حل مسائل بهینه‌سازی پویا با یک الگوریتم رقابت استعماری بهبودیافته

مهدی صادقی مقدم[1]*، صمد نجاتیان[2]، حمید پروین[3]،کرم‌الله باقری فرد[4]، هادی یعقوبیان[5]

دانشجوی دکترای گروه مهندسی کامپیوتر، واحد یاسوج، دانشگاه آزاد اسلامی، یاسوج، ایران[1]

دانشیار گروه مهندسی برق، واحد یاسوج، دانشگاه آزاد اسلامی، یاسوج، ایران[2]

دانشیار گروه مهندسی کامپیوتر، واحد یاسوج، دانشگاه آزاد اسلامی، یاسوج، ایران[3، 4، 5]

چکیده

مسائل بهینه‌سازی در بیشتر موارد با فرض ثابت‌بودن شرایط محیطی تعریف می‌شوند. بیشتر مسائل موجود در جهان واقعی محیط‌های به‌طور مداوم در حال تغییرند؛ بنابراین ما به الگوریتم‌های بهینه‌سازی نیاز داریم که بتوانند مسائل را در محیط‌های پویا به‌خوبی حل کنند. مسائل بهینه‌سازی پویا مسائلی هستند که در طول زمان دستخوش تغییر می‌شوند. چنین محیط‌هایی ویژگی‌هایی نظیر نبود قطعیت، تغییرات زمانی و پیچیدگی‌های ساختاری را به همراه دارند که فرایند بهینه‌سازی را به چالشی جدی تبدیل می‌کند. در مواجهه با این چالش‌ها، الگوریتم‌های تکاملی از مؤثرترین روش‌ها برای حل مسائل بهینه‌سازی پویا (DOPs) مطرح شده‌اند. الگوریتم رقابت استعماری (ICA) که بر مبنای هوش ازدحامی و رقابت میان کشورهای استعمارگر طراحی شده‌است به‌دلیل توانایی در حل مسائل بهینه‌سازی ایستا مورد توجه ویژه قرار گرفته است؛ بااین‌حال، این الگوریتم در محیط‌های پویا عملکرد ضعیفی از خود نشان می‌دهد؛ زیرا فاقد مکانیسم‌هایی برای حفظ تنوع، تطبیق سریع با تغییرات محیطی و پیگیری نقاط بهینه جدید است. در این پژوهش، یک نسخهٔ بهبودیافته از ICA با هدف غلبه بر محدودیت‌های مذکور ارائه شده‌است. در طراحی این الگوریتم، از ترکیب سازوکار حافظه و استراتژی خوشه‌بندی استفاده شده‌است. سازوکار حافظهٔ اطلاعات نقاط بهینه گذشته را ذخیره کرده و در شرایط مناسب از این اطلاعات برای تسریع فرایند بهینه‌سازی استفاده می‌کند و استراتژی خوشه‌بندی k-means، به حفظ تنوع جمعیت کمک می‌کند و از انباشت راه‌حل‌ها در نواحی خاص جلوگیری می‌کند. این دو مؤلفه با یکدیگر، عملکرد الگوریتم در محیط‌های پویا را بهبود می‌بخشند. الگوریتم پیشنهادی در کنار الگوریتم‌های پیشرفته‌ای نظیر (FTmPSO(TMO، RAmQSO-s4، RmNAFSA-s4، TFTmPSO، RFTmPSO، mQSO10 (5+5q)، FMSO، CellularPSO، Multi-SwarmPSO، mCPSO، *AmQSO، FTMPSO، almPSO، و CDEPSA مورد ارزیابی قرار گرفت. نتایج تجربی نشان داد که الگوریتم پیشنهادی توانسته است، در زمینه‌هایی نظیر سرعت هم‌گرایی، تطبیق‌پذیری به تغییرات محیطی و حفظ تنوع جمعیت، عملکردی برتر از سایر الگوریتم‌ها ارائه کند. از ویژگی‌های کلیدی الگوریتم پیشنهادی، توانایی در حفظ نقاط بهینه حتی پس از تغییر محیط و مقیاس‌پذیری آن در مواجهه با مسائل بهینه‌سازی پویا با ابعاد بزرگ و پیچیدگی‌های بالا است. استفاده از خوشه‌بندی k-means باعث شده‌است که الگوریتم در مواجهه با تغییرات پیچیدهٔ محیطی از تمرکز بیش‌ازحد بر روی نقاط خاص جلوگیری کرده و تنوع جمعیت را به شکل مؤثری حفظ کند. این امر نشان می‌دهد که الگوریتم پیشنهادی نه تنها در محیط‌های آزمایشگاهی بلکه در کاربردهای واقعی با تغییرات سریع و پویا نیز کارآمد خواهد بود.

واژگان کلیدی: بهینه‌سازی پویا، محیط‌های پویا، حافظه، الگوریتم رقابت استعماری، خوشه‌بندی، محک قله‌های متحرک.

Solving dynamic optimization problems with an Improved Imperialis Competition Algorithm

Mahdi Sadeghi Moghadam[1*], Samad Nejatian[2], Hamid Parvin[3], Karamullah Bagheri Fard[4], Hadi Yagoubian[5]

PhD Student, Department of Computer Engineering, Yasouj Branch, Islamic Azad University, Yasouj, Iran[1*]

* نویسندهٔ عهده‌دار مکاتبات

* Corresponding author

• تاریخ ارسال مقاله: ۱۴۰۲/۶/۱۰ • تاریخ پذیرش: ۱۴۰۴/۴/۳۰ • تاریخ انتشار: ۱۴۰۴/۶/۲۲ • نوع مطالعه: پژوهشی

سال ۱۴۰۴ شمارهٔ ۲ پیاپی ۶۳

Associate Professor, Department of Electrical Engineering, Yasouj Branch, Islamic Azad University, Yasouj, Iran[2]
Associate Professor, Department of Computer Engineering, Yasouj Branch, Islamic Azad University, Yasouj, Iran[3, 4, 5]

Abstract

Optimization issues are often defined statically, assuming constant environmental conditions. However, in many real-world scenarios, problem environments are dynamic and continuously changing. Thus we need optimization algorithms that could solves those issues in dynamic environments as well. Dynamic optimization problems are change(s) that may occur through the time. Such environments are characterized by uncertainty, temporal changes, and structural complexities, which makes the optimization process a significant challenge. In addressing these challenges, evolutionary algorithms have emerged as one of the most effective approaches for solving dynamic optimization problems (DOPs). Among these algorithms, the Imperialist Competitive Algorithm (ICA), designed based on swarm intelligence and competition among imperialist countries, has garnered considerable attention due to its capability in solving static optimization issues. In this research, Imperialist Competitive Algorithms, inspired by the historical and political processes of colonization and assimilation, have been known as one of the efficient evolutionary algorithms. These algorithms face numerous challenges when dealing with dynamic problems , including reduced population diversity, performance degradation in conditions of rapid environmental changes, and limitations in optimal convergence. These cases indicate the need to develop improved and more adaptable versions of these algorithms. Using concepts such as memory, population clustering, and repulsion mechanisms, this algorithm has been able to maintain population diversity at all stages while increasing the speed of convergence in the face of environmental changes. The key feature of the proposed algorithm is the use of memory to store previous optimal solutions, a clustering mechanism to manage population diversity, and repulsion to prevent unnecessary accumulation of solutions in specific regions. Nevertheless, ICA exhibits poor performance in dynamic environments because it lacks mechanisms to maintain diversity, quick adaptation to environmental changes, and new optima track. This study presents an improved version of ICA aimed at overcoming these limitations. The proposed algorithm incorporates a combination of a memory mechanism and a clustering strategy to enhance its adaptability to environmental changes and preserve diversity within the population. The memory mechanism stores information about previous optima and utilizes it under appropriate conditions to accelerate the optimization process.for clustering method is used for clustering. Clustering in the proposed method ensures that diversity is maintained for the population during the execution of the algorithm. In this study, our goal is to solve problems that change the environment in a global way. That means, the fitness of all points in the environment changes. By testing just one point in the environment and comparing the fitness obtained with its previously stored value, we can detect a change in the environment. On the other hand, the clustering strategy, particularly the k-means technique, to maintain maintain population diversity and prevents the convergence of solutions to specific regions. Together, these two components create a balance between exploration and exploitation, thereby improving the algorithm's performance in dynamic environments. To evaluate the performance of the proposed algorithm, the Moving Peaks Benchmark (MPB) was used as a standard metric. Due to its capability to simulate complex and diverse changes in dynamic environments—particularly in Branke's second scenario—MPB is one of the most recognized tools for assessing the performance of dynamic optimization algorithms. The proposed algorithm was evaluated alongside advanced algorithms such as FTmPSO (TMO), RAmQSO-s4, RmNAFSA-s4, TFTmPSO, RFTmPSO, mQSO10 (5+5q), FMSO, CellularPSO, Multi-SwarmPSO, mCPSO, AmQSO*, FTMPSO, almPSO, and CDEPSA. Experimental results demonstrated that the proposed algorithm outperformed other methods in areas such as convergence speed, adaptability to environmental changes, and population diversity preservation. A key feature of the proposed algorithm is its ability to retain identified optima even after environmental changes. Additionally, the use of the k-means clustering technique has ensured that the algorithm effectively avoids excessive focus on specific regions and maintains population diversity while facing complex environmental changes. Another advantage of this algorithm is its scalability in handling dynamic optimization issues with high dimensions and complexities. These findings indicate that the proposed algorithm is not only effective in laboratory settings but also suitable for real-world applications with fast and dynamic changes.

Keywords: **Dynamic Optimization, Dynamic Environments, Memory, Imperialist Competition Algorithm, Clustering, Moving Peaks Benchmark.**

۱-مقدمه

موضوع بهینه‌سازی در علوم مختلف یکی از مهم‌ترین ابزارها برای حل مسائل پیچیده است. هدف اصلی این مسائل یافتن مجموعه‌ای از مقادیر بهینه برای متغیرهای یک مسئله است که مقدار تابع هدف را بیشینه یا کمینه سازد. این مسائل درکل به دو دسته تقسیم می‌شوند: مسائل ایستا و مسائل پویا.

در مسائل ایستا، شرایط مسئله و قیود آن در طول زمان ثابت باقی می‌مانند، به همین دلیل یافتن راه حل بهینه به نسبت ساده‌تر است، اما در بسیاری از مسائل واقعی، شرایط مسئله دائم در حال تغییر است. این تغییرات ممکن است، شامل تغییر در متغیرها، قیود یا حتی تابع هدف باشد. چنین محیط‌های پویایی چالش‌های متعددی را برای الگوریتم‌های بهینه‌سازی به همراه دارند و نیازمند روش‌هایی هستند که بتوانند مداوم با تغییرات جدید سازگار شوند[۱].

