الگوریتم بهینه سازی جستجوی گروهی با مکانیزم روش یادگیری از روی تضاد برای بهینه سازی در محیط های پویا

زاهده ناظمی[1]، محمدرضا میبدی[2]

[1] دانشکده برق، رایانه و فناوری اطلاعات، دانشگاه آزاد اسلامی، قزوین

zahedeh1218@gmail.com

[2]دانشکده مهندسی کامپیوتر، فناوری اطلاعات، دانشگاه صنعتی امیرکبیر، تهران

mmeybodi@aut.ac.ir

چکیده:

بسیاری از مسائل دنیای واقعی ماهیتی پویا دارند، به این مفهوم که موقعیت و مقدار بهینه سراسری آنها در طول زمان تغییر می کنند. یکی از این مسائل معروف در بهینه سازی محیط های پویا، مسئله تابع محک قله های متحرک می باشد، که رفتاری شبیه به مسائل پویا در دنیای واقعی را دارد. در واقع یک چالش اصلی در مسائل بهینه‌سازی دینامیک بحث تنوع پذیری[1] جمعیت است، بسته به تغییرات کم یا شدید محیط، تنوع پذیری مناسب اعضای جمعیت یک الگوریتم تکاملی می تواند راه گشای کشف نقطه بهینه سراسری مسئله تا قبل از تغییر محیط باشد، در این مقاله سعی شده است با تکیه بر نقاط قوت الگوریتم جستجوی گروهی[2] وارائه نسخه دینامیک این الگوریتم نسخه بهبود یافته ی ترکیبی این الگوریتم و همچنین با استفاده از رویکرد یادگیری از روی تضاد[3] ایده‌ی جدید جهت ایجاد تنوع پذیری اعضای جمعیت در هنگام تغییر محیط ارائه دهیم. و کارآیی الگوریتم جستجوی گروهی را بهبود ببخشیم و از آن برای حل مسائل بهینه سازی حوزه دینامیک استفاده کنیم.که بتواند تعدادی از مسائل بهینه سازی در محیط های دینامیک با کارایی بالاتر را حل نماید.

کلمات کلیدی

الگوریتم های چند جمعیتی، الگوریتم جستجوی گروهی ، یادگیری از روی تضاد، تابع محک قله های متحرک[4] ، محیط های پویا

۱. مقدمه

بسیاری از مسائل بهینه سازی در دنیای واقعی پویا می باشند در این مسائل بهینه سراسری و بهینه های محلی در طول زمان تغییر می کنند مثال های عملی از این نوع مسائل عبارتند از شناسایی تصاویر متحرک در پس زمینه های در حال تغییر، ساختن مدلی برای صورت های مالی در شرایط متغییر بازار ، داده کاوی در شرایطی که پایگاه داده ها به طور پیوسته به روز رسانی می شود، مسائل زمانبندی با منابع پویا، مسیر یابی وسایل نقلیه و ...این مسائل نیازمند الگوریتم های بهینه سازی هستند که علاوه بر یافتن بهینه بتوانند بهینه های متغییر را دنبال نمایند.

---

[1] Diversity

[2] Group Search Optimizer(GSO)

[3] Opposition-Based Learning

[4] .Moving peaks benchmark

در دهه اخیر استفاده از روشهای تکاملی و هوش جمعی به منظور بهینه سازی در محیط های پویا توسط بسیاری از محققین در حوزه یادگیری ماشین و هوش مصنوعی مورد پیشنهاد قرار گرفته است. دلیل اصلی این توجه نیز نهفته در طبیعت ذاتی و سیر تکامل و تطبیق پذیری این روشها در طول فرایند اجرای شان می باشد که این پدیده برای مسائل بهینه سازی پویا که محیط به طور مداوم در حال تغییر است بسیار مناسب می باشد.

بسیاری از مسائل بهینه سازی کاربردی از این حیث که پاسخ بهینه با گذشت زمان تغییر می کند، پویا هستند. بنابراین در این گونه مسائل یک الگوریتم بهینه‌سازی نه تنها باید مقدار بهینه را بیابد بلکه باید قادر باشد تا با تغییرات محیط، نقطه ی بهینه را در فضای جستجو با حداکثر دقت ممکن دنبال کند[۱]. واضح است که اگر تغییرات به وجود آمده در نمونه‌ی مسئله بنیادین باشند، بهترین رویکرد حل مجدد مسئله‌ی بهینه‌سازی جدید از ابتدا می‌باشد. اما در اغلب کاربردهای عملی تغییرات محیطی تدریجی می‌باشند. در چنین حالتی، می توان با استفاده از اطلاعات جمع‌آوری شده درباره‌ی محیط مسئله طی فرآیند بهینه‌سازی، سرعت این فرآیند را پس از رخداد هر تغییر در محیط افزایش داد[۱].

۱-۱- چالش‌های موجود در محیط‌های پویا

دو چالش اصلی پیش روی الگوریتم‌های تکاملی در محیط‌های پویا وجود دارد که به منظور دست یافتن به نتایج بهینه باید بر آنها غلبه کرد. این چالش‌ها عبارتند از:

حافظه‌ی منسوخ شده[۵]:

هنگامی که تغییری در محیط رخ می‌دهد، ممکن است پاسخ‌های مطلوبی که قبلاً یافت شده‌اند دیگر خوب نباشند و اگر حافظه‌ی ذرات بروزرسانی نشود باعث گمراه شدن جمعیت الگوریتم تکاملی و حرکت افراد به سمت پاسخ‌های نامطلوب می‌شود.

از بین رفتن تنوع[۶]:

با هر بار ایجاد تغییر در محیط، مدت زمانی که صرف می‌شود تا جمعیت الگوریتم تکاملی که به صورت جزئی همگرا شده‌اند مجدداً پراکنده شوند، قله‌ی جابجا شده را بیابند و دوباره به آن همگرا شوند باعث افت شدید کارایی الگوریتم تکاملی می‌شود.

الگوریتم‌های تکاملی از دو مشکل حافظه‌ی منسوخ شده و از بین رفتن تنوع در محیط‌های پویا رنج می‌برند.از آنجا که بیشتر رویکردهایی که برای مقابله با این دو چالش مطرح شده‌اند به صورت صریح نسبت به تغییرات به وجود آمده در چشم انداز مسئله واکنش نشان می‌دهند، ابتدا راه‌های شناسایی تغییرات محیط بررسی می‌شود. هر مسئله‌ای که وابسته به زمان باشد را می توان به عنوان یک مسئله‌ی پویا در نظر گرفت، با این وجود از دیدگاه الگوریتم‌های تکاملی تمامی این گونه مسائل به یک اندازه جالب توجه نیستند. در این زمینه مسائلی بیشتر مورد توجه قرار می گیرند که تابع شایستگی آنها، قبل و بعد از وقوع تغییرات در محیط برخی از شباهت‌های قابل استخراج را نمایش دهد. چراکه اگر مسئله به طور کامل و بدون ارجاعی به پیشینه‌ی آن تغییر یابد، باید آن را به عنوان دنباله‌ای از مسائل مستقل در نظر گرفت که هر کدام باید از ابتدا حل شوند.

اگر تغییر در مسائل بهینه سازی پویا گسترده باشد و فضای مسئله پس از تغییر دارای شباهت کمی با قبل باشد، شروع مجدد می تواند تنها گزینه مناسب باشد و هر گونه استفاده مجدد از اطلاعات جمع آوری شده بر مبنای مسئله قبلی گمراه کننده خواهد بود. در اکثر مسائل دنیای واقعی، امید می رود که تغییرات نسبتا هموار باشد یعنی بین محیط مسئله پیش از تغییر و پس از تغییر ارتباطی وجود داشته باشد تا بتوان از اطلاعات قبلی برای تسریع فرآیند بهینه سازی استفاده نمود.

سوال اساسی که در اینجا مطرح می شود این است که چه اطلاعاتی باید نگهداری شود و این اطلاعات چگونه باید برای تسریع عمل جستجو پس از تغییر محیط استفاده شوند. مسئله دیگر این است که در بیشتر الگوریتم های بهینه سازی، پس از اینکه الگوریتم به یک نقطه همگرا شد، تنوع خود را از دست می دهد. که این امر باعث کاهش انطباق پذیری و سازگاری الگوریتم در مقابل تغییر در محیط خواهد بود، بنابراین در کنار انتقال اطلاعات بین عامل های الگوریتم بهینه سازی بین دو تغییر در محیط، می بایست به دنبال راهکارهایی برای افزایش تنوع و سازگاری الگوریتم پس از تغییرات در محیط نیز بود.

