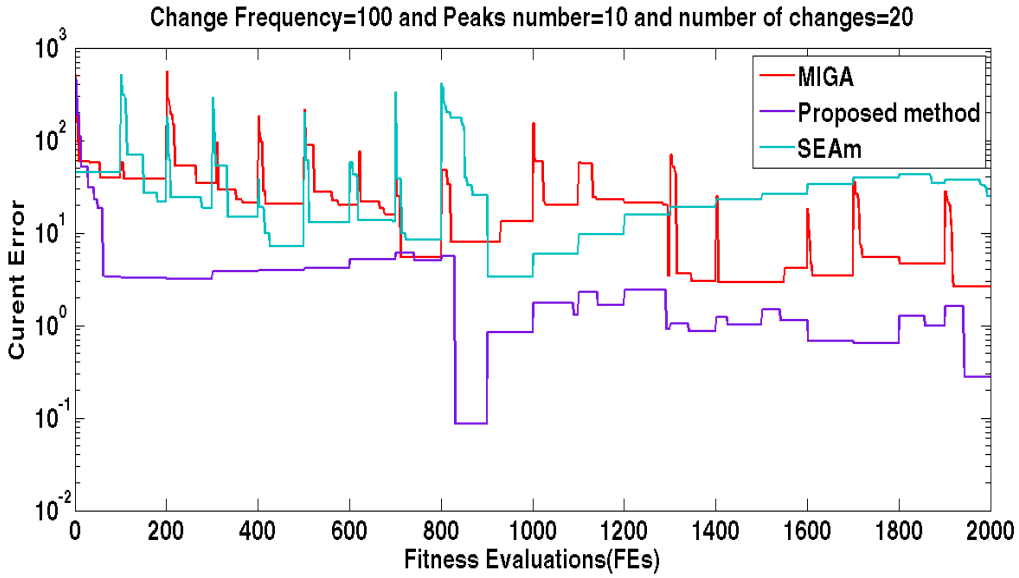
شکل ۱۱: نحوه تغییرات در قله‌ها در تابع محک قله‌های متحرک

جدول ۳: میانگین خطای آفلاین روش پیشنهادی و سایر روش‌ها در فرکانس تغییرات مختلف و با تعداد قله‌های مختلف