در محیط‌های پویا، بهینه‌های جهانی ثابت نیستند و با گذر زمان تغییر می‌کنند؛ ازاین‌رو، الگوریتم‌های بهینه‌سازی باید نه‌تنها قادر به یافتن بهینه‌ها باشند، بلکه توانایی دنبال‌کردن این تغییرات را نیز داشته باشند؛ همچنین در بسیاری از موارد لازم است، الگوریتم‌ها بتوانند چندین راه‌حل نزدیک بهینه را هم‌زمان شناسایی و ذخیره کنند. این الزامات مستلزم استفاده از الگوریتم‌هایی است که بتوانند هم تنوع جمعیت را حفظ کنند و هم از حافظه برای بهره‌برداری از اطلاعات پیشین استفاده کنند[۲].

یکی از چالش‌های اصلی در مسائل بهینه‌سازی پویا، جلوگیری از هم‌گرایی زودهنگام است. این مشکل زمانی رخ می‌دهد که الگوریتم به یک نقطهٔ بهینهٔ محلی هم‌گرا می‌شود و توانایی کاوش بیشتر در فضای مسئله را از دست می‌دهد. این مسئله، به‌ویژه در محیط‌هایی که تغییرات سریع و مداوم دارند، عملکرد الگوریتم را به‌شدت محدود می‌کند؛ برای رفع این مشکل، نیاز به الگوریتم‌هایی است که بتوانند تعادلی میان کاوش در فضای جدید و بهره‌برداری از اطلاعات پیشین برقرار کنند[۳].

برای حل مسائل بهینه‌سازی پویا، الگوریتم‌های الهام‌گرفته از طبیعت نظیر روش‌های تکاملی و الگوریتم‌های مبتنی بر هوش جمعی، به‌دلیل کارایی بالا در مواجهه با پیچیدگی‌های محیط‌های پویا مورد توجه قرار گرفته‌اند. این الگوریتم‌ها قادر به تطبیق با تغییرات محیطی هستند و می‌توانند با استفاده از جمعیت متغیر خود، بهینه‌های جدید را شناسایی کنند[۴]؛ بااین‌حال، الگوریتم‌های تکاملی در محیط‌های پویا با محدودیت‌هایی مواجه‌اند؛ از جمله از دست‌دادن تنوع جمعیت پس از هم‌گرایی به یک بهینه و نداشتن توانایی در مدیریت تغییرات ناگهانی محیط[۵].

یکی از الگوریتم‌های پرکاربرد در بهینه‌سازی، الگوریتم رقابت استعماری است که بر مبنای ایدهٔ رقابت میان کشورهای استعمارگر و مستعمره‌ها طراحی شده‌است؛ این الگوریتم به‌دلیل سرعت هم‌گرایی بالا، انعطاف‌پذیری و قابلیت تطبیق با شرایط مختلف، به یکی از ابزارهای محبوب برای حل مسائل بهینه‌سازی تبدیل شده‌است؛ بااین‌حال، این الگوریتم به‌طورخاص برای مسائل ایستا طراحی شده و در محیط‌های پویا عملکرد ضعیفی دارد؛ زیرا قادر به حفظ تنوع جمعیت یا مدیریت حافظهٔ اطلاعات پیشین نیست[۶].

در این پژوهش، یک نسخهٔ بهبودیافته از الگوریتم رقابت استعماری با هدف رفع محدودیت‌های آن در محیط‌های پویا ارائه شده‌است. نوآوری اصلی این پژوهش در استفاده از سازوکار حافظه و خوشه‌بندی برای افزایش توانایی الگوریتم در حفظ تنوع و تطبیق با تغییرات محیطی نهفته است. سازوکار حافظه امکان ذخیره و استفاده مجدد از اطلاعات بهینه‌های پیشین را فراهم می‌کند و خوشه‌بندی کمک می‌کند جمعیت در فضای مسئله یک‌نواخت توزیع شود. این رویکرد باعث جلوگیری از هم‌گرایی زودهنگام و افزایش دقت الگوریتم در یافتن بهینه‌های جدید می‌شود[۷]. علاوه‌براین، برای افزایش سرعت هم‌گرایی از اصول سامانه‌های آشوبی استفاده شده‌است؛ این سامانه‌ها توانایی پیش‌بینی دقیق‌تری نسبت به روش‌های تصادفی دارند و می‌توانند عملکرد الگوریتم را در محیط‌های پویا بهبود بخشند. روش پیشنهادی قادر است سه چالش اصلی در مسائل بهینه‌سازی پویا را برطرف کند:

۱. شناسایی تغییرات محیطی با سرعت و دقت بالا؛
۲. حفظ تنوع جمعیت برای جلوگیری از هم‌گرایی زودهنگام؛
۳. استفادهٔ بهینه از اطلاعات پیشین برای تطبیق سریع با تغییرات جدید[۸].

این مقاله در پنج بخش ارائه شده‌است: در بخش اول، مقدمه‌ای بر مسائل بهینه‌سازی پویا و اهمیت آن ارائه شده‌است. در بخش دوم، پیشینهٔ پژوهشی و روش‌های موجود برای حل این مسائل مرور می‌شوند. بخش سوم، جزئیات روش پیشنهادی شامل سازوکار حافظه و خوشه‌بندی را شرح می‌دهد. در بخش چهارم، نتایج تجربی حاصل از اجرای روش پیشنهادی تحلیل می‌شود. در بخش پنجم، جمع‌بندی پژوهش و پیشنهاداتی برای پژوهش‌های آینده ارائه می‌شود.

۲- ادبیات پژوهش و مروری بر کارهای پیشین

ادبیات مربوط به پژوهش در ابتدای این بخش به تفصیل شرح داده شده‌است. در بخش بعدی، آثار مرتبط با مطالعه شرح داده شده و سعی شده‌است تا روش‌های مشابه از جنبه‌های مختلف مورد بررسی قرار گیرد.

۲-۱- مرور ادبیات

این بخش مسائل بهینه‌سازی پویا (DOPs) را که در طول زمان بدون توجه به اقدامات بهینه‌سازی یا الگوریتم‌های یادگیری تغییر می‌کنند، شرح می‌دهد؛ همچنین این بخش ابتدا یک تعریف اساسی از مسائل پویا ارائه می‌دهد؛ سپس چالش‌های مسائل پویا را که آن‌ها را از مسائل ایستا جدا می‌کند، مورد بحث قرار می‌دهد و اصطلاحاتی را که در سراسر این مقاله استفاده خواهد شد، تعریف می‌کند. از آنجا که ICA به‌عنوان الگوریتم پایه در روش پیشنهادی در این مقاله استفاده شده‌است، ICA به‌تفصیل شرح داده شده‌است. در ادامه یک طبقه‌بندی از مسائل بهینه‌سازی پویا در محیط‌های پویا ارائه می‌شود.

۲-۱-۱- مسائل بهینه‌سازی پویا

بهینه‌سازی قلب بسیاری از فرایندهایی است که در طبیعت اتفاق می‌افتد؛ علاوه‌براین، نقش مهمی در حوزه‌های تجاری و مهندسی ایفا می‌کند. به‌منظور حل مسائل بهینه‌سازی، حداقل یک تابع هدف بر حسب معمول بر اساس پارامترهای مسئله طراحی می‌شود؛ سپس، هدف پارامترهای ورودی است که تابع(های) هدف را تحت محدودیت‌های لازم با استفاده از روش‌های ریاضی یا هوشمند بهینه می‌کند. هنگامی‌که مسائل بهینه‌سازی بسیار پیچیده‌اند، استفاده از روش‌های ریاضی بسیار دشوار یا حتی غیرممکن می‌شود. برای چنین مسائلی می‌توان از روش‌های هوشمند برگرفته از طبیعت استفاده کرد. در بیشتر مسائل بهینه‌سازی دنیای واقعی، تابع هدف یا محدودیت‌ها می‌توانند در طول زمان تغییر کنند؛ بنابراین، بهینه‌سازی این مسائل ممکن است منجر به تغییر نقاط بهینهٔ این مسائل در طول زمان شود. اگر هر رویداد نامطمئنی در فرایند بهینه‌سازی در نظر گرفته شود، مسئله بهینه‌سازی پویا نامیده می‌شود[۹-۱۲]. در مسائل بهینه‌سازی پویا، اطلاعات در طول زمان منتشر می‌شود، ممکن است رویدادهای نامطمئن رخ دهد یا الزامات مشکل ممکن است در طول زمان تغییر کند. یکی از راهکارهای بهبود بهینه‌سازی و یادگیری در محیط‌های پویا، استفاده از اطلاعات گذشته است؛ پیداکردن راه‌حل‌های امیدوارکننده در یک محیط جدید با استفاده از راه‌حل‌های محیط‌های پیشین، بیشتر آسان‌تر است. یک راه متداول برای ذخیره و بازیابی اطلاعات گذشته استفاده از حافظه است که راه‌حل‌ها به‌صورت دوره‌ای ذخیره می‌شوند و درصورت تغییر محیط می‌توان آن‌ها را بازیابی و اصلاح کرد[۱۲]. حافظه می‌تواند به جست‌وجوی پاسخ‌های سریع و کارآمد به تغییرات در یک مسئله پویا کمک کند. حافظه‌های استاندارد، با وجود نقاط قوتشان، نقاط ضعف بسیاری دارند که اثربخشی آن‌ها را محدود می‌کند[۷،۱۰،۱۱ و۱۲]. DOP ها به طور معمول به‌صورت زیر نشان داده می‌شوند[۱۳]:

$$DOP = \begin{cases} Optimize\ F(X,\Psi^t) \\ s.t.\ X \in F(\Psi^t) \subseteq S, t \in T \end{cases} \quad (۱)$$

Ψ پارامترهای محیطی است که در طول زمان با دوره‌های ثابت تغییر می‌کنند و t شاخص زمانی با $t \in [0,T]$ است که T تعداد محیط‌هاست؛ همچنین $S \in R^n$ فضای جست‌وجو، $F: S \times T \to R$ تابع تناسب است که مقادیر عددی $F(X,\Psi^t) \in R$ را به هر راه حل ممکن $(X \in S)$ در زمان t و $F(\Psi^t)$ مجموع راه‌حل‌های موجود $X \in F(t) \subseteq S$ در زمان t است؛ درنتیجه، برای یک DOP با حالت‌های محیطی T، دنباله‌ای از محیط‌های ایستای T وجود دارد[۱۳]:

$$F(X) = \{f(X,\Psi^1), f(X,\Psi^2), \dots, f(X,\Psi^T)\} \quad (۲)$$

که در آن Ψ^i پارامترهای i-امین محیط را نشان می‌دهد.

۲-۲-طبقه‌بندی محیط‌های پویا

اگرچه همهٔ مسائل پویا شامل تغییرات مکرر در چشم‌انداز شایستگی است، مسائل پویا ممکن است از جهات مختلفی متفاوت باشد. برخی از مسائل ممکن است، تغییرات تصادفی و پیوسته‌ای داشته باشند که بیشتر اتفاق می‌افتد؛ درحالی‌که برخی دیگر ممکن است، تغییرات شدید و نادری داشته باشند که تا حدی قابل پیش‌بینی باشد. در این بخش برخی از راه‌هایی که ممکن است، یک مسئله در محیط‌های پویا طبقه‌بندی شود، فهرست می‌شود که این طبقه‌بندی مشابه [۱۴،۱۵] است.