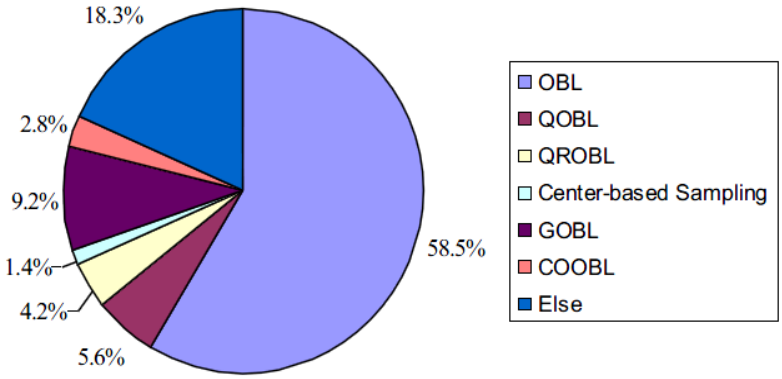
---

[۵] .Outdated memory

[۶] .Diversity loss

یادگیری از روی تضاد[۷][۲]:

در بیشتر الگوریتم‌های تکاملی از بحث مهمی در ایجاد تنوع بهتر اعضای جمعیت استفاده می‌شود،که مرتبط با یادگیری از روی تضاد[۶،۳] است، تاکنون الگوریتم های مختلفی با به کار بردن مبحث یادگیری از روی تضاد توانسته‌اند کارآیی بهتری نسبت به نسخه اصلی آن الگوریتم گزارش دهند، از این جمله می توان به الگوریتم‌های گرگ‌های خاکستری[۷]، الگوریتم تکامل تفاضلی[۸]، الگوریتم بهینه‌سازی اجتماع ذرات [۱۰،۹]، الگوریتم بهینه‌سازی جوجه[۱۱]، الگوریتم بهینه‌سازی ژنتیک[۱۲] ، الگوریتم بهینه‌سازی رقابت استعماری[۱۳]، الگوریتم بهینه‌سازی اجتماع گربه‌ها[۱۴]، الگوریتم بهینه‌سازی کرم شب-تاب[۱۶،۱۵]، الگوریتم جستجوی گرانشی[۱۸،۱۷] و ... اشاره کرد. روش‌های مختلف یادگیری از روی معکوس(تضاد) در شکل ۱ نشان داده شده است.



شکل(۱): روش‌های مختلف یادگیری از روی معکوس(تضاد) [۲]

عمدتا کاربردهای که تاکنون از مباحث یادگیری از روی تضاد بیان شده است بیشتر در رابطه با تولید جمعیت اولیه، به دلیل بالا بردن تنوع پذیری اعضای جمعیت است، در محیط‌های پویا به دلیل اینکه محیط دائما در حال تغییر است، نیاز به سازکارهای مناسبی جهت حفظ تنوع‌پذیری جمعیت بحث مهمی است، از این رو در این مقاله قصد داریم با به کار بردن این مفاهیم و ارائه ایده‌ی جدید جهت ایجاد تنوع‌پذیری اعضای جمعیت در هنگام تغییر محیط کارآیی الگوریتم جستجوی گروهی را بهبود ببخشیم و از آن برای حل مسائل بهینه‌سازی حوزه دینامیک استفاده کنیم.

در این مقاله راهکار مناسبی جهت بدست آوردن نسخه دینامیک الگوریتم جستجوی گروهی را ارائه دادیم. با توجه به اینکه الگوریتم بهینه‌سازی جستجوی گروهی[۸] [۱۹] یک الگوریتم جدید به شمار می‌رود تاکنون تغییراتی انگشت شماری روی این الگوریتم انجام شده است. الگوریتم بهینه‌سازی جستجوی گروهی مبتنی بر رفتار جستجوی حیوانات و تئوری زندگی جمعی است. این الگوریتم از نظر ادراکی ساده است و در اکثر موارد، یک پارامتر تنظیم شده‌ی مناسب، درصد رنجرها، تنوع بالایی ازمسائل بهینه سازی را می تواند حل کند و کارایی GSO نسبت به دیگر پارامترها مثل ماکسیمم زاویه ی تعقیب حساس نیست. این ویژگی حل مسائل بهینه سازی را برای برنامه‌های کاربردی جهان واقعی جذاب می کند .

در این مقاله یک الگوریتم بهینه سازی مبتنی بر الگوریتم GSO پیشنهاد می گردد، که برای عمل در محیط های پویا طراحی شده است و تمام الزامات محیط-های پویا را برآورده می کند. الگوریتم پیشنهادی از راهکارهای نوینی برای رسیدن به نتایج بهتر بهره برده است. الگوریتم پیشنهادی بر روی سناریو۲ که در

---

[۷] Opposition-Based Learning

[۸] Group Search Optimizer(GSO)

جدول ۱ آمده است و بر روی بنچ مارک قله های متحرک (MPB) [۹] که یکی از معرفترین بنچ مارک ها در محیط های پویا است به کار رفته و کارایی آن با هفت الگوریتم دیگر به نام های [۲۰]MQSO،[۲۱]AMQSO،[۲۲]MPSO،[۲۳]HMPSO،[۲۴]APSO،[۲۵]CPSO،[۲۶] CPSOR در فرکانس های مختلف با تعداد قله های مختلف مقایسه شده است معیار مقایسه، خطای برون خطی[۲۷] است که یکی از معیار های اصلی مقایسه الگوریتم های طراحی شده برای محیط های پویا می باشد. نتایج آزمایشات نشان می دهد که الگوریتم پیشنهادی از کارایی قابل قبولی برخوردار است.

ادامه این مقاله بدین ترتیب سازماندهی شده است. در بخش دوم به تشریح الگوریتم پیشنهادی می پردازیم. در بخش سوم نتایج آزمایشات مورد بررسی قرار می گیرند و در بخش آخر به بیان نتیجه گیری می پردازیم.

۲ – رهیافت پیشنهادی

۲-۱- الگوریتم بهینه سازی جستجوی گروهی به منظور بهینه سازی در محیط های پویا (Dyn-GSO-OBL) :

در این زیربخش از مقاله الگوریتم پیشنهادی (Dyn-GSO-OBL) مبتنی بر بهینه سازی جستجوی گروهی برای بهینه سازی در محیط های پویا ارائه می گردد .این الگوریتم از مکانیسم ها و راهکارهای نوینی برای بهبود کارایی الگوریتم بهره می برد. در ادامه این مکانیسم ها با جزئیات کافی مورد بررسی قرار گرفته اند.

۲-۲ - مکانیسم های استفاده شده در الگوریتم پیشنهادی (Dyn-GSO-OBL)

۲-۲-۱- مکانیسم اول : استفاده از روشهای چندجمعیتی

در الگوریتم Dyn-GSO-OBL به جای استفاده از روش های تصادفی سازی در جمعیت اولیه که تاکنون در تمامی مقالات به از آن استفاده شده است. ما با استفاده از رویکرد تصادفی سازی جمعیت اولیه با مکانیزم یادگیری از روی تضاد این کار را انجام می دهیم. تعداد از پیش تعیین شده ای دسته (جمعیت) وجود دارد که هر دسته وظیفه یافتن وتعقیب یک قله را بر عهده دارد .در محیط های پویا بهینه های محلی ممکن است پس از تغییر در محیط تبدیل به بهینه سراسری شوند بنابراین لازم است تا حد ممکن تمام آنها تحت پوشش دسته ها قرار گیرند .در این الگوریتم تعداد دسته ها در ابتدای الگوریتم تعیین می گردد و تا انتهای الگوریتم این تعداد ثابت باقی می ماند .عدم وجود چنین مکانیسمی در الگوریتم های هوش جمعی(مکانیسمی به منظور افزایش تنوع در محیط) سبب می شود با هر بار تغییر در محیط در صورت تغییر کردن بهینه سراسری، احتمال پیدا کردن بهینه سراسری در محیط بعدی کاهش چشمگیری داشته باشد و کارایی الگوریتم به شدت کاهش یابد.

در الگوریتم Dyn-GSO-OBL برخلاف[۲۸] که هر دسته از دو نوع ذره یعنی ذرات خنثی و ذرات کوانتوم تشکیل شده است، در این الگوریتم هر دسته از ذرات خنثی و ذرات با مکانیزم جدید OBL (که شرح داده خواهد شد) شکیل می شوند. ذرات خنثی در هر تکرار از اجرای الگوریتم بر اساس مکانیزم GSO که بر اساس تولید کننده، تکاپوکننده و محدود کننده موقعیت خود را بهنگام می کنند. ذرات OBL جستجوی محلی خود را با روشی که در [۲۸] ارائه شده بود انجام می دهند.

۲-۲-۲- مکانیسم دوم :

جلوگیری از همگرایی چند دسته به یک قله با استفاده از مکانیزم انحصار [۱۰]

یکی از مشکلات استفاده از روشهای چند جمعیتی به منظور بهینه سازی در محیط های پویا همگرا شدن چند دسته به طور همزمان به یک قله در صورت عدم وجود مکانیسمی در جهت دفع دسته هایی که به یکدیگر بسیار نزدیک گشته اند می باشد .شعاع انحصار اولین بار در [۲۸] روشی به منظور محدود نمودن فضای جستجوی هر دسته و جلوگیری از تداخل دسته ها در هنگام فرایند جستجو و دنبال نمودن یک قله توسط چند دسته پیشنهاد شد. عدم وجود چنین مکانیسمی در الگوریتم های چند جمعیتی مشکلات زیر را برای الگوریتم به منظور بهینه سازی در محیط های پویا ایجاد می کند.