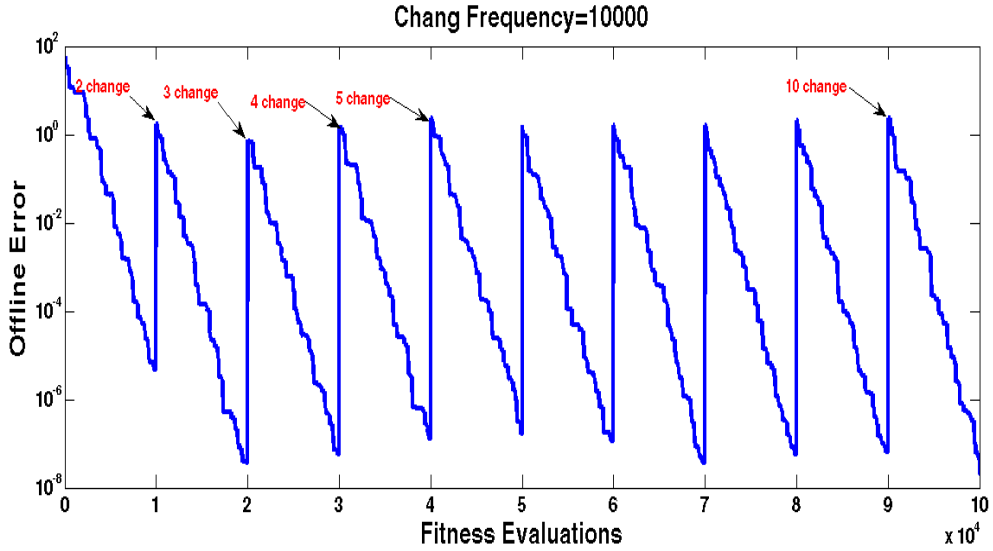
فرکانس تغییرات	تعداد قله‌ها	mQSO	Adaptive mQSO	rPSO	FMSO	Cellular PSO	CPSO	روش پیشنهادی
۵۰۰	۱	۳۳,۶۷	۳,۰۲	۴,۲۷	۲۷,۵۸	۱۳,۹۴	۱۴,۲۵۲۱	**۳,۲۳**
	۵	۱۱,۹۱	۵,۷۷	۱۶,۱۹	۱۹,۴۵	۹,۶۳	۳۶,۴۰۹۴	**۳,۸۶**
	۱۰	۹,۶۲	۵,۳۷	۱۷,۳۴	۱۸,۲۶	۹,۴۲	۲۰,۹۱۲۹	**۴,۵۴**
	۲۰	۸,۷۹	۶,۸۲	۱۷,۰۶	۱۷,۳۴	۸,۸۴	۱۳,۱۱۵۵	**۴,۸۵**
	۳۰	۸,۸۰	۷,۱۰	۱۶,۹۸	۱۶,۳۹	۷,۸۸	۱۰,۸۳۰۷	**۵,۶۱**
	۴۰	۸,۵۵	۷,۰۵	۱۶,۴۶	۱۵,۳۴	۷,۸۳	۱۰,۱۲۳۹	**۵,۷۰**
	۵۰	۸,۵۶	۸,۹۷	۱۵,۷۷	۱۵,۵۴	۸,۶۲	۹,۲۸۷۱۱	**۶,۵۲**
	۱۰۰	۸,۵۴	۷,۳۴	۱۴,۵۵	۱۲,۸۷	۱۱,۳۸	۷,۷۷۲۶۴	**۶,۹۵**
	۲۰۰	۸,۱۹	۷,۴۸	۱۳,۴۰	۱۱,۵۲	۱۱,۳۴	۶,۸۳۳۳۸	**۷,۳۴**
۱۰۰۰	۱	۱۸,۶	۲,۳۳	۱,۹۴	۱۴,۴۲	۶,۷۷	۸,۹۳۵۳۸	**۲.۱۰**
	۵	۶,۵۶	۲,۹۰	۱۳,۷۷	۱۰,۵۹	۵,۳۰	۸,۶۲۲۳۹	**۲.٤٤**
	۱۰	۵,۷۱	۴,۵۶	۱۵,۵۵	۱۰,۴۰	۵,۱۵	۷,۴۸۱۹	**۲.۸۳**
	۲۰	۵,۹۰	۵,۳۶	۱۵,۵۴	۱۰,۳۳	۵,۳۲	۶,۱۰۳۱۴	**۳.۲۷**
	۳۰	۶,۰۲	۵,۲۰	۱۴,۳۸	۱۰,۰۶	۵,۹۳	۵,۴۴۸۵۹	**۳,۵۹**
	۴۰	۶,۲۲	۵,۲۵	۱۴,۱۱	۹,۸۵	۵,۹۵	۵,۵۷۱۲۳	**۳,۶۵**
	۵۰	۵,۹۰	۶,۰۶	۱۳,۷۵	۹,۵۴	۵,۵۵	۵,۱۷۱۸۱	**۳,۷۲**
	۱۰۰	۶,۴۸	۴,۷۷	۱۲,۲۷	۸,۷۷	۶,۲۷	۴,۲۶۲۲۹	**۳,۸۳**
	۲۰۰	۶,۱۸	۵,۷۵	۱۱,۳۲	۸,۰۶	۶,۰۱	۳,۷۴۲۲	**۳,۸۷**
۵۰۰۰	۱	۵,۰۷	۰,۵۱	۰,۵۶	۳,۴۴	۲,۵۵	۰,۱۴۱۲	**۰,۴۵**
	۵	۱,۸۱	۱,۰۱	۱۲,۲۲	۲,۹۴	۱,۶۸	۰,۷۲۴۳	**۰,۵۱**
	۱۰	۱,۷۵	۱,۵۱	۱۲,۹۸	۳,۱۱	۱,۷۸	۱,۰۵۶۵۴	**۰,۶۴**
	۲۰	۲,۷۴	۲,۰۰	۱۲,۷۹	۳,۳۶	۲,۶۰	۱,۵۹۰۹	**۰,۶۷**
	۳۰	۳,۲۷	۲,۱۹	۱۲,۳۵	۳,۲۸	۲,۹۳	۱,۵۸۷۸	**۰,۷۲**
	۴۰	۳,۶۰	۲,۶۵	۱۱,۳۷	۳,۲۶	۳,۱۴	۱,۵۱۰۳	**۰,۸۶**
	۵۰	۳,۶۵	۲,۴۳	۱۱,۳۴	۳,۲۲	۳,۲۶	۱,۵۴۳۶	**۰,۸۹**
	۱۰۰	۳,۹۳	۲,۶۸	۹,۷۳	۳,۰۶	۳,۴۱	۱,۴۱۰۵	**۰۹۵**
	۲۰۰	۳,۸۶	۲,۶۲	۸,۹۰	۲,۸۴	۳,۴۰	۱,۲۴۳۸	**۰,۹۸**