فرکانس: در برخی از مسائل دنیای واقعی، تغییرات در محیط ممکن است با سرعت کم و در برخی مسائل دیگر و در برخی موارد با سرعت زیاد رخ دهد. مهم‌ترین جنبهٔ فرکانس تغییر این است که یک الگوریتم یادگیری یا بهینه‌سازی چه مدت زمان باید برای یافتن راه‌حل هم پیش از تأثیر بر عملکرد و هم پیش از اینکه تغییر دیگری رخ دهد، پیدا کند. فرکانس تغییرات نشان می‌دهد که

محیط پس از چند بار ارزیابی فضای مسئله تغییر می‌کند[۱۴].

شدت: برخی از مسائل دنیای واقعی شدت تغییرات جزئی دارند و ردیابی این تغییرات تا حدودی آسان به نظر می‌رسد، اما برخی مسائل شدت تغییرات بالایی دارند و این تغییرات تأثیر زیادی بر چشم‌انداز شایستگی دارند[۱۴].

پیش‌بینی‌پذیری: تغییرات در برخی مسائل از الگوی خاصی پیروی می‌کنند؛ در برخی دیگر، تغییرات به طور کامل تصادفی‌اند. برخی تغییرات در فضای مسئله کوچک هستند تا الگوریتم بتواند این مسائل را پیش‌بینی کند، اما اگر تغییرات محیطی زیادی وجود داشته باشد، پیش‌بینی این مسائل برای الگوریتم مشکل خواهد بود[۱۴].

قابلیت تشخیص: در برخی مسائل لحظهٔ تغییر در محیط به‌راحتی قابل تشخیص است؛ بنابراین الگوریتم دقیق می‌داند چه زمانی تغییر در چشم‌انداز تناسب داده است؛ در برخی دیگر، ممکن است مدتی طول بکشد تا متوجه شوید که محیط تغییر کرده‌است. تشخیص لحظهٔ تغییر در محیط می‌تواند تأثیر زیادی بر نحوهٔ حل یک مسئله داشته باشد[۱۴].

۲-۳-چالش‌های محیط‌های پویا

در محیط‌های پویا، تغییرات غیر قابل پیش‌بینی در محیط چالش‌هایی را برای الگوریتم‌های بهینه‌سازی ایجاد می‌کند که عبارت‌اند از:

تشخیص لحظهٔ تغییر: از آنجا که تغییرات محیطی باعث نامعتبرشدن مقادیر شایستگی افراد جمعیت می‌شوند، لازم است پس از هر تغییر، شایستگی همهٔ افراد دوباره محاسبه و به‌روز شود. برای تشخیص تغییر، می‌توان شایستگی یک فرد را دوباره ارزیابی کرد؛ اگر با مقدار ثبت‌شدهٔ پیشین متفاوت بود، به معنای تغییر محیط است.

حفظ تنوع: هنگامی که افراد جمعیت به یک نقطه هم‌گرا می‌شوند، تنوع در جمعیت کاهش می‌یابد. این کاهش تنوع، ردیابی سریع نقاط بهینهٔ جدید را پس از تغییر محیط دشوار کرده و سرعت هم‌گرایی الگوریتم را به‌شدت کاهش می‌دهد.

قله‌های بدون پوشش: اگر تعداد قله‌های بهینه (نقاط بهینه) از تعداد افراد جمعیت بیشتر باشد، برخی قله‌ها ممکن است پوشش داده نشوند. افزایش بیش‌ازحد جمعیت برای پوشش همهٔ قله‌ها نیز هزینهٔ محاسباتی را افزایش و سرعت الگوریتم را کاهش می‌دهد. راه حل ضد هم‌گرایی پیشنهاد شده در [۱۵] با توزیع مجدد بدترین دسته در فضای مسئله، این چالش را برطرف می‌کند.

مصالحه بین اکتشاف و بهره‌وری: اکتشاف به معنای جست‌وجوی نقاط جدید در فضای جست‌وجو برای جلوگیری از گیرافتادن در بهینه‌های محلی است. بهره‌وری نیز به بهبود راه‌حل‌های موجود در مناطق شناسایی‌شده اشاره دارد. یافتن تعادل مناسب بین این دو فرایند برای کارایی و دقت یک الگوریتم بهینه‌سازی موفق، بسیار حیاتی و چالش‌برانگیز است.

۲-۴-الگوریتم رقابت استعماری

شکل (۱) شبه‌کد الگوریتم رقابت استعماری[1] (ICA) [۶] را نشان می‌دهد. همانند دیگر الگوریتم‌های تکاملی، این الگوریتم، نیز با تعدادی جمعیت اولیهٔ تصادفی که هر کدام از آن‌ها یک «کشور» نامیده می‌شوند؛ شروع می‌شود. تعدادی از بهترین عناصر جمعیت (معادل نخبه‌ها در الگوریتم ژنتیک) به‌عنوان امپرطوری[2] انتخاب می‌شوند؛ باقیماندهٔ جمعیت نیز به‌عنوان مستعمره[3] در نظر گرفته می‌شوند. استعمارگران بسته به قدرت خود، این مستعمرات را با یک روند خاص که در ادامه می‌آید به سمت خود می‌کشند. قدرت کل هر امپراتوری، به هر دو بخش تشکیل‌دهندهٔ آن یعنی کشور سلطه‌جو (به‌عنوان هستهٔ مرکزی) و مستعمرات آن، بستگی دارد. در حالت ریاضی، این وابستگی با تعریف قدرت امپراتوری به صورت مجموع قدرت کشور امپراتوری، به اضافهٔ درصدی از میانگین قدرت مستعمرات آن مدل شده‌است. با شکل‌گیری امپراتوری‌های اولیه، رقابت استعماری میان آن‌ها شروع می‌شود. هر امپراتوری که نتواند در رقابت استعماری موفق عمل کرده و بر قدرت خود بیفزاید (و یا دست‌کم از کاهش نفوذ خود جلوگیری کند)، از صحنهٔ رقابت استعماری حذف خواهد شد. در جریان رقابت‌های استعماری، به‌تدریج بر قدرت امپراتوری‌های بزرگتر افزوده شده و امپراتوری‌های ضعیف‌تر حذف خواهند شد.

Step 1: ***Initialization.*** Define the optimization problem; Select some random points as new position of colonies; Initialize the empires.
Step 2: ***Colonies Movement.*** Move the colonies toward their relevant imperialist.
Step 3: ***Imperialist Updating.*** If the new colony has lower cost than that of imperialist, exchange the positions of that colony and the imperialist.
Step 4: ***Imperialistic Competition.*** Pick the weakest colony from the weakest empire and give it to the empire that has the most likelihood to possess it.
Step 5: ***Implementation.*** Eliminate the powerless empires.
Step 6: ***Terminating Criterion Control.*** Repeat Steps 2-5 until a terminating criterion is satisfied.

(شکل-۱): شبه‌کد الگوریتم رقابت استعماری[۶]
(Figure-1): pseudocode of the Imperialist Competitive Algorithm [6]

[1] Imperialist Competitive Algorithm
[2] Imperialist
[3] Colony

با گذشت زمان، مستعمرات از لحاظ قدرت به امپراتوری‌ها نزدیک‌تر خواهند شد و شاهد نوعی هم‌گرایی خواهیم بود. حد نهایی رقابت استعماری، زمانی است که یک امپراتوری واحد در دنیا داشته باشیم، با مستعمراتی که از لحاظ موقعیت، به خود کشور سلطه‌جو خیلی نزدیک‌اند. در ادامه مراحل الگوریتم رقابت استعماری تشریح می‌شوند.

۲-۴-۱-شکل‌دهی امپراتوری‌های اولیه

در بهینه‌سازی، هدف یافتن یک جواب بهینه بر حسب متغیرهای مسئله است. ما یک آرایه از متغیرهای مسئله را که باید بهینه شوند، ایجاد می‌کنیم. در الگوریتم ژنتیک این آرایه کروموزوم[1] نامیده می‌شود. در اینجا نیز این آرایه یک کشور نامیده می‌شود. در یک مسئلهٔ بهینه‌سازی N_{var} بعدی، یک کشور یک آرایهٔ $1 \times N_{var}$ است. این آرایه به‌صورت رابطهٔ زیر تعریف می‌شود[۶].

$$country = [p_1, p_2, p_3, ..., p_{N_{var}}] \quad (۳)$$

برای شروع الگوریتم، تعداد $N_{country}$ کشور اولیه را ایجاد می‌کنیم. N_{imp} تا بهترین اعضای این جمعیت (کشورهای دارای کمترین مقدار تابع هزینه) را به‌عنوان امپراتوری انتخاب کنیم. باقیماندهٔ N_{col} تا از کشورها، مستعمراتی را تشکیل می‌دهند که هرکدام به یک امپراتوری تعلق دارند. برای تقسیم مستعمرات اولیه بین امپراتوری‌ها، به هر امپراتوری، تعدادی از مستعمرات را که این تعداد متناسب با قدرت آن است، می‌دهیم. برای انجام این کار، با داشتن هزینهٔ همهٔ امپریالیست‌ها، هزینهٔ نرمال آن‌ها را به صورت رابطهٔ (۴) در نظر می‌گیریم[۶].

$$C_n = \max_i \{c_i\} - c_n \quad (۴)$$

که در آن c_n، هزینهٔ امپراتوری nام، $\max_i \{c_i\}$ بیشترین هزینه میان امپراتوری‌ها و C_n، هزینهٔ نرمال‌شدهٔ این امپراتوری است. هر امپراتوری که دارای هزینهٔ بیشتری باشد (امپراتوری ضعیف‌تری باشد)، دارای هزینهٔ نرمالیزهٔ کمتری خواهد بود. با داشتن هزینهٔ نرمالیزه، قدرت نسبی نرمالیزهٔ هر امپراتوری، به‌صورت رابطهٔ (۵) محاسبه‌شده و بر مبنای آن، کشورهای مستعمره بین امپراتوری‌ها تقسیم می‌شوند[۶].

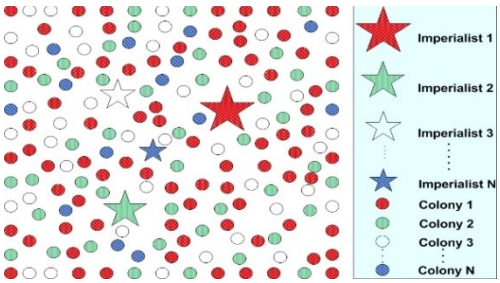
$$p_n = \left| \frac{C_n}{\sum_{i=1}^{N_{imp}} C_i} \right| \quad (۵)$$

از یک دید دیگر، قدرت نرمالیزه‌شدهٔ یک امپراتوری، نسبت مستعمراتی است که به‌وسیلهٔ آن امپراتوری اداره می‌شود؛ بنابراین تعداد اولیهٔ مستعمرات یک امپراتوری برابر خواهد بود با [۶]:

$$N.C._n = round\{p_n . (N_{col})\} \quad (۶)$$

که در آن $N.C._n$، تعداد اولیهٔ مستعمرات یک امپراتوری و N_{col} نیز تعداد کل کشورهای مستعمرهٔ موجود در جمعیت کشورهای اولیه است. $round$ نیز تابعی است که نزدیک‌ترین عدد صحیح به یک عدد اعشاری را می‌دهد. با درنظرگرفتن $N.C.$ برای هر امپراتوری، به این تعداد، از کشورهای مستعمرهٔ اولیه به‌صورت تصادفی انتخاب کرده و به امپراتوری nام می‌دهیم. با داشتن حالت اولیهٔ تمام امپراتوری‌ها، الگوریتم رقابت استعماری شروع می‌شود. روند تکامل در یک حلقه قرار دارد که تا برآورده‌شدن یک شرط توقف ادامه می‌یابد.