---

[۹].Moving peaks benchmark

[۱۰] Exclusion Mechanism

۱. هدر رفتن تعداد بسیار زیادی ارزیابی تابع شایستگی که می توانست برای یافتن قله های دیگر در فضای جستجو مورد استفاده قرار گیرد(با توجه به محدود بودن فاصله زمانی بین دو تغییر در محیط، تعداد دفعاتی که تابع شایستگی در هر محیط می تواند مورد ارزیابی قرار گیرد نیز محدود می باشد.)
۲. کاهش شدید کارایی الگوریتم
۳. همگرا شدن الگوریتم به تعداد قله های کمتر و در نتیجه احتمال کمتر در یافتن قله سراسری توسط الگوریتم در محیط بعدی

۲-۲-۳- مکانیسم سوم :

حفظ قابلیت جستجوی سراسری الگوریتم با استفاده از عملگر ضد همگرایی [۱۱]

یکی دیگر از مشکلات استفاده از روشهای چند جمعیتی به منظور بهینه سازی در محیط های پویا زمانی است که تعداد قله ها از تعداد دسته ها بیشتر باشد . در این حالت زمانیکه کلیه دسته ها همگرا شده باشند(قله تحت نظر خود را پیدا کرده باشند) حتی با اتفاق افتادن تغییر در محیط، دسته ها تنها سعی خواهند نمود که در دنباله روی(با جستجوی محلی) و یافتن قله تحت نظر خود موفق باشند لذا در صورتیکه بهینه سراسری تغییر کرده و در بین بهینه های یافت شده نباشد، الگوریتم نیز تلاشی برای پیدا کردن قله های جدید نخواهد کرد و این موضوع سبب کاهش شدید در کارایی الگوریتم می گردد. با اعمال این عملگر، دسته ای که بهترین فرد موجود در آن نسبت به بهترین فرد سایر دسته ها ضعیف تر باشد انتخاب و با هدف پیدا کردن قله های جدید در فضای مسئله بطور مجدد مقداردهی اولیه می شود .بدین منظور از شعاع همگرایی [۱۲] که اولین بار در [۲۸] به منظور شناسایی دسته های همگرا شده پیشنهاد شد، استفاده می گردد.

چنانچه فاصله اقلیدسی بین دورترین نقاط یک جمعیت از یکدیگر از شعاع همگرایی (convr) کمتر باشد، آن دسته همگرا شده است .آزمایشات انجام شده در [۲۸] نشان داد که تاثیر عدم وجود این عملگر در محیط های با تعداد قله های بالاتر بسیار بیشتر می باشد.

۲-۲-٤- مکانیسم چهارم:

شناسایی تغییر در محیط و افزایش تنوع در محیط پس از آن

یکی از مهمترین مکانیسم هایی که می بایست هر الگوریتم بهینه سازی در محیط های پویا دارا باشد، شناسایی زمانی است که تغییر در محیط اتفاق می افتد. در بعضی از کارهای پیشین انجام شده در بهینه سازی در محیط های پویا، زمان اتفاق افتادن تغییر در محیط به عنوان یک اطلاعات اضافی در هنگام شروع مسئله به الگوریتم اعلام شده است و الگوریتم با علم به این اطلاع اضافی، تنوع در محیط را درست بعد از زمان تغییر در محیط افزایش داده و کارایی الگوریتم بالا می رود . اما در اکثر کاربردهای واقعی زمان اتفاق افتادن تغییر در محیط ناشناخته می باشد و همچنین ممکن است این زمان ثابت نباشد لذا در این پایان نامه نیز زمان اتفاق افتادن تغییر در محیط برای الگوریتم ندانسته در نظر گرفته شده و الگوریتم سعی در شناسایی تغییر در محیط را خواهد داشت .برای شناسایی تغییر در محیط در الگوریتم Dyn-GSO-OBL ، یک نقطه در ابتدای اجرای الگوریتم در فضای مسئله در نظر گرفته می شود و در پایان هر تکرار مقدار آن به طور مجدد توسط تابع شایستگی سنجیده می شود .در صورتی که تغییری در محیط رخ داده باشد مقدار شایستگی این نقطه تغییر کرده است .بر خلاف [۲۸] در ذره کوانتم که حول مقدار Gbest با استفاده از توزیع یکنواخت به تولید مقادیر برای حفظ تنوع می پرداخت در رویکرد پیشنهادی حول بهترین نقطه بهینه سراسری Gbest که مهمترین داده ما است که از مرحله قبل بدست آمده است می باشد. شروع به تولید مقادیر متنوع بر اساس مکانیزم روش یادگیری از روی معکوس(تضاد) می کنیم. و مکانیزم به روز رسانی ذرات که در الگوریتم GSO بر اساس تولید کننده، تکاپوکننده و محدود کننده ذرات مجدداً مورد ارزیابی قرار می گیرد تا مقادیر حافظه معتبر شوند. داشتن مکانیزم تنوع خوب پس از کشف تغییرات بسیار مهم می باشد. و این که ما بتوانیم حول داده قبلی Gbest شروع به تولید مقادیر اولیه کنیم بسیار ارزشمند می باشد. استفاده از روش یادگیری از روی تضاد با این محوریت فکری بیان شده است که، تضاد مقدار Gbest شاید بتواند ما را به مقادیر که حول و یا حتی خود Gbest در لحظه کنونی از زمان برساند. و رسیدن به این چنین مقداری موفقیت بزرگی خواهد بود. و ما می توانیم در زمان کمتری به بهینه سراسری دست پیدا کنیم.

این مکانیسم تاثیر بسیار بالاتری را در کاهش کارایی الگوریتم نسبت به سایرین ایفا می کند .همچنین نحوه پیاده سازی آن نیز می تواند بر روی تعداد دفعات ارزیابی تابع شایستگی و درنتیجه کارایی الگوریتم تاثیر بسزایی داشته باشد.

---

[۱۱] . Anti-Convergence Operator

[۱۲] . Convergence Radius

۲-۲-۵- مکانیسم پنجم :

استفاده از متد یادگیری از روی تضاد (OBL) به منظور حل مشکل ایجاد تنوع در افراد پس از وقوع تغییرات در محیط

یکی از مکانیسم های دیگری برای بهینه سازی در محیط های پویا و یافتن بهینه سراسری بسیار لازم می باشد حفظ تنوع بعد از ایجاد تغییرات در الگوریتم است. زمانی که تغییرات در مسائل پویا به صورت پریودیک یا چرخشی اتفاق افتند، ممکن است بهینه های محلی و سراسری به موقعیت های قبلی خود در محیط و یا به نزدیکی آنها برگردند .از اینرو استفاده مجدد از راه حلهای قبلی به منظور کاهش زمان محاسباتی و هدایت مناسب فرایند جستجو می تواند بسیار مفید باشد. برای استفاده مجدد از راه حل های قبلی بسیاری از محققین بهره گیری از نوعی حافظه در الگوریتم تکاملی را به منظور ردیابی تغییرات پریودیک مفید می دانند.اما با توجه به معایب استفاده از حافظه که کمترین آن نیاز به داشتن حافظه اضافی می باشد.این الگوریتم با نگه داری بهترین نقطه (بهینه سراسری) که تنها اطلاعات مفید از محیط قبل می باشد.و با ایجاد نقاط متضاد حول این نقطه به تعداد زیاد سعی در بهبود راه حل ها دارد. و همچنین باعث ایجاد تنوع پس از کشف تغییرات در محیط می شود. و در هر دسته تاثیر بسزایی را در کاهش خطای بهینه سازی و افزایش کارایی الگوریتم باعث می شود . لذا در الگوریتم Dyn-GSO-OBL پس از بروزرسانی ذرات خنثی، بهترین فرد موجود در هر دسته بعنوان یک ذره متضاد به جای استفاده از ذره کوانتمی در نظر گرفته می شود. و همچنین مقداردهی اولیه به منظور بهبود عملکرد الگوریتم با این روش صورت می گیرد.

.۲-۳- ساختار الگوریتم پیشنهادی (Dyn-GSO-OBL)

شبه کد مربوط به الگوریتم Dyn-GSO-OBL در شکل ۲ و فلوچارت الگوریتم پیشنهادی در شکل ۳ آمده است. در این الگوریتم از کلیه مکانیسم های مطرح شده در بالا بهره لازم برده شده است .استفاده از ایدهٔ ذرهٔ متضاد به صورت ذکر شده موجب افزایش سرعت همگرایی بر اساس تعداد ارزیابی شایستگی می شود .همچنین می تواند مشکل کاهش تنوع در دسته ها را پس از تغییر محیط که در[۱] پیشنهاد شده را با روش بهبود یافته ای، به خوبی حل کند.

$$\mathbf{X_{Quantum,j}}=\mathbf{Gbest_{bestSwarm,j}} +(R_{cloud} \times R_j) \qquad (۲)$$

در الگوریتم پیشنهادی برخلاف [۲۸] که هر دسته از دو نوع ذره یعنی ذرات معمولی و ذره کوانتم تشکیل شده است. در این رویکرد هر دسته از دو نوع ذره یعنی ذره معمولی و ذره یادگیری از روی تضاد(OBL) تشکیل می شوند. و فرمول ذره کوانتم که $\mathbf{Gbest_{bestSwarm,j}}$ بهترین ذره ای که از جمعیت قبل بدست آمده است. R_{cloud} یک مقداری ما بین صفر و یک است. و R_j ابعاد فضای جستجو می باشد. ما این فرمول را با استفاده از روش یادگیری بر روی تضاد تغییر دادیم.