جدول ۴: میانگین خطای آفلاین برای روش پیشنهادی و دیگر روش‌ها با طول تغییرات مختلف

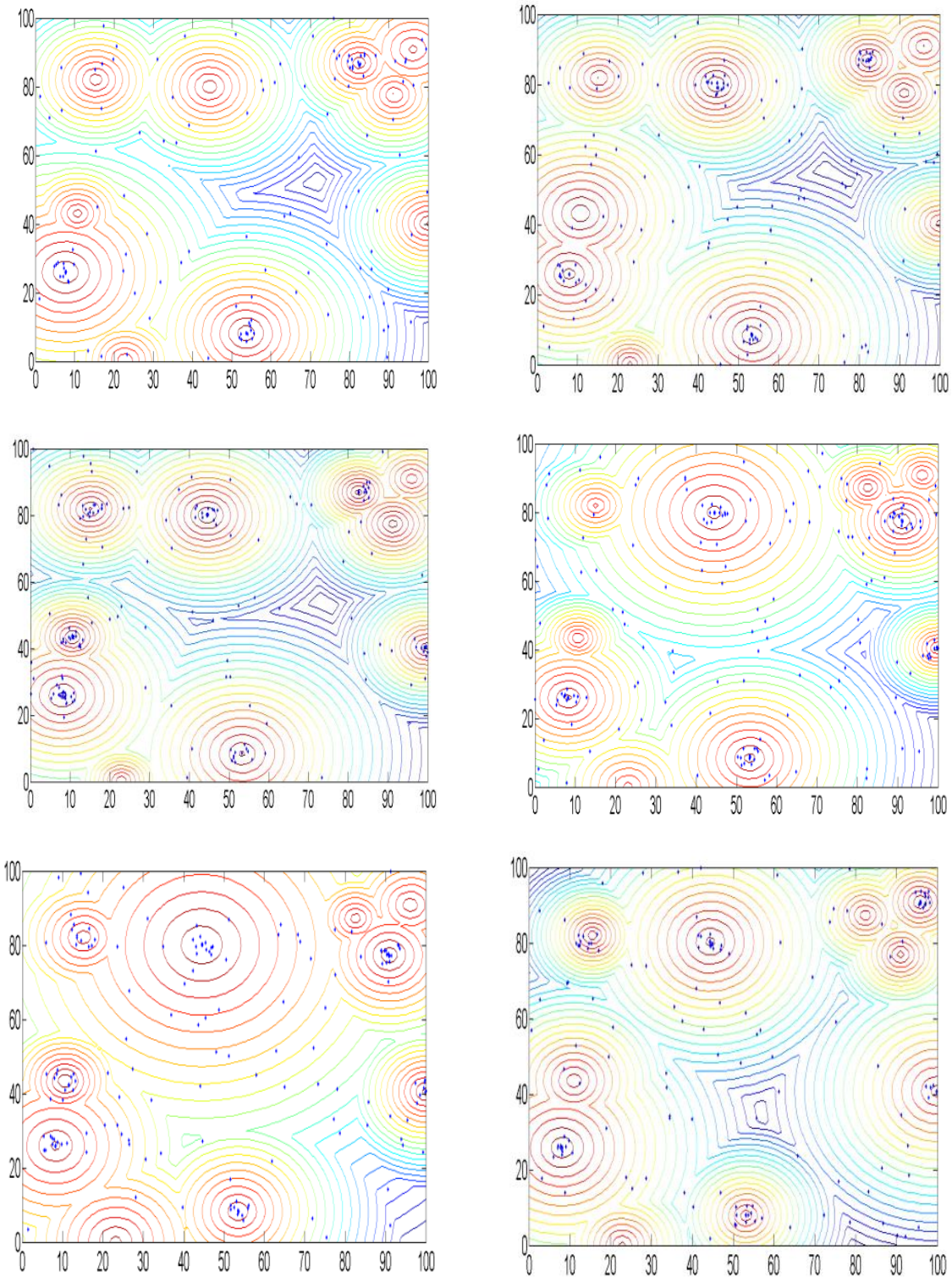
الگوریتم‌ها	طول تغییرات قله‌ها						
	۰	۱	۲	۳	۴	۵	۶
روش پیشنهادی	**۰,۱۵**	**۰,۶۴**	**۰,۸۲**	**۰,۹۵**	**۱,۲۳**	**۱,۴۲**	**۱,۸۷**
CPSO	۰,۴۶۵	۰,۷۱۵	۰,۸۴۳	۰,۹۱۱	۰,۹۹۷	۱,۰۸	۱,۲۳
mQSO	۱,۱۸	۱,۷۵	۲,۴۰	۳,۰۰	۳,۵۹	۴,۲۴	۴,۷۹
rSPSO	۰,۷۴	۱,۵۰	۱,۸۷	۲,۴	۲,۹۰	۳,۲۵	۳,۸۷
ESCA	۱,۷۲	۱,۵۳	۱,۵۷	۱,۶۷	۱,۷۲	۱,۷۸	۱,۷۹
CESO	۰,۵۸	۱,۳۸	۱,۷۸	۲,۰۳	۲,۲۳	۲,۵۲	۲,۷۴