شکل (۲) چگونگی شکل‌گیری امپراتوری‌های اولیه را نشان می‌دهد؛ همان‌گونه که در این شکل نشان داده شده‌است امپراتوری‌های بزرگتر تعداد بیشتری مستعمره دارند. در این شکل امپریالست شمارهٔ یک قوی‌ترین امپراتوری را ایجاد کرده‌است و بیش‌ترین تعداد مستعمرات را دارد.



(شکل-۲): چگونگی شکل‌گیری امپراتوری‌های اولیه[۶]

(Figure-2): Generation of initial empires[6]

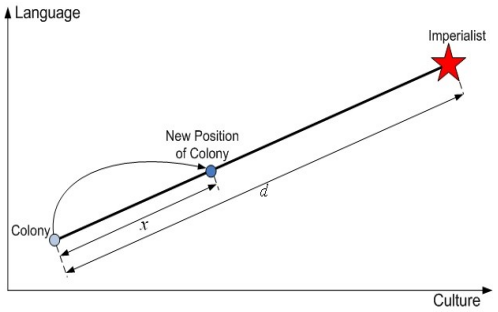
۲-۴-۲-مدل‌سازی سیاست جذب: حرکت مستعمره‌ها به سمت امپراتوری

سیاست هم‌گون‌سازی[2] (جذب) با هدف تحلیل فرهنگ و ساختار اجتماعی مستعمرات در فرهنگ حکومت مرکزی انجام می‌گرفت. کشورهای استعمارگر برای افزایش نفوذ خود شروع به ایجاد عمران (ایجاد زیرساخت‌های حمل‌ونقل، تأسیس دانشگاه و دیگر موارد) کردند؛ برای مثال کشورهایی نظیر انگلیس و فرانسه با تعقیب سیاست همگون‌سازی در مستعمرات خود در فکر ایجاد انگلیس نو[3] و فرانسهٔ نو[4] در مستعمرات خویش بودند. با درنظرگرفتن شیوهٔ نمایش یک کشور در حل مسئلهٔ بهینه‌سازی، درحقیقت این حکومت مرکزی با اعمال سیاست جذب سعی داشت تا کشور مستعمره را در راستای ابعاد مختلف اجتماعی سیاسی به خود نزدیک

[1] chromosome

[2] Assimilation
[3] New England
[4] New France

کند. این بخش از فرایند استعمار در الگوریتم بهینه‌سازی، به‌صورت حرکت مستعمرات به سمت کشور امپراتوری مدل شده‌است. شکل (۳)، شمای کلی این حرکت را نشان می‌دهد. مطابق شکل (۳) کشور سلطه‌جو کشور مستعمره را در راستای محورهای فرهنگ و زبان به سمت خود جذب می‌کند؛ همان‌گونه که در این شکل نشان داده شده‌است، کشور مستعمره[1]، به اندازهٔ x واحد در جهت خط واصل مستعمره به استعمارگر[2]، حرکت کرده و به موقعیت جدید[3] کشانده می‌شود.

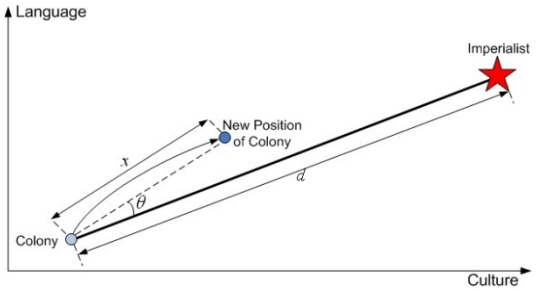


(شکل-۳): شمای کلی حرکت مستعمرات به سمت امپراتوری [۶]
(Figure-3): Movement of colonies toward their relevant imperialist [6].

در شکل (۳) فاصلهٔ میان استعمارگر و مستعمره با d نشان داده شده‌است. x نیز عددی تصادفی با توزیع یکنواخت (و یا هر توزیع مناسب دیگر) است؛ یعنی برای x داریم[۶].

$$x \sim U(0, \beta \times d) \qquad (۷)$$

که در آن β عددی بزرگ‌تر از یک و نزدیک به دو است. یک انتخاب مناسب می‌تواند $\beta = 2$ باشد. وجود ضریب $\beta > 1$ باعث می‌شود تا کشور مستعمره در حین حرکت به سمت کشور استعمارگر از جهت‌های مختلف به آن نزدیک شود.



(شکل-۴): حرکت واقعی مستعمرات به سمت سلطه‌جو [۶]
(Figure-4): Movement of colonies toward their relevant imperialist in a randomly deviated direction[6].

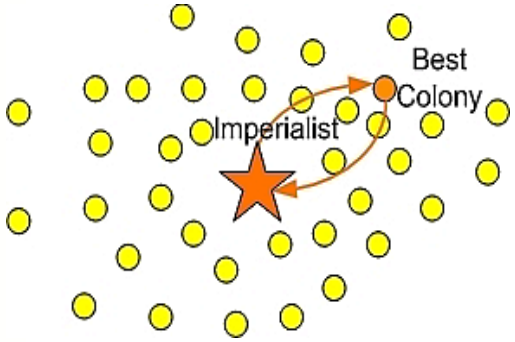
در سیاست جذب، وقایع تاریخی نشان می‌دهند که نتایج همیشه مطابق برنامهٔ کشورهای استعمارگر پیش نمی‌رفت و انحرافاتی در روند اجرا وجود داشت. در الگوریتم رقابت استعماری، این انحراف با اضافه‌کردن یک زاویهٔ تصادفی θ به مسیر حرکت مستعمرات به سمت کشور سلطه‌گر شبیه‌سازی می‌شود؛ به این ترتیب، مستعمره به جای حرکت مستقیم در جهت بردار واصل به کشور سلطه‌گر، با یک زاویهٔ تصادفی انحراف پیدا می‌کند که این زاویه به صورت تصادفی با توزیع یکنواخت تعیین می‌شود. شکل (۴) این انحراف در مسیر حرکت مستعمرات به سمت کشور سلطه‌گر را نشان می‌دهد.

$$\theta \sim U(-\gamma, \gamma) \qquad (۸)$$

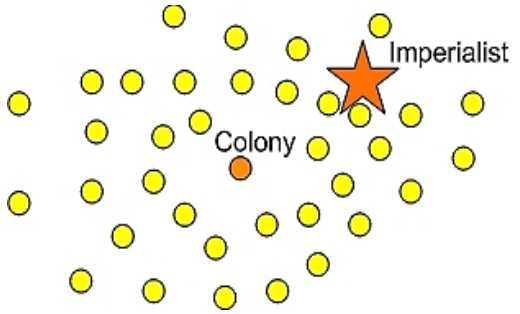
در رابطهٔ (۸) γ پارامتری دل‌خواه است که افزایش آن باعث افزایش جست‌وجوی اطراف امپراتوری شده و کاهش آن نیز باعث می‌شود تا مستعمرات تا حد ممکن، به بردار واصل مستعمره به استعمارگر نزدیک حرکت کنند.

۲-۴-۳-جابه‌جایی موقعیت مستعمره و امپراتوری

در فرایند جذب ممکن است، برخی از مستعمره‌ها به موقعیت‌هایی با هزینهٔ کمتر از کشور سلطه‌گر دست یابند. در این حالت، مستعمره و کشور سلطه‌گر موقعیت خود را با یکدیگر جابه‌جا می‌کنند. پس از این جابه‌جایی، کشور مستعمره به سلطه‌گر جدید تبدیل می‌شود و سیاست‌های جذب را بر سایر مستعمرات اعمال می‌کند. این جابه‌جایی به الگوریتم امکان می‌دهد تا به نقاط بهینه‌تر دست یابد و انعطاف‌پذیری بیشتری در فرایند بهینه‌سازی داشته باشد. شکل‌های (۵) و (۶) موقعیت‌های جابه‌جا شده و وضعیت نهایی امپراتوری را پس از تغییر موقعیت‌ها نمایش می‌دهند.



(شکل-۵): تغییر جای استعمارگر و مستعمره [۶]
(Figure-5): Exchanging the positions of a colony and the imperialist[6]



(شکل-۶): کل امپراتوری، پس از تغییر موقعیت‌ها [۶]
(Figure-6): The entire empire, after position change of [6]

[1] Colony
[2] Imperialist
[3] New Position of Colony

۲-۴-۴-قدرت کل یک امپراتوری

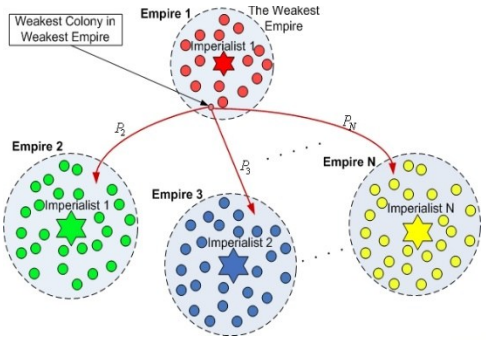
قدرت یک امپراتوری برابر است با قدرت کشور استعمارگر، به اضافهٔ درصدی از قدرت کل مستعمرات آن. بدین ترتیب برای هزینهٔ کل یک امپراتوری داریم [۶].

$$T.C_n = Cost(imperialist_n) + \xi\, mean\{Cost(colonies\ of\ empire_n)\} \quad (۹)$$

که در آن $T.C_n$ هزینهٔ کل امپراتوری nام و ξ عددی مثبت است که به‌طور معمول بین صفر و یک و نزدیک به صفر در نظر گرفته می‌شود. کوچک درنظرگرفتن ξ، باعث می‌شود که هزینهٔ کل یک امپراتوری، کمابیش برابر با هزینهٔ حکومت مرکزی آن (کشور سلطه‌جو) شود و افزایش ξ نیز باعث افزایش تأثیر میزان هزینهٔ مستعمرات یک امپراتوری در تعیین هزینهٔ کل آن می‌شود. در حالت نوعی $\xi = 0.05$ در بیشتر پیاده‌سازی به جواب‌های مطلوبی منجر شده‌است.