یادگیری تضاد معمولی:

$$x_{new} = (Low\_x + up\_x) - x$$

همانطور که مشخص است این یادگیری داری پارامتر نیست. می توانیم این روش برای ایجاد اکتشاف و استخراج با کنترل پارامتر انجام داد

$$x_{new} = (Low\_x + up\_x) - k \times x$$

اکتشاف فرآیند $k \leq 1$

اکتشاف استخراج $k \geq 1$

در این رابطه k همان نقش R_{cloud} در رابطه ی ذره کوانتمی را دارد. پس فرمول تغییر می کنه به

$$x_{new} = (Low\_x + up\_x) - R_{cloud} \times x$$

از طرفی داریم:

$$X\_OBL = delta - x$$

$$Delta=(upper + lower) \text{ or } (Gbest + dim)$$

در نتیجه داریم:

$$\mathbf{X_{Opposition,j}=Gbest_{bestSwarm,j}} - R_{cloud} \times x + dim \quad \textbf{رابطه ٤-١}$$

در اینجا تنوع براساس موقعیت x تعیین می شود. و سپس بر اساس مقدارGbest و موقعیت x مقدار تضاد x را تخمین می زنیم. بر خلاف روش هایی که از توزیع یکنواخت و توزیع گوسین و ... استفاده می کنند. در این روش ها یک نوع یادگیری و جود دارد. و سعی دارند که یک تخمینی بر اساس مقادیر موجود بزنند. درحالی که در یک سیستم تصادفی، وضعیت فعلی سیستم مستقل از وضعیت پیشین آن است .در مدل تصادفی، به هیچ عنوان نمی توان تخمینی از وضعیت بعدی داشت .

Algorithm: Dyn-GSO-OBL

Begin
FOR swarm i
  FOR Particle j
    Generate OBL the initial members;
    Evaluate the fitness values of initial members;
  END FOR
Calculate Gbest
END FOR
Initialize Test_point
WHILE (the termination conditions are not met)
FOR swarm i
  FOR particle j
    Choose producer: Find the producer G*j*.**X***p* of the group;
    Perform producing: perform producer behaviour;
    Perform scrounging: Randomly select a percentage from the rest members to perform scrounging; All scroungers will follow its group best member G*j*.**X***p*;
    Perform dispersion: The rest members (rangers) will perform ranging;
    Fitness evaluation: Calculate the fitness function of each member from this group by combining its variables with the best solutions found so far by the remaining groups;
  END FOR
END FOR
Execute exclusion and anti convergence[6]
Evaluate Test_point
  if new value is different from the previous one
    Re-evaluate each particle attractor
End
foreach swarm i

```
   Update Oi,j based on equation 4.1
    if f(Oi,j) > f(Gbesti)
    Gbesti = Oi,j
    end
endfor

END WHILE
end
```

شکل ۲ : شبه کد الگوریتم پیشنهادی دوم (Dyn-GSO-OBL)

۳- ارزیابی

۳-۱- تابع محک قله‌های متحرک [۱۳]

با هدف ایجاد پلی بر روی شکاف میان مسائل بسیار پیچیده و دشوار دنیای واقعی و مسائل ساده‌ی بهینه‌سازی، مسئله‌ای چند بعدی شامل تعداد مختلفی قله پیشنهاد شده است که ارتفاع، پهنا و موقعیت هر قله هنگامی که تغییری در محیط روی می‌دهد به آرامی تغییر می‌کنند[۲۹،۲۷].

مسئله‌ی محک قلل متحرک شامل m قله در فضای پارامتر با مقادیر حقیقی n بعدی می‌باشد و تابع شایستگی در آن بصورت بیشینه بر روی تمام توابع قله‌ها تعریف می‌شود. و می‌توان آنرا به صورت زیر فرمول‌بندی کرد:

$$f(\vec{x},t)=\max(B(\vec{x},max_{i=1\dots m}\,P(h_i(t),w_i(t),\vec{p}_i(t)))) \quad (۷-۲)$$

در این فرمول $B(\vec{x})$ چشم‌انداز پایه‌ی مسئله است که با گذشت زمان تغییر نمی‌کند، و P تابعی است که معرف شکل یک قله است بطوریکه هر یک از m قله‌ی موجود در فضای جستجو، پارامترهای متغیر با زمان ارتفاع h، پهنا w و بردار مکان p مختص خود را دارند.

پس از آنکه مقدار تابع هدف به تعداد دفعات Δ_e بار ارزیابی می‌شود، ارتفاع، پهنا و موقعیت هر قله تغییر می‌کند. ارتفاع و پهنای هر قله بوسیله‌ی جمع جبری با یک متغییر تصادفی گوسی تغییر می‌کنند و موقعیت هر قله بوسیله‌ی یک بردار v با طول ثابت s حرکت داده می‌شود، بنابراین پارامتر s (به همراه پارامترهای مقیاس نسبتاً کم اهمیت‌تر ارتفاع و پهنا) امکان کنترل شدت تغییرات محیطی را فراهم می‌کند و Δ_e هم نمایانگر فرکانس تغییرات محیط است.

پارامتر جدید λ مشخص می‌کند که چه میزان از تغییرات موقعیت یک قله وابسته به حرکت قبلی آن است. اگر پارامتر $\lambda = 0.0$ باشد حرکت هر قله در جهتی کاملا تصادفی می‌باشد و اگر $\lambda = 1.0$ باشد هر قله همیشه در همان جهت سابق خود حرکت می‌کند(تا زمانی که با مرز فضای پارامترها برخورد کند که در اینصورت همانند برخورد یک توپ به دیوار به عقب پس رانده می‌شود). تاثیر پارامتر λ بر روی جهت حرکت یک قله در شکل (۲-۳) نمایش داده شده است.

بطور صریح تر می توان حرکت یک قله را بصورت فرمول‌های زیر مدل کرد:

$$\sigma \in N(0,1)$$

$$h_i(t) = h_i(t-1) + \text{height\_severity}.\sigma \quad (۸-۲)$$

$$w_i(t) = w_i(t-1) + \text{width\_severity}.\sigma \quad (۹-۲)$$

$$p_i(t) = p_i(t-1) + \vec{v_i}(t) \quad (۱۰-۲)$$

بردار جابجایی $\vec{v_i}$ ترکیبی خطی از بردار تصادفی $\vec{r}$ و بردار جابجایی $\vec{v_i}(t-1)$ است که به اندازه‌ی طول s نرمال شده است:

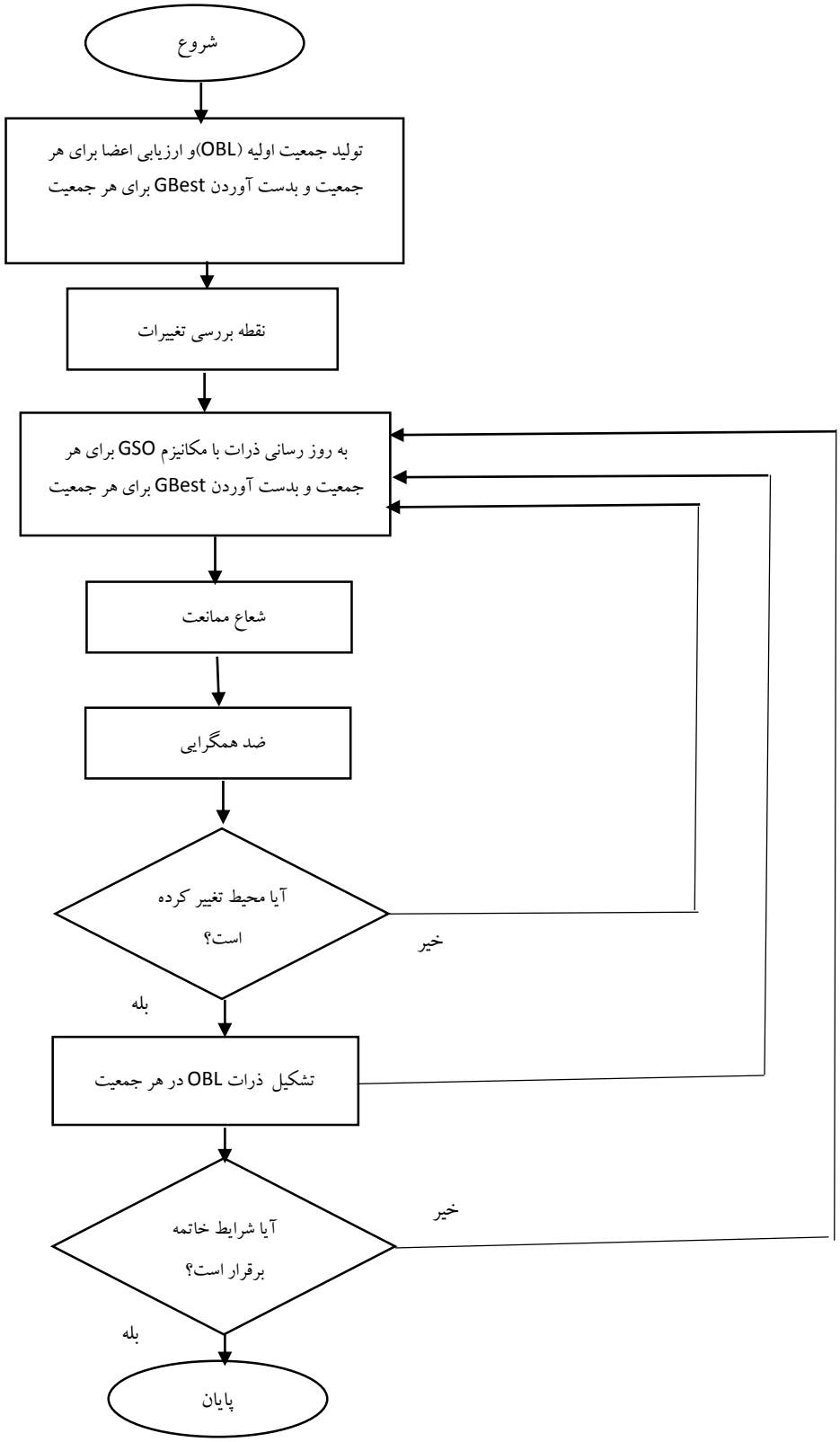
$$\vec{v_i}(t) = \frac{s}{|\vec{r}+\vec{v}_i(t-1)|}((1-\lambda)\vec{r} + \lambda\,\vec{v_i}(t-1)) \quad (۱۱-۲)$$