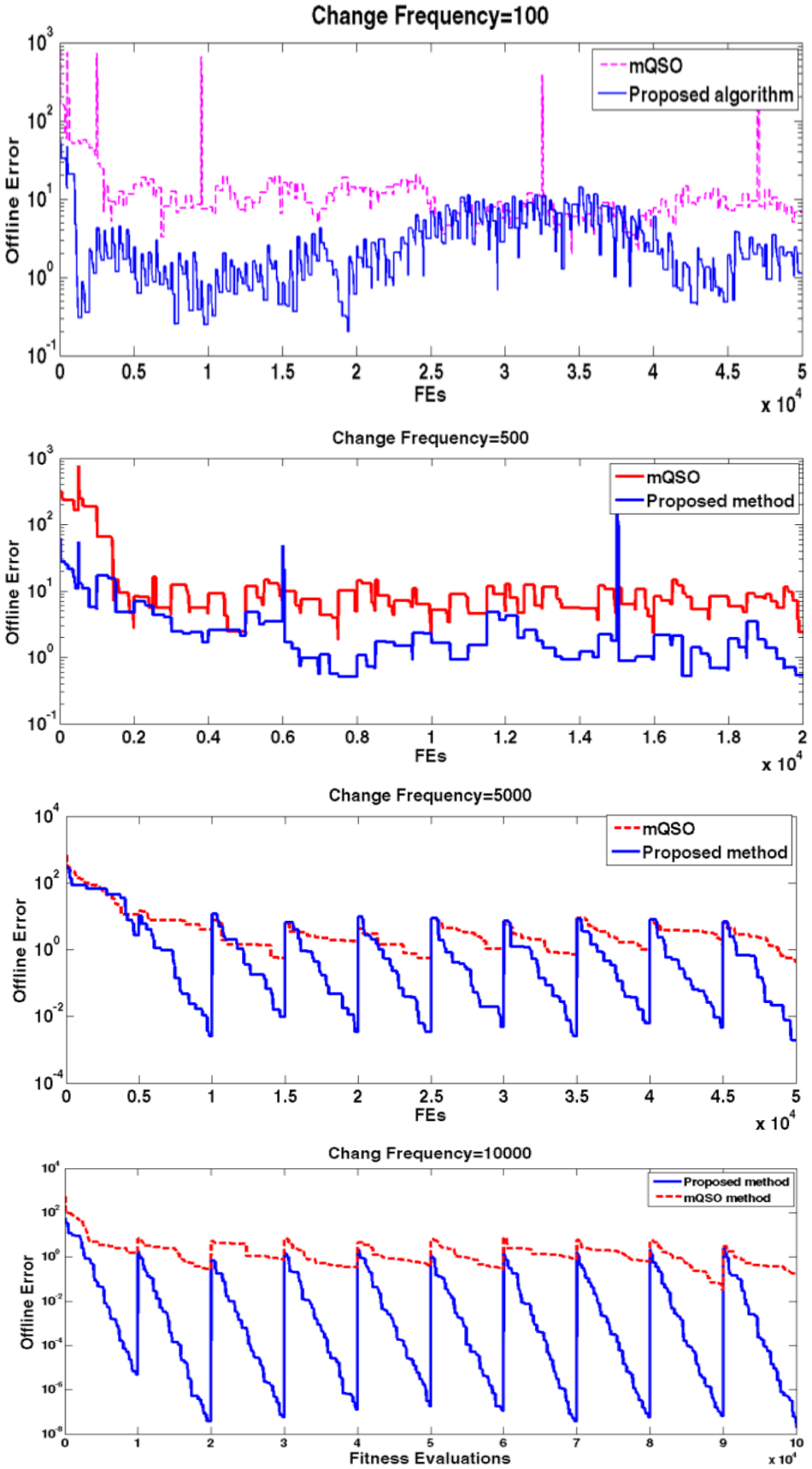
شکل ۱۲: مقایسه روش پیشنهادی با دو روش مبتنی بر حافظه، SEAm و MIGA
از نظر مقدار خطای آفلاین در فرکانس ۱۰۰ و ۱۰ قله