۲-۴-۵-رقابت استعماری

همان گونه که پیش‌تر نیز بیان شد، هر امپراتوری که نتواند بر قدرت خود بیفزاید و قدرت رقابت خود را از دست بدهد، در جریان رقابت‌های امپریالیستی حذف خواهد شد. این حذف‌شدن تدریجی صورت می‌پذیرد؛ بدین معنی که به مرور زمان امپراتوری‌های ضعیف مستعمرات خود را ازدست داده و امپراتوری‌های قوی‌تر این مستعمرات را تصاحب کرده و بر قدرت خویش می‌افزایند. برای مدل‌کردن این واقعیت فرض می‌کنیم که امپراتوری در حال حذف، ضعیف‌ترین امپراتوری موجود است؛ بدین ترتیب، در تکرار الگوریتم، یک یا چند تا از ضعیف‌ترین مستعمرات ضعیف‌ترین امپراتوری را برداشته و برای تصاحب این مستعمرات، رقابتی را میان کلیهٔ امپراتوری‌ها ایجاد می‌کنیم. مستعمرات مذکور، به طور حتم به وسیلهٔ قوی‌ترین امپراتوری تصاحب نخواهند شد، بلکه امپراتوری‌های قوی‌تر، احتمال تصاحب بیشتری دارند. شکل (۷) شمای کلی این بخش از الگوریتم رقابت استعماری را نشان می‌دهد.



(شکل-۷): شمای کلی رقابت استعماری [۶]

(Figure-7): Imperialistic competition[6]

در شکل (۷) امپراتوری شمارهٔ ۱ ضعیف‌ترین امپراتوری در نظر گرفته شده و یکی از مستعمرات آن در معرض رقابت امپریالیستی قرار گرفته‌است و امپراتوری‌های ۲ تا N برای تصاحب آن با هم رقابت می‌کنند. برای مدل‌سازی رقابت میان امپراتوری‌ها برای تصاحب این مستعمرات، ابتدا احتمال تصاحب هر امپراتوری (که متناسب با قدرت آن امپراتوری است)، را با درنظرگرفتن هزینهٔ کل امپراتوری، به ترتیب زیر محاسبه می‌کنیم؛ ابتدا از روی هزینهٔ کل امپراتوری، هزینهٔ کل نرمالیزه‌شدهٔ آن را تعیین می‌کنیم.

$$N.T.C._n = \max_i\{T.C._i\} - T.C._n \quad (۱۰)$$

در رابطهٔ (۱۰) $T.C._n$ ، هزینهٔ کل امپراتوری nام و $N.T.C._n$ نیز، هزینهٔ کل نرمالیزه‌شدهٔ آن امپراتوری است. هر امپراتوری که $T.C._n$ کمتری داشته باشد $N.T.C._n$ بیشتری خواهد داشت؛ در حقیقت $T.C._n$ معادل هزینهٔ کل یک امپراتوری و $N.T.C._n$ معادل قدرت کل آن است. امپراتوری با کمترین هزینه دارای بیشترین قدرت است. با داشتن هزینهٔ کل نرمالیزه‌شده، احتمال (قدرت) تصاحب مستعمرهٔ رقابت، توسط هر امپراتوری، به‌صورت رابطهٔ (۱۱) محاسبه می‌شود [۱۶].

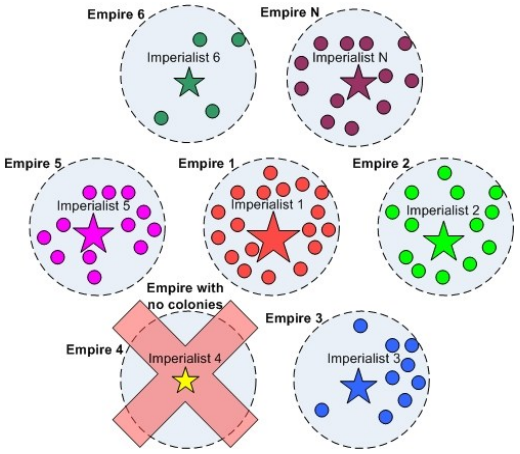
$$p_{p_n} = \left| \frac{N.T.C._n}{\sum_{i=1}^{N_{imp}} N.T.C._i} \right| \quad (۱۱)$$

با داشتن احتمال تصاحب هر امپراتوری، مکانیزمی همانند چرخهٔ رولت[1] در الگوریتم ژنتیک مورد نیاز است تا مستعمرهٔ مورد رقابت را با احتمال متناسب با قدرت امپراتوری‌ها در اختیار یکی از آن‌ها قرار دهد. با تصاحب مستعمره به وسیلهٔ یکی از امپراتوری‌ها، عملیات این مرحله از الگوریتم نیز به پایان می‌رسد.

۲-۴-۶-سقوط امپراتوری‌های ضعیف

همان‌گونه که بیان شد، در جریان رقابت‌های امپریالیستی، خواه ناخواه، امپراتوری‌های ضعیف به تدریج سقوط کرده و مستعمرات آن‌ها به‌دست امپراتوری‌های قوی‌تر می‌افتد. شروط متفاوتی را می‌توان برای سقوط یک امپراتوری در نظر گرفت. در الگوریتم رقابت استعماری، یک امپراتوری زمانی حذف‌شده تلقی می‌شود که مستعمرات خود را از دست داده باشد. شکل (۸) این مسئله را به‌خوبی نشان می‌دهد. در این شکل، امپراتوری شمارهٔ ۴ به‌علت ازدست‌دادن کلیهٔ مستعمرات خود دیگر قدرتی برای رقابت ندارد و باید از میان دیگر امپراتوری‌ها حذف شود.

[1] Roulette Wheel



(شکل-۸): سقوط امپراتوری ضعیف [۶]

(Figure-8): The fall of a weak empire [6]

۲-۴-۷-هم‌گرایی

الگوریتم رقابت استعماری تا برآورده‌شدن یک شرط هم‌گرایی، و یا تا اتمام تعداد کل تکرارها ادامه می‌یابد. پس از مدتی، همهٔ امپراتوری‌ها سقوط کرده و تنها یک امپراتوری باقی خواهد ماند و سایر کشورها تحت کنترل این امپراتوری واحد قرار می‌گیرند. در این دنیای ایده‌آل جدید، همهٔ مستعمرات، به‌وسیلهٔ یک امپراتوری واحد اداره می‌شوند و موقعیت‌ها و هزینه‌های مستعمرات، برابر با موقعیت و هزینهٔ کشور سلطه‌جو است. در این دنیای جدید، تفاوتی، نه‌تنها، میان مستعمرات، بلکه میان مستعمرات و کشور امپریالیست وجود ندارد؛ به عبارت دیگر، تمام کشورها، در عین حال، هم مستعمره و هم استعمارگرند؛ در چنین موقعیتی رقابت امپریالیستی به پایان رسیده و به‌عنوان یکی از شروط توقف الگوریتم متوقف می‌شود.

۲-۵-حافظه

ایدهٔ حفظ افراد نخبه (راه‌حل‌های خوب) در یک حافظهٔ صریح [۱۶ و ۱۷] برای الگوریتم به نظر جذاب است. بسیاری از مسائل مطرح‌شده در بهینه‌سازی و هوش مصنوعی ایستا هستند، داده‌ها یا اطلاعات دربارهٔ مسائل شناخته شده و قابل پیش‌بینی‌اند؛ اگر چه بیشتر مسائل دنیای واقعی پویا هستند اطلاعات دربارهٔ این مسائل در طول زمان تغییر می‌کنند و رویداد نامشخص ممکن است اتفاق بیفتد، با گذشت زمان ممکن است ملزومات مسئله تغییر کند. یک تکنولوژی برای بهبود بهینه‌سازی و یادگیری محیط‌های پویا استفاده از اطلاعات گذشته است که از راه‌حل‌های محیط‌های پیشین استفاده شود که ساده‌تر از پیداکردن راه‌حل‌های امیدبخش در یک محیط جدید است. یک راه معمول جهت حفظ اطلاعات گذشته استفاده از حافظه است که راه‌حل‌ها به‌صورت دوره‌ای ذخیره می‌شوند و زمانی که محیط تغییر می‌کند می‌توان آن‌ها را بازیابی کرد. حافظه می‌تواند به جست‌وجوی سریع کمک کند و در تغییرات مسائل پویا کارآمد باشد. انواع متعددی از حافظه وجود دارد که یک سیستم حافظهٔ استاندارد برای بهینه‌سازی پویا پدید آمده‌است. در این سیستم حافظهٔ استاندارد تعداد محدودی راه‌حل ذخیره شده و آن‌گاه در جست‌وجو به‌کار می‌رود تا درصورت تغییر محیط نیز فرایند جست‌وجو را به-سمت راه‌حل‌های خوب هدایت کند. در بسیاری از مسائل پویا حالت فعلی محیط بیشتر شبیه به حالات دیدهشدهٔ پیشین است. استفاده از اطلاعات گذشته ممکن است، به سیستم کمک کند تا با تغییرات بزرگ در محیط بهتر تطبیق پیدا کند و در طول زمان بهتر اجرا شود. یک راه برای حفظ و بهره‌برداری از اطلاعات گذشته استفاده از حافظه است؛ محلی که راه‌حل‌ها به‌صورت دوره‌ای در آن ذخیره می‌شود و در زمانی که محیط تغییر می‌کند می‌توان آن‌ها را بازیابی کرد. روش‌های بر پایهٔ حافظه برای بهینه‌سازی پویا ممکن است با توجه به چگونگی ذخیره‌سازی حافظه، به دو دسته حافظهٔ ضمنی و حافظهٔ صریح تقسیم شوند. حافظهٔ ضمنی اطلاعات گذشته را به‌عنوان بخشی از یک عضو ذخیره می‌کند، اما حافظهٔ صریح اطلاعات را مجزا از جمعیت، به‌طورمعمول به‌صورت یک دسته از راه‌حل‌های خوب پیشین ثبت می‌کند. روش حافظهٔ صریح به‌طور گسترده‌تر بررسی شده‌است و عملکرد بهتری نسبت به روش حافظهٔ ضمنی در مسایل پویا دارد[۱۷]. حافظه یک تعداد متناهی از داده‌ها را ذخیره می‌کند که حاوی اطلاعات تولیدشده به‌وسیلهٔ فرایند جست‌وجو است که ممکن است، پس از این که تغییرات در محیط انجام شود برای کمک به جست‌وجو مورد استفاده قرار گیرد. توابع حافظه به‌عنوان یک نسخهٔ پیچیده از نخبه‌گرایی هستند که راه‌حل‌های خوب در جمعیت صرف نظر از نتایج حاصل از جست‌وجو را حفظ می‌کنند. حافظه یک اندازهٔ ثابت دارد، هنگامی که با جست‌وجوی مبتنی بر جمعیت مورد استفاده قرار گیرد آن اندازه به‌طور کلی نسبت به اندازهٔ کل جمعیت کوچک است. در یک مسئلهٔ پویا، همهٔ افراد باید در هر مرحله از فرایند جست‌وجو ارزیابی شوند که آن دسته از افراد ذخیره‌شده در حافظه را نیز شامل می‌شود. اطلاعات ذخیره‌شده در حافظه را می‌توان به دو دسته تقسیم کرد: اطلاعات محیطی و اطلاعات کنترلی. اطلاعات محیطی کمک می‌کند تا تصویری از وضعیت فعلی مسئله به‌دست آید؛ در صورت تغییر مسئله، اطلاعات محیطی ممکن است برای تعیین شباهت بین محیط فعلی و محیطی که در حافظه ذخیره شده‌است استفاده شود. اطلاعات کنترلی

شامل اطلاعاتی است که در فرایند جست‌وجو استفاده می‌شود؛ برای مثال اطلاعات کنترلی ممکن است تنها یک راه‌حل ذخیره‌شده در قبل باشد که می‌تواند دوباره در جمعیت جاگذاری شود یا ممکن است حاوی یک مدل احتمالی از جمعیت بر اساس الگوریتم تکاملی باشد که می‌تواند برای دوباره مقداردهی اولیهٔ جمعیت بعد از یک تغییر استفاده شود.