بردار تصادفی $\vec{r}$ با تولید اعداد تصادفی برای هر بعد و نرمال‌سازی آنها با طول s بدست می‌آید. مختصات x، ارتفاع h، و پهنای w برای هر قله بصورت تصادفی در محدوده‌ای مشخص مقداردهی اولیه می‌شوند. حال به وسیله‌ی یک مولد اعداد تصادفی، می‌توان مسائل محک متعددی را با خصوصیات بنیادین یکسان تولید نمود.

پیچیدگی تابع براحتی با افزایش تعداد ابعاد و یا تعداد قله‌ها تغییر می‌کند. علاوه بر این می‌توان اندازه‌ی گام تغییرات را براحتی با استفاده از پارامتر s تغییر داد.

---

[۱۳] Moving Peaks Benchmark (MPB)



شکل ۳: فلوچارت الگوریتم پیشنهادی (Dyn-GSO-OBL)

۳- نتایج:

تابع محکی که در این زیربخش به منظور ارزیابی کارایی الگوریتم پیشنهادی مورد استفاده قرار گرفته است، تابع محک قله های متحرک (MPB) می باشد ، بطور کامل در رابطه با آن بحث شد. همچنین تنظیمات پارامترهای مربوط به این تابع محک، مطابق با جدول۱ است .این تابع محک دارای سه سناریو می باشد که معروفترین سناریوی آن، سناریوی شماره ۲ می باشد .جدول ۱ تنظیم پارامترهای مربوط به این سناریو را نشان داده است .در این زیربخش به منظور ارزیابی بهتر الگوریتم پیشنهادی، مقادیر دیگری نیز برای تعدادی از پارامترها از جمله تعداد قله از ۱ قله تا ۲۰۰ قله و فرکانس تغییر از ۵۰۰ تا ۵۰۰۰ در نظر گرفته شده است.

جدول ۱ : پارامترهای مربوط به مسئله. MPB

نام پارامتر	مقدار پارامتر در سناریوی ۲	مقدار پارامترهای آزمایش شده
تعداد قله ها M	۱۰	**بین ۱ تا ۲۰۰**
فرکانس تغییر CF	۵۰۰۰	**۵۰۰،۱۰۰۰،۲۵۰۰ و ۵۰۰۰**
میزان تغییر ارتفاع	۷,۰	**۷,۰**
شکل قله	Cone	**Cone**
تابع اولیه	ندارد	**ندارد**
طول جابجایی S	۱,۰	**۱,۰**
تعداد ابعاد N	۵	**۵**
محدوده مکانی قله ها	[۰و۱۰۰-۰و۰]	[۰و۱۰۰-۰و۰]
محدوده پارامتر ارتفاع	[۰و۷۰-۰و۳۰]	[۰و۷۰-۰و۳۰]
محدوده پارامتر عرض	[۱-۱۲]	[۱-۱۲]
مقدار ارتفاع اولیه قله ها	۵۰	۵۰

لازم بذکر است که به منظور سنجش کارایی الگوریتم پیشنهادی از خطای برون خطی استفاده شده است.

٤- تنظیم پارامترهای مربوط به الگوریتم پیشنهادی(Dyn-GSO-OBL)

تنظیم تعدادی از پارامترها در الگوریتم Dyn-GSO-OBL مطابق با مقادیر تنظیم شده در مقاله اصلی و مقادیر مربوط به سایر پارامترهای پیشنهاد شده بر اساس آزمایشات بسیار بدست آمده اند .در این الگوریتم تعداد دسته ها برابر با ۱۰ و تعداد ذرات در هر دسته برابر با ۵ در نظر گرفته شده اند .پارامترهای مربوط به GSO شامل $\varphi_{i1}^{k} = \left(\frac{\pi}{4}\right)$ زاویه شروع است .انحصار و ضدهمگرایی مطابق با [۲۸] در نظر گرفته شده است .مقدار LowL و HighL نیز به ترتیب برابر ۰ و۱۰۰در نظر گرفته شدهاند .

٥-٤-١- تحلیل آماری

از آنجا که ارزیابی کارایی الگوریتم پیشنهادی تنها با مقایسه میانگین های بدست آمده از الگوریتم پیشنهادی و سایر الگوریتم ها به تنهایی قابل قبول نبوده و کافی نمی باشد، لذا به منظور ارزیابی بهتر الگوریتم پیشنهادی و اثبات معنادار بودن اختلاف آن با الگوریتم های دیگر، از تحلیل های آماری بهره گرفته شده است. برای این منظور از آزمون آماری one sample t test (آزمون t تک نمونه ای) استفاده شده است .این تست جزو آزمونهای پارامتریک بوده و به منظور بررسی میانگین یک متغیر کمی با یک مقدار ثابت بکار می رود[۲۹]. یکی از شروط استفاده از این تست آماری، شرط نرمال بودن داده ها می باشد .اگرچه طبق قضیه حد مرکزی که در زیر مطرح شده است چنانچه تعداد نمونه ها بزرگتر یا مساوی با ۳۰ باشد، نیازی به اثبات نرمال بودن جامعه آماری نبوده و توزیع میانگین نمونه، بصورت نرمال فرض می گردد.

- **قضیه حد مرکزی**

اگر یک نمونه تصادفی n (n>=30) تایی از یک جامعه دلخواه با میانگین μ و انحراف معیار σ انتخاب شود، توزیع میانگین نمونه بصورت نرمال خواهد بود .با توجه به توضیحات بالا در این رساله کلیه آزمایشات حداقل ۳۰ بار بصورت مجزا و با پیگرد های متفاوت برای تولید اعداد تصادفی متفاوت تکرار شده اند تا بتوان با بهره گیری از این آزمون، معنادار بودن اختلاف بین میانگین مقادیر بدست آمده از الگوریتم را با مقدار بدست آمده از بهترین الگوریتم بر روی سناریو های یکسان مورد ارزیابی قرار داد .فرضهای H_0 و H_1 در این آزمون فرض به صورت رابطه۳-۱ تا رابطه ۳-۲ تعریف می شوند .فرض H_0 مشخص می کند که اختلاف معناداری بین میانگین خطاهای برون خطی بدست آمده از الگوریتم پیشنهادی با خطای برون خطی گزارش شده از بهترین الگوریتم بر روی سناریوی j وجود ندارد و این دو با یکدیگر برابر هستند .اما فرض H_1 مشخص می کند اختلاف معناداری بین میانگین خطاهای برون خطی بدست آمده از الگوریتم پیشنهادی با خطای برون خطی گزارش شده از بهترین الگوریتم بر روی سناریوی j وجود دارد.

$$H0:\mu_{j=a_j^*} \quad \text{رابطه۳-۱}$$

$$H1:\mu_{j\neq a_j^*} \quad \text{رابطه ۳-۲}$$

$$a_i = argmin(\mu_{i,j})^* \quad \text{رابطه۳-۳}$$

$$i^* \in \text{all algorithms, } j \in \text{all scenarios}$$

در روابط بالا a_i ، بهترین مقدار بدست آمده توسط بهترین الگوریتم بر روی سناریوی j و μ_j میانگین بدست آمده از اعمال الگوریتم پیشنهادی بر روی سناریوی j می باشد. آزمون T-test به صورت two-tailed t test با درجه آزادی ۲۹ و آلفا برابر۰,۰۵ انجام شده است . چنانچه مقدار p-value کوچکتر از آلفا برابر با ۰,۰۵ باشد، می توان گفت که با سطح اطمینان ۹۵ درصد، میانگین بدست آمده از الگوریتم پیشنهادی از میانگین بهترین نتیجه بدست آمده از الگوریتم های دیگر و بر روی سناریوهای مختلف بهتر بوده است .برای تسهیل در مقایسه، در جداول زیر چنانچه الگوریتمی با سطح اطمینان۹۵ درصد، نتیجه بهتری نسبت به سایر الگوریتم ها ارائه داده باشد به صورت پررنگ نوشته شده است(فرض H_0 رد شده است) همچنین چنانچه اختلاف معناداری بین نتیجه بهترین الگوریتم بهتر الگوریتم پیشنهادی وجود نداشته باشد، هر دو مقدار به صورت پررنگ نوشته شده اند(فرض H_0 پذیرفته شده است.)