شکل ۱۳: نمودار خطای آفلاین روش پیشنهادی در فرکانس تغییرات ۱۰۰۰۰ و ۱۰ قله



شکل ۱۴: نحوه هم‌گرا شدن زنبورها به قله‌های در حال نوسان در طول اجرای الگوریتم (ترتیب اجرا از چپ به راست در هر سطر)



شکل ۱۵: متوسط خطای آفلاین در هر تکرار برای روش پیشنهادی در فرکانس‌های ۱۰۰، ۵۰۰، ۵۰۰۰ و ۱۰۰۰۰ و با ۱۰ قله در مقایسه با روش mQSO

۱۰- نتیجه‌گیری

الگوریتم‌های بهینه سازی هوشمند در محیط‌های پویا باید به‌گونه‌ای طراحی شوند که بتوانند به‌طور مناسب وکارا بهینه موردنظر را دنبال کنند. در این مقاله ما از ترکیب حافظه با الگوریتم کلونی زنبور مصنوعی برای حفظ راه‌حل‌های خوب استفاده کردیم و از یک استراتژی مناسب برای حفظ تنوع در جمعیت بهره بردیم. همچنین در این الگوریتم برای ایجاد جمعیت اولیه به‌جای تصادفی‌سازی جمعیت اولیه از تابع نگاشت لجستیک که یک نگاشت غیرخطی دینامیکی است و به‌جای حرکت تصادفی، از حرکت آشوب‌گون استفاده کردیم؛ زیرا در جهان واقعی رفتارها به‌جای تصادفی‌بودن بیشتر آشوب‌گونه‌اند و در رفتار حرکتی آشوب‌گون نسبت به رفتار حرکتی تصادفی می‌توان پیش‌بینی مناسب‌تری از آینده داشت و در حرکت آشوب‌گون میزان هم‌گرایی جمعیت به بهینه موردنظر بیشتر است؛ به همین دلیل حرکت آشوب‌گونه نسبت به حرکت تصادفی باعث سرعت هم‌گرایی بیشتر الگوریتم می‌شود. برای غلبه بر چالش‌های موجود در محیط‌های پویا باید یک استراتژی مناسب اتخاذ کرد. روش پیشنهادی می‌تواند برای بیشتر مسائل دینامیکی از جمله زمان‌بندی فعالیت‌ها، بهینه سازی مسیریابی در شبکه‌های سنسوری متحرک، داده کاوی 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. یک الگوریتم بهینه‌سازی باید به‌گونه‌ای طراحی شود که بتواند به‌نحو مطلوبی چالش‌های موجود در مسائل دینامیکی را حل کند و کارایی مناسبی در مواجهه با این‌گونه مسائل از خود نشان دهد. استفاده از تکنیک‌های خوشه‌بندی در روش پیشنهادی می‌تواند گزینه مناسبی برای کارهای آتی باشد.

۶- مراجع

[1] Yang, S., Li C., "A Clustering Particle Swarm Optimizer for Locating and Tracking Multiple Optima in Dynamic Environments", IEEE Transactions on Evolutionary Computation, Vol. 14, No. 6, 2010, pp. 959-974.

[2] Yang. S, "Explicit memory schemes for evolutionary algorithms in dynamic environments", In S. Yang, Y.-S. Ong, and Y. Jin, editors, Evolutionary Computation in Dynamic and Uncertain Environments, volume 51 of Studies in Computational Intelligence, Springer-Verlag, 2007, PP. 3-28.

[3] Kamos, i M., Hashemi, A.B., Meybodi, M.R., "A New Particle Swarm Optimization Algorithm for Dynamic Environments", SEMCCO, 2010, pp. 129-138.

[4] Blackwell, T., Branke, J., "Multi-Swarms, Exclusion, and Anti-Convergence in Dynamic Environments", IEEE Transactions on Evolutionary Computation 10, 2006, 459–472.