۲-۵-۱-راه‌کارهای جایگزینی در حافظه

ذخیره‌سازی و نگهداری داده در حافظه، یکی از مهم‌ترین مسائل در کارهای پیشین به شمار می‌رفت؛ نخست اینکه می‌بایستی تصمیم گرفته می‌شد که چگونه داده‌های موجود در حافظه به‌روزرسانی شوند؛ دوم تعیین اینکه کدام داده به‌عنوان یک دادهٔ ورودی می‌توانست به سایر داده‌های موجود اضافه شود. با توجه به اینکه حافظه‌ها فضای ذخیره‌سازی محدودی دارند، درصورتی‌که یک شخص درخواست ذخیره‌سازی اطلاعاتی را داشته باشد می‌بایستی دادهٔ جدید با یکی از داده‌های موجود جایگزین شود. سازوکار انتخاب داده‌ای که باید در حافظهٔ اصلی تغییر کند (از میان داده‌های پیشین)، استراتژی جایگزینی نامیده می‌شود. بیشتر استراتژی‌های موجود به‌گونه‌ای طراحی شده‌اند که بتوانند فضای بیشتری از حافظه را نگهداری کنند؛ برای نمونه یکی از استراتژی‌های جایگزینی، دو تا از مدخل‌های نزدیک به هم را برای خروج از حافظه انتخاب می‌کند، بعد از آن نیز مقدار تناسب دو مدخل را با یکدیگر مورد مقایسه قرار می‌دهد و مدخلی را برای خروج در نظر می‌گیرد که مقدار تناسب پایین‌تری داشته باشد.

برای جایگزینی افراد جمعیت در حافظه، استراتژی‌های مختلفی ارائه شده‌اند که دو راه‌کار نخست و دوم در مرجع [۱۸ و ۱۹] آمده‌اند:

راه‌کار۱: در این راه‌کار دو فرد در حافظه با کم‌ترین فاصله (بیشترین شباهت) انتخاب شده و فردی که کارایی کمتری داشته باشد نامزد 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 بالعکس.

راه‌کار۲: اگر $fit(j) \times \frac{d_{ij}}{d_{max}} \leq fit(new)$ باشد درآن صورت فرد j-ام با بهترین فرد فعلی جایگزین می‌شود. که در این رابطه $fit(j)$ را کارایی فرد j-ام و d_{ij} فاصلهٔ بین دو فرد i و j، d_{max} بیشترین فاصلهٔ ممکن بین دو فرد i و j است.

راه‌کار۳: در این استراتژی به هر یک از افراد موجود در حافظه مشخصه‌ای به نام سن اختصاص داده می‌شود. افراد در مرحلهٔ نخست ورود به حافظه دارای سن برابر و سن همهٔ آن‌ها صفر است و در هر نسل یک واحد به مشخصهٔ سن آن‌ها اضافه می‌شود. اگر بیشترین سن را برای افراد حافظه برابر صد در نظر بگیریم در این صورت اگر سن افراد به صد برسد؛ یعنی آن افراد پیرترین افراد جمعیت حافظه‌اند و ایدهٔ مورد نظر بدین‌صورت است که اگر حافظه پر شود و نیاز به به‌روزرسانی و جایگزینی داشته باشد در این‌صورت جوان‌ترین افراد از حافظه حذف می‌شوند و افراد جدید جایگزین می‌شوند؛ بهطوری‌که گنجایش حافظه بیشتر نشود [۱۸ و ۱۹].

۲-۵-۲-بازیابی از حافظه

حافظه‌های ذخیره‌سازی به دو روش می‌توانند به وسیلهٔ جست‌وجوگرهای مبتنی بر جمعیت مورد بررسی قرار گیرند: در طول اجرا و یا پس از هر بار تغییرات. درصورتی‌که اطلاعات حافظه پس از هر بار تغییرات مورد بازیابی قرار گیرد، بدترین مقدار از m داده جمعیت با یک رونوشت از m داده در حافظهٔ اصلی جایگزین خواهد شد؛ درغیراین‌صورت اگر بازیابی در طول اجرا صورت پذیرد، آن‌گاه داده‌های حافظهٔ اصلی در هر بازه تولید می‌توانند مورد بازیابی قرار گیرند. در هر نسل جمعیت مربوط به r راه حل تولیدشده به‌وسیلهٔ نسل پیشین، با یک کپی جدید از m راه حل تولیدشدهٔ فعلی در حافظهٔ اصلی ترکیب شده و تشکیل جمعیت p را درون حافظه می‌دهند. الگوریتم‌های یادگیری که مبتنی بر جمعیت نیستند، زمانی اقدام به بازیابی یک راه‌حل می‌کنند که یک رویداد ویژه به وقوع پیوسته باشد؛ این رویداد می‌تواند یک تغییر در حافظه و یا تشابه بالای محیط جاری با یکی از داده‌های موجود در حافظه باشد. (برخلاف روش پیشین، این روش در هر گام نسبت به بازیابی راه‌حل‌ها اقدام نمی‌کند). حافظه از فرایند جست‌وجو به صورت جداگانه نگهداری می‌شود؛ به این صورت که تا زمانی که عمل‌گرهای جست‌وجو نسبت به تغییر داده‌های اضافه‌شده به جمعیت فعلی اقدام می‌کنند، مابقی داده‌های موجود در حافظهٔ اصلی تغییر نخواند داشت. در بیشتر حافظه‌های موجود، مقادیر منحصربه‌فرد بازیابی‌شده از حافظه کامل برابر مقادیر ذخیره‌سازی شده‌اند. (هم‌اکنون نیز نمونه‌هایی وجود دارند که پس از بازیابی جمعیت‌های داده‌ای، نسبت به اضافه‌کردن مقادیری به آن اقدام می‌کنند).

یکی از استراتژی‌های بازیابی از حافظه بدین صورت است که بهترین فرد از حافظه را مشخص کرده و آن را جایگزین بدترین فرد از جمعیت اصلی کنند [۱۸ و ۱۹].

۲-۶-خوشه‌بندی

خوشه‌بندی یکی از شاخه‌های یادگیری بی‌ناظر است که در طی آن نمونه‌ها به یک سری خوشه[1] افراز می‌شوند؛ به‌گونه‌ای که نمونه‌های موجود در هر خوشه از نظر شباهت به هم نزدیک‌اند و با نمونه‌های موجود در دیگر خوشه‌ها متفاوت‌اند. معیارهای زیادی برای اندازه‌گیری شباهت وجود دارد که یکی از این معیارها می‌تواند فاصلهٔ اقلیدسی[2] بین دو نمونه باشد. نمونه‌هایی که به هم شبیه‌اند فاصلهٔ اقلیدسی کمتری نسبت به هم دارند و در یک خوشه قرار می‌گیرند. به این نوع خوشه‌بندی، خوشه‌بندی مبتنی بر فاصله[3] می‌گویند. یکی از روش‌های خوشه‌بندی مبتنی بر فاصله، روش مبتنی بر مرکز است. مرکز هر خوشه را می‌توان از طریق میانگین داده‌های آن خوشه نمایش داد. در این روش مبتنی بر میانگین[4] داده‌های شبیه به مرکز خوشه به آن خوشه تعلق پیدا می‌کنند. در ادامه انواع روش‌های خوشه‌بندی حافظه آورده شده‌اند[۲۰].

۲-۶-۱-خوشه‌بندی اقلیدسی

در خوشه‌بندی اقلیدسی[5]، تنها مراکز خوشه‌ها محاسبه می‌شوند و فاصلهٔ اقلیدسی بین خوشه‌ها اندازه‌گیری می‌شود. روش خوشه‌بندی اقلیدسی سربار خیلی کمی دارد. در خوشه‌بندی اقلیدسی به‌عنوان بخشی از مدل می‌بایست میانگین نقاط در حافظه را محاسبه کرد[۲۱].

۲-۶-۲-خوشه‌بندی گاوسی[6]

نوع دوم از خوشه‌بندی برای حافظه‌ها، خوشه‌بندی گاوسی است که مدل‌های غنی‌تری برای استفادهٔ حافظه فراهم می‌کند. در این روش، هر دادهٔ حافظه یک مدل گاوسی از نقاط در آن محاسبه می‌کند[۲۲].

۲-۶-۳-خوشه‌بندی گاوسی افزایشی

در نسخهٔ خوشه‌بندی افزایشی مربوط به حافظه، نقاطی که به خوشه اضافه می‌شوند به‌عنوان (مانند) نقاطی هستند که در حافظه ذخیره می‌شوند. زمانی که یک نقطه به اندازهٔ کافی با خوشهٔ موجود که یک خوشهٔ جدید باید خلق شود متفاوت باشد، دو خوشهٔ موجود با هم ادغام می‌شوند؛ بنابراین تعدادی خوشه‌های یکسان باقی می‌مانند[۲۳].

[1] Cluster
[2] Euclidean distance
[3] Distance-based Clustering
[4] K-means
[5] Euclidean clustering
[6] Gaussian clustering