۵- ارزیابی کارایی الگوریتم پیشنهادی (Dyn-GSO-OBL)

تاثیر مقادیر مختلف برای پارامتر فرکانس تغییر بر روی کارایی الگوریتم برای ارزیابی الگوریتم Dyn-GSO-OBL ، الگوریتم شناخته شده و پایه ای برای بهینه سازی در محیطه ای پویا به نام ها ی

[۱]MQSO، [۲] AMQSO، [۳]MPSO، [۴]HMPSO، [۵]APSO، [۶]CPSO ، [۷]CPSOR و [۸] DYNDE به منظور مقایسه در نظر گرفته شده است . این الگوریتمها با توجه به مقادیری که در مقالاتشان برای پارامترها عنوان شده و الگوریتمی که توسط اینجانب مورد پیاده سازی قرار گرفته اند و نتایج بدست آمده از آنها در این آزمایش نشان داده شده است .آزمایشات ۳۰ بار بصورت مجزا و با پیگردهای متفاوت برای اعداد تصادفی تکرار شده اند و متوسط مقادیر خطای برون خطی و خطای استاندارد الگوریتم Dyn-GSO-OBL به همراه سه الگوریتم دیگر در جداول 2 تا 5 ، برای فرکانس های تغییر 500، 1000 ، 2500 و 5000 و با تعداد قله های مختلف نشان داده شده است .در این جداول خطای استاندارد در کنار خطای برون خطی درون پرانتز نشان داده شده و برای و نتیجه بهتر در هر ردیف برای تسهیل مقایسه به صورت پررنگ نوشته شده است .همچنین زمانیکه بهترین الگوریتم ها در یک ردیف تفاوت معناداری با یکدیگر ندارند، همه بصورت پررنگ نوشته شده اند.

ارزیابی کارایی الگوریتم پیشنهادی

جدول ۲: مقایسه خطای برون خطی و خطای استاندارد الگوریتمها برای فرکانسهای تغییر ۵۰۰

الگوریتم	تعداد قله ها							
	۱	۵	۱۰	۲۰	۳۰	۵۰	۱۰۰	۲۰۰
MQSO	۳۳,۶۷(۳,۴۲)	۱۱,۹۱(۰,۷۶)	۹,۶۲(۰,۳۴)	۹,۰۷(۰,۲۵)	۸,۸(۰,۲۱)	۸,۷۲(۰,۲)	۸,۵۴(۰,۱۶)	۸,۱۹(۰,۱۷)
AMQSO	۳,۰۲(۰,۳۲)	۵,۷۷(۰,۵۶)	۵,۳۷(۰,۴۲)	۶,۸۲ (۰,۳۴)	۷,۱۰(۰,۳۹)	۷,۷۵(۰,۳۲)	۷,۳۴(۰,۳۱)	۷,۴۸(۰,۱۹)
MPSO	۸,۷۱(۰,۴۸)	۶,۶۹(۰,۲۶)	۷,۱۹(۰,۲۳)	۸,۰۱(۰,۱۹)	۸,۴۳(۰,۱۷)	۸,۷۶(۰,۱۸)	۸,۹۱(۰,۱۷)	۸,۸۸(۰,۱۴)
HMPSO	۸,۵۳(۰,۴۹)	۷,۴۰(۰,۳۱)	۷,۵۶(۰,۲۷)	۷,۸۱(۰,۲۰)	۸,۳۳(۰,۱۸)	۸,۸۳(۰,۱۷)	۸,۸۵(۰,۱۶)	۸,۸۵(۰,۱۶)
APSO	۴,۸۱(۰,۱۴)	۴,۹۵(۰,۱۱)	۵,۱۶(۰,۱۱)	۵,۸۱(۰,۰۸)	۶,۰۳(۰,۰۷)	۵,۵۹(۰,۰۶)	۶,۰۸(۰,۰۶)	۶,۲۰(۰,۰۴)
Dyn-GSO-OBL	**۳,۵۰(۰,۶۰)**	**۳,۶۷ (۰,۳۲)**	**۴,۵۹(۰,۳۵)**	**۵/۸۷(۰,۲۰)**	**۵,۴۸ (۰,۳۰)**	**۴,۳۰(۰,۳)**	**۵,۶۷(۰,۱۸)**	**۵,۶۷(۰,۲۱)**

همانطور که در جدول۲ مشاهده می شود، الگوریتم Dyn-GSO-OBL بهتر از کلیه الگوریتم ها بر روی فرکانس تغییر ۵۰۰ و تعداد قله های متفاوت عمل نموده است .همچنین در شکل ۴، مقایسه خطای جاری و خطای استاندارد Dyn-GSO-OBL برای فرکانس تغییر ۵۰۰ و تعداد قله های ۱۰ نشان داده شده است.

جدول۳ : مقایسه خطای برون خطی و خطای استاندارد الگوریتمها برای فرکانسهای تغییر ۱۰۰۰

الگوریتم	تعداد قله ها							
	۱	۵	۱۰	۲۰	۳۰	۵۰	۱۰۰	۲۰۰
MQSO	۵,۵۴(۰,۱۱)	۵/۸۳ (۰/۱۳)	۵,۸۷(۰,۱۳)	۵,۸۱(۰,۱۵)	۵,۸۵(۰,۱۵)	۵,۷۱(۰,۲۲)	۶,۵۶(۰,۳۸)	۱۸,۶(۱,۶۳)
AMQSO	۲/۳۳(۰/۳۱)	۲/۹۰ (۰/۳۲)	۴/۵۶ (۰/۴۰)	۵/۳۶ (۰/۴۷)	۵/۲۰ (۰/۳۸)	۶/۰۶ (۰/۱۴)	۴/۷۷ (۰/۴۵)	۵/۷۵ (۰/۲۶)
MPSO	۴/۴۴ (۰/۲۴)	۳/۹۳ (۰,۱۶)	۴/۵۷ (۰,۲۳)	۸,۰۱(۰,۱۹)	۸,۴۳(۰,۱۷)	۸,۷۶(۰,۱۸)	۸,۹۱(۰,۱۷)	۸,۸۸(۰,۱۴)
HMPSO	۴,۴۶(۰,۴۹)	۴,۲۷(۰,۰۸)	۶,۶۱(۰,۷)	۴,۶۶(۰,۱۲)	۴,۸۳(۰,۰۹)	۴,۹۶(۰,۰۳)	۵,۱۴(۰,۰۸)	۵,۲۵(۰,۰۸)
APSO	۲,۷۲(۰,۰۴)	۲,۹۹(۰,۰۹)	۳,۷۸(۰,۰۸)	۴,۱۳(۰,۰۶)	۴,۱۲(۰,۰۴)	۴,۱۱(۰,۰۳)	۴,۲۶(۰,۰۴)	۴,۲۱(۰,۰۲)
Dyn-GSO-OBL	**۳,۵۰(۰,۶۰)**	**۳,۶۷ (۰,۳۲)**	**۴,۵۹(۰,۳۵)**	**۵/۸۷(۰,۲۰)**	**۵,۴۸ (۰,۳۰)**	**۴,۳۰(۰,۰۴)**	**۴,۲۰(۰,۰۳)**	**۳,۷۴ (۰,۰۳)**

همانطور که در جدول ۳ مشاهده می شود، الگوریتم Dyn-GSO-OBL بهتر از کلیه الگوریتم ها ی [۲۰]MQSO،[۲۱]AMQSO،[۲۲]MPSO، [۲۴]APSO [۲۳]HMPSO، [۲۵]CPSO،[۲۶] CPSOR بر روی فرکانس تغییر ۱۰۰۰ و تعداد قله های متفاوت عمل نموده است .همچنین در شکل ۵ مقایسه خطای جاری و خطای استاندارد Dyn-GSO-OBL برای فرکانس تغییر ۱۰۰۰ و تعداد قله های ۱۰ نشان داده شده است.