[5] Blackwell, T. and Branke, J., "Multi-swarm optimization in dynamic environments", In: G.R. Raidl, editor, Applications of Evolutionary Computing, volume 3005 of Lecture Notes in Computer Science, Springer, Berlin, Germany, 2004, pp.489–500.

[6] Blackwell, T. and Branke, J and Li, X., "Particle swarms for dynamic optimization problems", Swarm Intelligence, Springer Berlin Heidelberg, 2008, PP.193-217.

[7] Du, W., Li, B., "Multi-Strategy Ensemble Particle Swarm Optimization for Dynamic Optimization", Information Sciences: an International Journal, Vol. 178, 2008, pp. 3096–3109.

[8] Li, C. and Yang, S., "A clustering particle swarm optimizer for dynamic optimization", in Proc. Congr. Evol. Comput, 2009, pp. 439–446.

[9] Li, C. and Yang, S., "Fast Multi-Swarm Optimization for Dynamic Optimization Problems", Proc, Int'l Conf. Natural Computation, Vol. 7, No. 3, 2008, pp. 624-628.

[10] Karaboga, D. Basturk, B., "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm", Journal of Global Optimization, 39, 2009, PP. 459–471.

[11] Krasnogor, N. and Smith, j.,"A Tutorial for Competent Memetic Algorithms: Model, Taxonomy, and Design Issues", IEEE Trans. Evolutionary Computation, Vol. 9, No. 5, 2005, pp. 474-488.

[12] Yang, S., "Explicit memory schemes for evolutionary algorithms in dynamic environment", s. In S. Yang, Y.-S. Ong, and Y. Jin, editors, Evolutionary Computation in Dynamic and Uncertain Environments, volume 51 of Studies in Computational Intelligence, Springer-Verlag, 2007, PP. 3-28.

[13] Ryan, C., "Dyploidy without dominance", In J. T. Alander, editor, Proceedings of the Nordic Workshop on Genetic Algorithms, 1997, P. 6370.

[14] Yang, S., "Genetic algorithms with elitism-based immigrants for changing optimization problems", In Applications of Evolutionary Computing, Lecture Notes in Computer Science 4448, 2007, PP. 627–636.

[15] Ramsey, C. Grefenstette, J., Case-based initialization of genetic algorithms", In S. Forrest, editor, Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, 1993, PP. 84-91.

[16] Trojanowski, K. and Michalewicz, Z., "Searching for optima in non-stationary environments", In Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999), IEEE Press, 1999, PP. 1843-1850.

[17] Wang, H. Yang, S. Ip D.WH., "A memetic particle swarm optimisation algorithm for dynamic multi-modal optimization problems", Int J Syst Sci 43(7), 2012, PP.1268–1283.

[18] Branke, J., "Memory enhanced evolutionary algorithms for changing optimization problems", In Congress on Evolutionary Computation, 1999, PP. 1875–1882.

[19] Morrison, R and DeJong, K., "A test problem generator for non-stationary environments", In Congress on Evolutionary Computation, 1999, PP. 2047–2053.

[20] Branke, J. The Moving Peaks Benchmark Website, http://www.aifb.unikarlsruhe. De/jbr/movpeaks.

[21] Parrott, D and Li, X., "Locating and Tracking MultipleDynamic Optima by A Particle Swarm Model Using peciation", in IEEE Transaction on Evolutionary Computation, vol. 10, No. 4, 2006, pp. 440-458.

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

[23] Lung, R. I and Dumitrescu, D., "Evolutionary swarm cooperative optimization in dynamic environments", Natural Comput., Vol. 9, No. 1, 2010, pp. 83–94.

[24] Bird, S and Li, X., "Using regression to improve local convergence", in Proc. Congr. Evol. Comput, 2007, pp. 592–599.

[25] Lung, R. I and Dumitrescu, D., "A collaborative model for tracking optima in dynamic environments", in Proc. Congr. Evol. Comput, 2007, pp. 564–567.

[26] Li, C. and Yang, S., "A general framework of multipopulation methods with clustering in undetectable dynamic environments", Evolutionary Computation, IEEE Transactions on16(4), 2012, PP.556–577.

[27] Nasiri, B. and Meybodi, M., "Speciation based firefly algorithm for optimization in dynamic environments", International Journal of Artificial Intelligence, 8(S12), 2012, PP.118–132.

[28] Noroozi, V., Hashemi, A. and Meybodi, M., "Cellularde: a cellular based differential evolution for dynamic optimization problems", Adaptive and Natural Computing Algorithms, 2012, pp. 340–349.