۲-۷-نظریهٔ آشوب

تئوری آشوب رفتار پویای سامانه‌هایی که به شرایط اولیه بسیار حساس‌اند را بررسی می‌کند. تغییرات کوچک در انتخاب‌های اولیه (مانند خطاهای گردکردن در محاسبات عددی) موجب تفاوت در نتایج سامانه‌های پویا می‌شود و به‌طورکلی پیش‌بینی‌های طولانی‌مدت غیر ممکن است[۲۱]. رفتار یک سامانهٔ آشوبی به‌طور معمول نه به‌طور کامل قطعی است و نه کامل تصادفی؛ یعنی نه همیشه رفتارش به طور کامل قابل پیش‌بینی است و نه این‌که به‌هیچ‌وجه نمی‌توان رفتارش را پیش‌بینی کرد[۲۱]. پدیده‌های تصادفی که هیچ علتی در آن پیدا نمی‌شود، اکنون به‌وسیلهٔ تئوری آشوب توجیه می‌شوند. یک سامانهٔ آشوب‌گونه می‌تواند پیش‌بینی دقیق‌تری برای آینده در مقایسه با یک سامانه تصادفی داشته باشد[۲۱]. سامانه‌های آشوبی سام‌های باز هستند؛ این بدین معنا است که سامانه‌هایی هستند که با محیط در حال تغییروتحول خود سازگارند و هم با محیط امروز تعادل برقرار می‌کنند و هم با محیط فردا[۲۱]. سیستم‌های آشوبی، بین دو تعادل، تعادل واسطی ایجاد می‌کنند. از یک تعادل به تعادل دیگر می‌رسند. نظریهٔ آشوب در مورد وابستگی بین اجزا در شرایط عدم تعادل بحث می‌کند و مشخص می‌کند وقتی که یک سیستم از یک حالت تعادلی خارج شد، چگونه به تعادل جدید وارد می‌شود. در واقع سیستمی که دارای خصوصیات آشوبی است، در حالت معمول دارای ثبات است. این سیستم در واکنش به تغییرات جدید، به تدریج از محدودهٔ نظم و ثبات به محدودهٔ آشوب نزدیک می‌شود. یک سیستم تا حد مشخصی می‌تواند رفتار آشوبی از خود نشان دهد. انرژی داخلی سیستم و نوسانات آن می‌تواند به اندازهٔ محدود و پایان‌پذیری تغییر در سیستم ایجاد کند که به آن لبهٔ آشوب[7] می‌گوئیم. لبهٔ آشوب نقطه‌ای است که فعالیت تمام اجزای سیستم به بیشترین حد می‌رسد و سیستم در بیشینهٔ پتانسیل و توانائی خود قرار می‌گیرد و در این نقطه سیستم دچار انشعاب می‌شود. در مرحلهٔ انشعاب، سیستم در یک فضای مجازی قرار می‌گیرد که انتخاب می‌کند به کدام سمت باید حرکت کند؛ درواقع در تعادل قبلی، محور تعادل یک موجودیت مجرد یا مجازی بود که تعادل را حفظ می‌کرد. این موجودیت مجرد یا مجازی، رفتار سیستم را به سمت خود جذب می‌کند و بدین صورت باعث تعادل آن می‌شود؛ ازاین‌رو آن را رباینده[8] می‌نامیم. در لبهٔ آشوب، سیستم از تعادلی که به واسطهٔ یک رباینده ایجاد شده بود، به تعادلی که به وسیلهٔ یک رباینده دیگر

[7] The Edge of Chaos
[8] Attractor

ایجاد شده‌است منتقل می‌شود و به عبارت دیگر تبدیل فاز[1] انجام می‌شود. لبهٔ آشوب محدوده‌ای است که در آن تبدیل فاز صورت می‌گیرد و سیستم از حالتی به حالتی دیگر تبدیل می‌شود (یا رابیندهٔ آن عوض می‌شود). در لبهٔ آشوب است که مشخص می‌شود کدام یک از رابینده‌ها باید به‌عنوان محور قرار گیرند و باید به آن‌ها پرش صورت گیرد و اینجا یک مرحله پیشرو صورت می‌گیرد و عمق آشوب بیشتر می‌شود. سیستم خود را سازماندهی می‌کند و در سطح بالاتری از پیچیدگی و یا تجزیه‌شدن قرار می‌گیرد. این همان تبدیل فاز است؛ مرحله‌ای که رخدادهای زودگذر در آن انجام می‌شود؛ هنگامی که یک سیستم پویای آشوبی، به علت آشفتگی و پریشانی محیط به بی ثباتی می‌رسد، یک رابینده خط سیر پریشان و حرکات سیستم را به سمت خود می‌کشد و در نقطهٔ تبدیل فاز، سیستم انشعاب پیدا می‌کند و رابینده خود را تغییر می‌دهد. این باعث جلو راندن به سمت نظم و تعادلی جدید بر اثر سازماندهی مجدد و یا تجزیه می‌شود. سیستم‌های آشوبی از همین خصوصیت بهره می‌برند و با استفاده از یک رابینده، به ثبات می‌رسند. یک رابینده باعث پایداری و ثبات سیستم می‌شود؛ درواقع رابینده، هر چه که می‌خواهد باشد، سیستم را از تفرق و از هم پاشیده‌شدن حفظ می‌کند. از ویژگی‌های تئوری آشوب می‌توان به خودسازماندهی (وفق‌دادن خود با شرایط محیطی) در محیط‌های پویا و خودمانایی (هر جزئی از سامانه دارای ویژگی کل بوده و مشابه آن است) و حساسیت به شرایط اولیه اشاره کرد. سامانه‌های آشوب مختلفی وجود دارند که در [۲۱] می‌توانید، انواع مختلفی از نگاشت‌های آشوب را مشاهده کنید. نمونهٔ مشهور تابع نگاشت لجستیک به صورت (۱۰) محاسبه می‌شود [۲۱].

$$\tau_{n+1} = A\tau_n(1-\tau_n) \qquad (۱۲)$$

در رابطهٔ (۱۲)، $\boldsymbol{\tau_n}$ یک عدد حقیقی در بازه $[۰,۱]$ است و پارامتر $\boldsymbol{A}$ که به ضریب لجستیک معروف است، منجر به ایجاد ویژگی‌های منحصر به فرد در تابع می‌شود. در این مقاله از نگاشت آشوب لجستیک استفاده شده و مقدار پارامتر $\boldsymbol{A}$ برابر با چهار در نظر گرفته شده‌است.

۲-۸-معیار کارایی الگوریتم‌ها در محیط پویا

معیارهای مختلفی برای سنجش کارایی الگوریتم‌های تکاملی در محیط‌های پویا وجود دارند که می‌توان به معیار برازش کلی[2]، معیار متوسط خطا[3]، معیار دقت[4]، معیار تطبیق‌پذیری[5]، خطای درون‌خطی[6] و برون‌خطی[7] اشاره کرد[۲۴]. بیشتر پژوهش‌گران از معیار خطای برون‌خطی برای سنجش کارایی روش خود و مقایسه با دیگر روش‌ها استفاده کرده‌اند. کارایی برون‌خطی بر اساس رابطهٔ (۱۳) محاسبه می‌شود[۲۴].

(۱۳)

$$Offlin\ Error\left(Pop^1, Pop^2, \dots, Pop^{FE}, fit(.)\right) = \frac{1}{FE}\sum_{t=1}^{FE} \min_j \left(fit(x_t^*) - fit\left(Pop_{j:}^{t,(\eta)}\right)\right)$$

که در معادلهٔ (۱۳)، FE بیشترین تعداد ارزیابی‌های شایستگی برای هر فرد و x_t^* بهینهٔ سراسری بعد از ارزیابی شایستگی و Pop^t بردار زمینه در t امین محیط است که بعد از η ارزیابی‌های شایستگی از ابتدای محیط جدید به‌روزرسانی می‌شود.

۲-۹-معیار محک قله‌های متحرک

مسائل محک برای بررسی عملکرد الگوریتم تکاملی در محیط‌های پویا استفاده می‌شود. مسائل محک می‌توانند شبیه‌سازهای خوبی برای محیط‌های پویا در دنیای واقعی باشند. یکی از معروف‌ترین مسائل معیار، معیار قله‌های متحرک (MPB)[8] است که برانک [۱۹] پیشنهاد کرد. در این مسئله تعدادی قله (بهینه) وجود دارد که در طول زمان تغییر می‌کنند. هر قله دارای ارتفاع، عرض و موقعیت است. در ادامه این معیار بررسی می‌شود. MPB یک مسئلهٔ پویای چندوجهی و چندبعدی است که در آن منظره از m قله در یک فضای n بعدی با ارزش واقعی تشکیل شده‌است. در MPB، چشم‌انداز تناسب قله P در فضای D بعدی است. در هر نقطه، شایستگی به‌عنوان بیشینهٔ توابع قله m تعریف می‌شود. این شایستگی را می‌توان به‌صورت زیر فرموله کرد[۱۹]:

$$F(\vec{x}, t) = max_{i=1\dots P}\ P_{sh}(\vec{x}, h_i(t), w_i(t), \vec{\rho}_i(t))) \qquad (۱۴)$$

که در آن $P_{sh}(\vec{x}, h_i(t), w_i(t), \vec{\rho}_i(t))$ تابعی است که برازش یک نقطهٔ معین $\vec{x}$ را برای یک قلهٔ توصیف‌شده با ارتفاع (h)، عرض (w) و موقعیت قلهٔ $\vec{\rho}$ توصیف می‌کند. هر ارزیابی Δe، h، w و $\vec{\rho}$ برای هر قله تغییر می‌کند و وضعیت محیط را تغییر می‌دهد. h و w هر قله توسط متغیرهای تصادفی گاوسی تغییر می‌کند و با پارامترهای

[1] Phase Transition

[2] Fitness Overall
[3] Average Error
[4] Accuracy
[5] Adaptability
[6] Online
[7] Offline
[8] Moving Peaks Benchmark

شدت ارتفاع (h_{sev}) و شدت عرض (w_{sev}) مقیاس‌بندی می‌شود. موقعیت با استفاده از طول شیفت s و ضریب هم‌بستگی λ جابه‌جا می‌شود. طول شیفت میزان حرکت قله را کنترل می‌کند؛ درحالی‌که عامل هم‌بستگی تعیین می‌کند که حرکت یک قله چقدر تصادفی باشد، اگر λ=0.0، حرکت یک قله به‌طور کامل تصادفی خواهد بود، اما اگر λ=1.0، قله همیشه در یک جهت حرکت می‌کند تا زمانی که به مرزی از فضای مختصات برسد که مسیر آن مانند پرتوی نور منعکس می‌شود. در زمان تغییر در محیط، تغییرات یک قلهٔ منفرد را می‌توان به‌صورت زیر توصیف کرد[۱۹]:

$$\begin{aligned} &r \in N(0,1)^D \\ &h_i(t) = h_i(t-1) + h_{sev}.r \\ &w_i(t) = w_i(t-1) + w_{sev}.r \\ &\vec{\rho}_i(t) = \vec{\rho}_i(t-1) + \vec{v}_i(t) \end{aligned} \quad (۱۵)$$

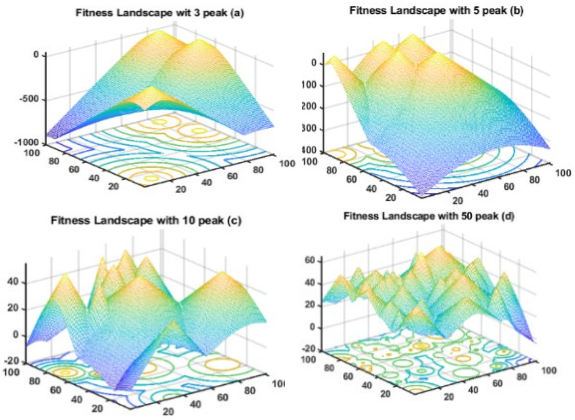
بردار $\vec{v}_i(t)$ یک بردار تصادفی $\vec{r}$ را با بردار تغییر پیشین $\vec{v}_i(t-1)$ ترکیب می‌کند. بردار تصادفی با رسم یکنواخت از [۰,۱] برای هر بعد و سپس مقیاس‌گذاری بردار به طول s ایجاد می‌شود. $\vec{v}_i(t)$ به‌صورت زیر به‌دست می‌آید[۱۹]:

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

تابع پیک برای h، w و p هر قله را می‌توان به‌صورت زیر محاسبه کرد[۱۹]:

$$P_{sh}(\vec{x}, h_i(t), w_i(t), \vec{\rho}_i(t)) = h_i(t) - w_i(t).\sqrt{\sum_{j=1..D}(x_j - \vec{\rho}_i(t)_j)^2} \quad (۱۷)$$

بخشی از رادیکال، فاصله بین نقطه وجود دارد و موقعیت هر قله بیان می‌شود[۱۹]. شکل (۹) فضای شایستگی را برای MPB در دو بعد نشان می‌دهد.