جدول ٤ : مقایسه خطای برون خطی و خطای استاندارد الگوریتمها برای فرکانسهای تغییر ۲۵۰۰

الگوریتم	تعداد قله ها							
	۱	۵	۱۰	۲۰	۳۰	۵۰	۱۰۰	۲۰۰
MQSO	۷,۶۴(۰,۴۶)	۳,۲۶(۰,۲۱)	۳,۱۲(۰,۱۴)	۳,۵۸(۰,۱۳)	۳,۶۳(۰,۱)	۳,۶۳(۰,۱)	۳,۵۸(۰,۰۸)	۳,۳(۰,۰۶)
AMQSO	۰,۸۷(۰,۱۱)	۲,۱۶(۰,۱۹)	۲,۴۹(۰,۱)	۲,۳۷ (۰,۱۱)	۳,۲۴(۰,۱۸)	۳,۶۸(۰,۱۵)	۳,۵۳(۰,۱۴)	۳,۰۷(۰,۱۲)
MPSO	۱,۷۹(۰,۱)	۲,۰۴(۰,۱۲)	۲,۲۶(۰,۱۶)	۳,۰۷(۰,۱۱)	۳,۱۵(۰,۰۸)	۳,۲۶(۰,۰۷)	۳,۳۱(۰,۰۵)	۳,۳۶(۰,۰۵)
HMPSO	۱,۷۵(۰,۱)	۱,۹۲(۰,۱۱)	۲,۳۹(۰,۱۶)	۲,۴۶(۰,۰۹)	۲,۵۷(۰,۰۵)	۲,۶۵(۰,۰۵)	۲,۷۲(۰,۰۴)	۲,۸۱(۰,۰۴)
APSO	۱,۰۶(۰,۰۳)	۱,۵۵(۰,۰۵)	۲,۱۷(۰,۰۷)	۲,۵۱(۰,۰۵)	۲,۶۱(۰,۰۲)	۲,۲۶(۰,۰۲)	۲,۲۶(۰,۰۲)	۲,۴۶(۰,۰۱)
Dyn-GSO-OBL	**۱,۰۲(۰,۱۳)**	**۱,۵۰(۰,۱۰)**	**۲,۱۰(۰,۰٦)**	**۲,۳۰(۰,۱۱)**	**۲,۲۰(۰,۱۱)**	**۲,۲٤(۰,۱۱)**	**۲,۱۵(۰,۰۹)**	**۲,٤۲(۰,۰٦)**

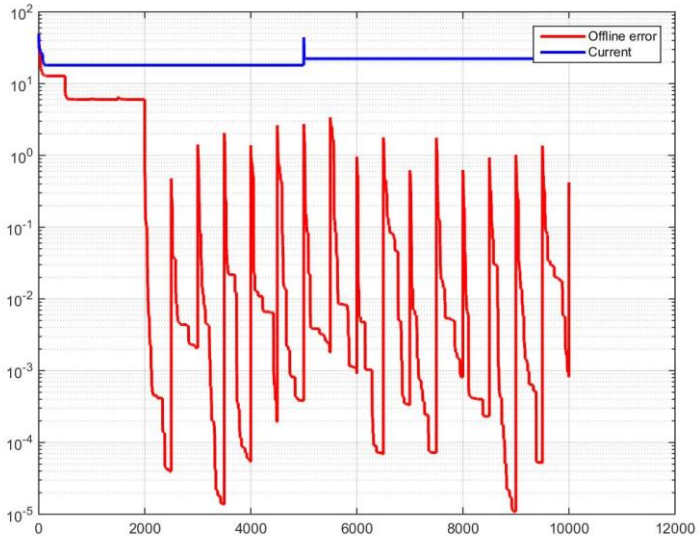
همانطور که در جدول ۴ مشاهده می شود، الگوریتم Dyn-GSO-OBL بهتر از کلیه الگوریتم های [۲۰]MQSO،[۲۱]AMQSO،[۲۲]MPSO، [۲۴]APSO [۲۳]HMPSO، [۲۵]CPSO،[۲۶] CPSOR بر روی فرکانس تغییر ۲۵۰۰ و تعداد قله های متفاوت عمل نموده است .همچنین در شکل ۶، مقایسه خطای جاری و خطای استاندارد Dyn-GSO-OBL برای فرکانس تغییر ۲۵۰۰ و تعداد قله های ۱۰ نشان داده شده است.

جدول ۵: مقایسه خطای برون خطی و خطای استاندارد الگوریتم ها برای فرکانس های تغییر ۵۰۰۰

	تعداد قله ها							
الگوریتم	**۱**	**۵**	**۱۰**	**۲۰**	**۳۰**	**۵۰**	**۱۰۰**	**۲۰۰**
MQSO	۴,۹۲(۰,۲۱)	۱,۸۲(۰,۰۸)	۱,۸۵(۰,۰۸)	۲,۴۸(۰,۰۹)	۲,۵۱(۰,۱)	۲,۵۳(۰,۰۸)	۲,۳۵(۰,۰۶)	۲,۲۴(۰,۰۵)
AMQSO	۰,۵۱(۰,۰۴)	۱,۰۱(۰,۰۹)	۱,۵۱(۰,۱)	۲ (۰,۱)	۲,۱۹(۰,۱۷)	۲,۴۳(۰,۱۳)	۲,۶۸(۰,۱۲)	۲,۶۲(۰,۱)
MPSO	۰,۹(۰,۰۵)	۱,۲۱(۰,۱۲)	۱,۶۱(۰,۱۲)	۲,۰۵(۰,۰۸)	۲,۱۸(۰,۰۶)	۲,۳۴(۰,۰۶)	۲,۳۲(۰,۰۴)	۲,۳۴(۰,۰۳)
HMPSO	۰,۸۷(۰,۰۵)	۱,۱۸(۰,۰۴)	۱,۴۲(۰,۰۴)	۱,۵(۰,۰۶)	۱,۶۵(۰,۰۴)	۱,۶۶(۰,۰۲)	۱,۶۸(۰,۰۳)	۱,۷۱(۰,۰۲)
APSO	۰,۵۳(۰,۰۱)	۱,۰۵(۰,۰۶)	۱,۳۱(۰,۰۳)	۱,۶۹(۰,۰۵)	۱,۷۸(۰,۰۲)	۱,۹۵(۰,۰۲)	۱,۹۵(۰,۰۱)	۱,۹(۰,۰۱)
CPSO	۷,۸۰(۱,۰۰)	۴,۲۰(۰,۳۲)	۴,۵۰(۰,۲۶)	۴,۰۰(۰,۱۶)	۳,۵۰(۰,۱۶)	۳,۵۰(۰,۱۳)	۳,۲۰(۰,۱۳)	۲,۵۰(۰,۰۹۱)
CPSOR	۶,۱۰(۰,۸۴)	۲,۹۰(۰,۳۴)	۲,۶۰(۰,۲۰)	۲,۶۰(۰,۳۰)	۲,۴۰(۰,۱۲)	۲,۵۰(۰,۱۰)	۲,۵۰(۰,۱۰)	۲,۳۰(۰,۰۹۷)
Dyn-GSO-OBL	**۰,۴۰(۰,۰۹)**	**۰,۸۲(۳,۴۲)**	**۱,۱۰(۰,۰۲)**	**۰,۹۰(۰,۰۲)**	**۱,۱۰(۰,۰۴)**	**۱,۳۲(۰,۰۳)**	**۱,۶۱(۰,۰۲)**	**۱,۹۹(۰,۰۴)**

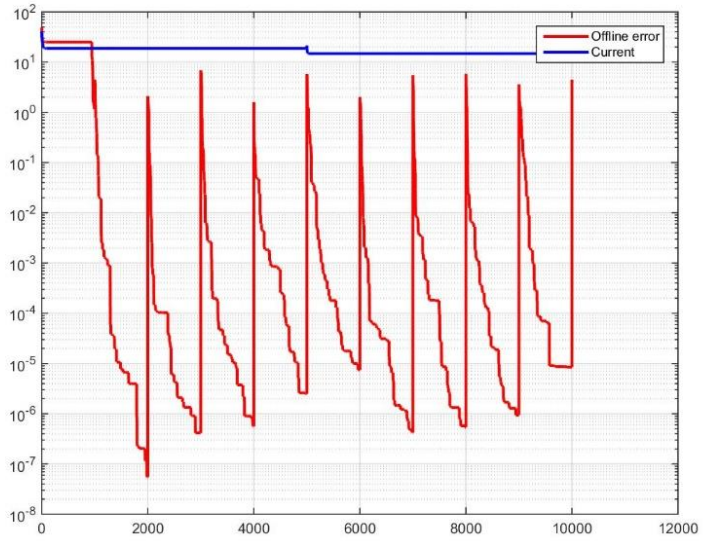
همانطور که در جدول ۵ مشاهده می شود، الگوریتم Dyn-GSO-OBL بهتر از کلیه الگوریتم ها ی [۲۰]MQSO،[۲۱]AMQSO،[۲۲]MPSO، [۲۴]APSO ،[۲۳]HMPSO، [۲۵]CPSO،[۲۶] CPSOR بر روی فرکانس تغییر ۵۰۰۰ و تعداد قله های متفاوت عمل نموده است .همچنین در شکل ۶، مقایسه خطای جاری و خطای استاندارد Dyn-GSO-OBL برای فرکانس تغییر ۵۰۰۰ و تعداد قله های ۱۰ نشان داده شده است.

با فرکانس تغییر ۵۰۰ و تعداد قله های مختلف



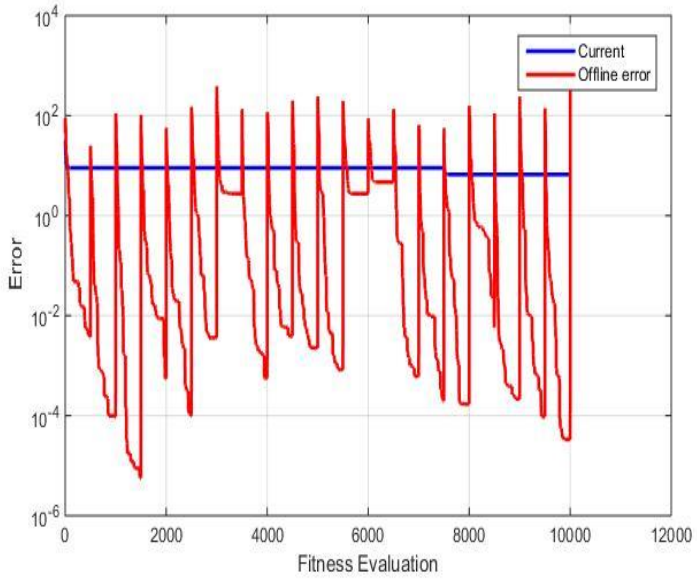
شکل٤: مقایسه خطای برون خطی و خطای استاندارد الگوریتم هاDyn-GSO-OBL(CF=500)

با فرکانس تغییر ۱۰۰۰ و تعداد قله های مختلف



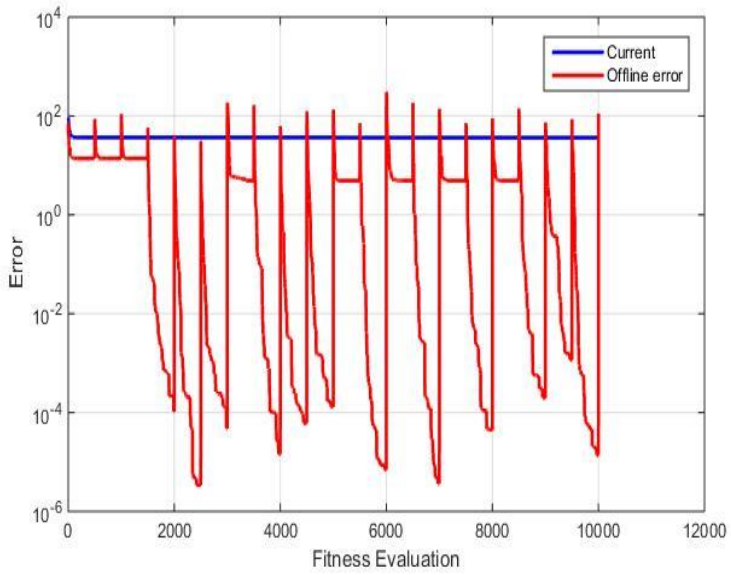
شکل۵: مقایسه خطای برون خطی و خطای استاندارد الگوریتم هاDyn-GSO-OBL(CF=1000)

با فرکانس تغییر ۲۵۰۰ و تعداد قله های مختلف



شکل٦: مقایسه خطای برون خطی و خطای استاندارد الگوریتم هاDyn-GSO-OBL(CF=2500)

با فرکانس تغییر ۵۰۰۰ و تعداد قله های مختلف



شکل۷: مقایسه خطای برون خطی و خطای استاندارد الگوریتم هاDyn-GSO-OBL(CF=5000)

۵– نتیجه گیری وکارهای آینده:

بر اساس یافته‌های این مقاله در زمینه‌ی بهینه‌سازی در محیط‌های پویا می‌توان موضوعات پژوهشی مختلفی رو به‌عنوان زمینه‌های تحقیقاتی آینده مطرح کرد.

در مسائل پویا، چالش های اساسی و مهمی پیش روی یک الگوریتم بهینه سازی است .برخی از این چالش ها می توانند شامل مشکل به روز کردن حافظه افراد جمعیت بعد از هر تغییر در محیط، حفظ تنوع در محیط بعد از همگرایی افراد به بهینه موردنظر، ظرفیت محدود حافظه، شناسایی زمان تغییرات در محیط باشند.حفظ تنوع پس از تغییرات در محیط یکی از چالش های محیط های پویا می باشد. فقط از یک رویکرد مبتنی برتضاد به نام OBL در این مقاله استفاده شده است. در آینده رویکردهای دیگر مورد بررسی قرار می گیرند.

مراجع:

۱. **T. Blackwell, J. Branke, and X. Li, (2008) “Particle Swarms for Dynamic Optimization Problems,” in Swarm Intelligence, Natural Computing Series, vol. Part II, Springer Berlin Heidelberg, pp. 193 – 217.**

۲. **H.R. Tizhoosh, (2005) “Opposition-based learning: a new scheme for machine intelligence”, Proceedings of International Conference on Computational Intelligence for Modeling Control and Automation, 695–701.**

۳. **H. Wang, Zh. Wu, Sh. Rahnamayan, Y. Liu, M. Ventresca, (2011) “Enhancing particle swarm optimization using generalized opposition-based learning” , Information Sciences.**

٤.Q. Xu, n. LeiWang, N. Wang, Xi. Hei, Li. Zhao, (2014) ”A review of opposition-based learning from2005 to2012”, Engineering Applications of Artificial Intelligence.

٥.H. Wang, Z. Wu, Y. Liu, J. Wang, D. Jiang, L. Chen, (2009) “Space transformation search: a new evolutionary technique”, Proc World Summit Genetic Evolut. Comput.

٦.S. Rahnamayan, G.G. Wang, (2009) “Center-based sampling for population-based algorithms”. IEEE Congress on Evolutionary Computation, Trondheim, Norway.

٧.Nasrabadi, M. Sohrabi, Y. Sharafi, and M. Tayari. (2016), "A parallel grey wolf optimizer combined with opposition based learning." Swarm Intelligence and Evolutionary Computation (CSIEC), 2016 1st Conference on. IEEE.

٨.W. Wei, et al. (2016),"Constrained differential evolution using generalized opposition-based learning." Soft Computing .

٩.H. Wang, et al. (2011),"Enhancing particle swarm optimization using generalized opposition-based learning." Information Sciences.

١٠.H. Wang, et al.(2007), "Opposition-based particle swarm algorithm with Cauchy mutation." Evolutionary Computation, 2007. CEC 2007. IEEE Congress on.

١١.Ch. Qu, et al. (2017), "Chicken Swarm Optimization Based on Elite Opposition-Based Learning." Mathematical Problems in Engineering.

١٢. M. Huang, W. Mingxu, and Liang Xu. "An improved genetic algorithm using opposition-based learning for flexible job-shop scheduling problem." Cloud Computing and Internet of Things (CCIOT), 2016 2nd International Conference on. IEEE, 2016.

١٣.Azad, H. R. Lashgarian. "An Application of Opposition Based Colonial Competitive Algorithm to Solve Network Count Location Problem." (2013),International Journal of Intelligent Systems and Applications.

١٤. Y. Kumar, and G. Sahoo. (2015), "An Improved Cat Swarm Optimization Algorithm Based on Opposition-Based Learning and Cauchy Operator for Clustering." Journal of Information Processing Systems .

١٥. H. Wang, W. Wang, and H. Sun. (2015), "Firefly algorithm with generalised opposition- based learning." International Journal of Wireless and Mobile Computing.

١٦.Yu, Shuhao, et al. (2015),"Enhancing firefly algorithm using generalized opposition-based learning. Computing .

١٧.H. Liu, G. Ding, and H. Sun. (2012),"An improved opposition-based disruption operator in gravitational search algorithm." Computational Intelligence And Design (Iscid), 2012 Fifth International Symposium On. Vol. 2. IEEE.

١٨. S. ÖZYÖN, et al. (2015),"Opposition-based gravitational search algorithm applied to economic power dispatch problems consisting of thermal units with emission constraints." Turkish Journal of Electrical Engineering & Computer Sciences 23.Sup.

١٩.H. Shan, Q. H. Wu, and J. R. Saunders. "Group search optimizer: an optimization algorithm inspired by animal searching behavior." IEEE transactions on evolutionary computation Vol. 13, No. 5, pp. 973-990.

٢٠.T. Blackwell and J. Branke, (2006),"Multi swarms, exclusion, and anti-convergence in dynamic environments", IEEE Transactions on Evolutionary Computation, Vol. 10, pp. 459-472.

٢١.T. Blackwell, J. Branke, and X. Li, (2008),"Particle swarms for dynamic optimization problems", Swarm Intelligence, pp. 193-217.

٢٢.T. Blackwell and J. Branke, (2006),"Multi swarms, exclusion, and anti-convergence in dynamic environments", IEEE Transactions on Evolutionary Computation, Vol. 10, No. 4, pp. 459–472.

٢٣.M. Kamosi, A. B. Hashemi, and M. R. Meybodi, (2010)"A hibernating multi-swarm optimization algorithm for dynamic environments", in Proceedings of the World Congress on Nature and Biologically Inspired Computing, pp. 363–369.

٢٤.Z.H. Zhan, J. Zhang, Y. Li. (2009),"Adaptive Particle Swarm Optimization", IEEE Transactions on Evolutionary Computation, Vol. 39, No. 6, pp. 1362 – 1381.

٢٥.S. Yang and C. Li, (2010)"A Clustering Particle Swarm Optimizer for Locating and Tracking Multiple Optima in Dynamic Environments," IEEE Transactions on Evolutionary Computation, Vol. 14, pp. 959-974.

٢٦.C. Li and S. Yang. (2011), "A general framework of multi-population methods with clustering in undetectable dynamic environments", IEEE Transactions on Evolutionary Computation, Vol. 16, pp. 556 - 577.

٢٧.J. Branke, and H. Schmeck. (2003),"Designing evolutionary algorithms for dynamic optimization problems." Advances in evolutionary computing .

٢٨.T. Blackwell and J. Branke, (2006)"Multiswarms, exclusion, and anti-convergence in dynamic environments," IEEE Transactions on Evolutionary Computation, vol. 10, pp. 459-472.

٢٩. S. J. Coakes and L. G. Steed.(1999) SPSS: Analysis without anguish: Versions 7.0, 7.5, 8.0 for Windows: Jacaranda Wiley.