(شکل -۹): چشم‌انداز تناسب با قله‌های مختلف در MPB [۱۹]

(Figure-9): Fitness Landscape with the different peaks in MPB [19]

۲-۱۰-مروری بر کارهای گذشته

استفاده از الگوریتم‌های تکاملی (EA) و رویکردهای هوش ازدحامی برای بهینه‌سازی DOPها یک حوزهٔ پژوهشی محبوب و فعال است و به‌طور فزاینده‌ای توجه جامعهٔ محاسبات تکاملی (EC) را به خود جلب کرده‌است. این بخش راه‌حل‌های مختلفی را برای DOP ها معرفی می‌کند که هر کدام می‌توانند بر برخی از چالش‌های DOP غلبه کنند.

وقتی انسان با دنیایی در حال تغییر روبه‌رو می‌شود، نه‌تنها بر آینده، بلکه بر گذشته نیز تمرکز می‌کند. نگاه‌کردن به راه‌حل‌های مشابه کمک می‌کند تا برای آینده تصمیم‌گیری کنیم. زمانی که با موقعیتی مواجه می‌شویم که پیش‌تر تجربه کرده‌ایم، بهتر می‌توانیم با آن کنار بیاییم، اگر از اطلاعات گذشته در بهینه‌سازی و یادگیری حل مسائل پویا در حین جست‌وجو استفاده شود، می‌تواند به فرایند جست‌وجو کمک کند. بیشتر مسائل در دنیای واقعی ماهیتی پویا دارند و محیط به طور پویا در حال تغییر است. در بیشتر موارد، این محیط تغییراتی را تعریف کرده‌است که می‌توان از آن‌ها در محیط‌های جدید با ذخیرهٔ راه‌حل‌های گذشته در حافظه استفاده کرد. با فرض شباهت بین محیط‌های پیوسته [۲۱ و ۲۴]، در DOPها استفاده از اطلاعات به‌دست‌آمده از محیط‌های پیشین برای سرعت‌بخشیدن به کشف بهینه در محیط جدید رایج است. در DOPها، این اطلاعات به طور معمول شامل مکان(های) ناحیه(های) فضای مسئله؛ یعنی قله(های) در چشم‌انداز شایستگی است. این اطلاعات بین محیط‌های پیوسته منتقل می‌شود و مکرر در طول هر محیط به‌روز می‌شود. DOPها به طور معمول از سه جزء اصلی برای استفاده از اطلاعات تاریخی استفاده می‌کنند: پوشش منطقهٔ امیدوارکننده، حافظهٔ ضمنی و حافظهٔ صریح [۲۱ و ۲۴]. مؤلفه‌های پوشش مناسب سعی می‌کنند چندین نفر را در اطراف منطقه(های) امیدوارکننده‌ای که در محیط‌های پیشین کشف شده‌اند، حفظ کنند. پژوهش‌های [۲۱، ۲۳، ۲۵، ۲۶] ثابت کرده‌است که حافظه به‌طور مؤثر مسائل بهینه‌سازی پویا را حل می‌کند. پژوهش‌گران راه‌حل‌های مختلفی برای بهبود حافظه در حل مسائل بهینه‌سازی پویا ارائه کرده‌اند که هر کدام به نحوی توانسته‌اند بر نقاط ضعف حافظهٔ استاندارد غلبه کنند. حافظه از چند جهت به بهینه‌سازی پویا کمک می‌کند: راه‌حل‌های خوب گذشته را حفظ می‌کند، جست‌وجوی راه‌حل‌های مشابه را پس از تغییرات محیطی سرعت می‌بخشد، و ورودی‌های حافظه تا حد زیادی به جست‌وجوی پس از تغییر کمک می‌کنند. آن‌ها به تنوع‌بخشیدن به فرایند جست‌وجو کمک می‌کنند و

درنهایت، حافظه به ایجاد مدلی از مسائل پویا در هر زمان کمک می‌کند. رویکردهای مبتنی بر حافظه برای مسائل بهینه‌سازی در محیط‌های پویا به دو دسته تقسیم می‌شوند: حافظهٔ ضمنی و حافظهٔ صریح [۲۱-۲۳].

حافظهٔ ضمنی: حافظهٔ ضمنی برای ذخیرهٔ تمام اطلاعات از جمله اطلاعات اضافی در مورد افراد (کروموزوم‌ها) استفاده می‌کند. همیشه به تمام اطلاعات مربوط به یک فرد (کروموزوم) نیاز نیست. با توجه به محدودیت حافظه، سعی می‌شود بهترین اطلاعات در مورد هر ذره ذخیره شود.

حافظهٔ صریح: حافظهٔ صریح برای DOPها اطلاعات تاریخی محیط‌ها و خاطرات پیشین را جدا از جمعیت در یک آرشیو حافظه ذخیره می‌کند. حافظهٔ صریح بسیار محبوب است و به‌طور گسترده‌ای برای بهینه‌سازی پویا استفاده می‌شود. راه‌بردهای خاص برای ذخیره و بازیابی اطلاعات از حافظهٔ بین راه‌کارها متفاوت است، اما ساختار کلی حافظه مشابه است. در ادامهٔ این مقاله، حافظه به‌عنوان یک بانک حافظهٔ صریح و ورودی حافظه به‌عنوان بخشی از حافظه تعریف شده‌است. اطلاعات تاریخی به‌دست‌آمده از محیط‌های پیشین، به‌ویژه مکان مناطق امیدوارکننده (بهینه)، می‌تواند برای سرعت‌بخشیدن به فرایند ردیابی در هر محیط جدید مفید باشد. برخی از DOPها از حافظهٔ صریح برای ذخیرهٔ اطلاعات تاریخی استفاده می‌کنند[۲۱ و ۲۳].

مقالهٔ [۲۰] رویکردی مبتنی بر الگوریتم ABC بهبودیافته با حافظهٔ صریح و خوشه‌بندی جمعیت برای حل مسائل بهینه‌سازی پویا ارائه می‌دهد. الگوریتم پیشنهادی از حافظهٔ صریح برای ذخیرهٔ بهترین راه‌حل‌های قدیمی و خوشه‌بندی برای حفظ تنوع جمعیت استفاده می‌کند. استفاده از بهترین راه‌حل‌های گذشته و حفظ تنوع در جمعیت راه‌حل‌های محیطی کاندید به تسریع هم‌گرایی الگوریتم کمک می‌کند. از مزایای این روش می‌توان به ردیابی قله‌های متحرک با استفاده از روش خوشه‌بندی، کارایی در شدت تغییرات و فضای مسئله با ابعاد بالا اشاره کرد.

مقالهٔ [۲۱] یک الگوریتم ژنتیک آشوب‌گونه مبتنی بر خوشه‌بندی و حافظه را برای حل مسائل پویا ارائه می‌کند. یک سامانهٔ آشوب‌گونه، پیش‌بینی دقیق‌تری از آینده نسبت به یک سامانهٔ تصادفی دارد و درجهٔ هم‌گرایی در الگوریتم را افزایش می‌دهد. به طور معمول، اطلاعات گذشته به الگوریتم اجازه می‌دهد تا پس از تغییر محیط، به سرعت با شرایط محیطی جدید سازگار شود؛ بنابراین استفاده از یک حافظهٔ خوب می‌تواند با یک استراتژی مناسب اطلاعات مفید گذشته را ذخیره کرده و برای استفادهٔ مجدد بازیابی کند. خوشه‌بندی در حافظه و جمعیت اصلی، تنوع را در طول اجرای الگوریتم با تبادل اطلاعات بین خوشه‌های مربوطه (خوشه‌هایی با برچسب مشابه) در حافظه و جمعیت اصلی حفظ می‌کند. به‌طور کلی، دو جنبه از مشارکت اساسی در این مقاله پیشنهاد شده‌است؛ روش خوشه‌بندی هر دو جمعیت اصلی و حافظه را خوشه‌بندی می‌کند؛ دیگری راه‌حل خوبی است که برای بهبود حافظه استفاده می‌شود. برای آزمایش کارایی روش پیشنهادی، از مسئلهٔ محک قله‌های متحرک (MPB) استفاده می‌شود که رفتاری مشابه مسائل پویا در دنیای واقعی را شبیه‌سازی می‌کند. نتایج تجربی کارایی مناسب روش پیشنهادی را در حل مسائل بهینه‌سازی پویا در مقایسه با سایر روش‌های موجود نشان می‌دهد.

در مقالهٔ [۲۷] رویکردی بر اساس الگوریتم ABC، خوشه‌بندی و حافظهٔ صریح ارائه شده‌است. روش پیشنهادی در این مقاله CMCABC نامیده می‌شود. در این رویکرد از حافظهٔ صریح برای ذخیرهٔ راه‌حل‌های خوب پیشین که قدمت چندانی ندارند استفاده شده‌است. حفظ تنوع در محیط‌های پویا یکی از چالش‌های اساسی در حل مسائل بهینه‌سازی پویا است. برای غلبه بر چالش تنوع از روش خوشه‌بندی استفاده شده‌است. استفاده از خوشه‌بندی در روش پیشنهادی می‌تواند به‌خوبی تنوع محیط مشکل را حفظ کند.

در [۲۸]، یک رویکرد مبتنی بر حافظه برای مسائل بهینه‌سازی پویای چندهدفه ارائه شده‌است. این مقاله یک حافظهٔ بهبودیافته و الگوریتم تکاملی چندهدفه را بر اساس تجزیه (که با dMOEA/D-M نشان داده شده‌است) ادغام کرده‌است. این روش از یک سازوکار تطبیق پایدار (STM) ساده و مؤثر (که با dMOEA/D-STM مشخص می‌شود) استفاده می‌کند.

در [۲۹]، بهینه‌سازی ازدحام ذرات خوشه‌بندی (CPSOR) برای حل مسائل بهینه‌سازی پویا پیشنهاد شد. در این روش ذرات خوشه‌بندی می‌شوند و ذرات درون هر خوشه را به‌صورت محلی جست‌وجو می‌کنند. خوشه‌بندی مورد استفاده در این روش از نوع خوشه‌بندی سلسله‌مراتبی است. پیچیدگی محاسباتی مربوط به خوشه‌بندی در این روش $O(N^3)$ است که N اندازهٔ جمعیت را نشان می‌دهد. هر ذره به سمت میانگین بهترین موقعیت مشاهده‌شده توسط ذرات در پارتیشنی که در آن قرار دارد حرکت می‌کند. راه‌حل بهینه برای توابع چند قله در محیط‌های پویا با ایجاد تغییرات در الگوریتم تکاملی بهینه‌سازی ازدحام ذرات به‌دست می‌آید. برای

انجام این کار از یک مدل مبتنی بر گونه‌سازی استفاده می‌شود که امکان توسعهٔ زیرجمعیت‌های موازی را فراهم می‌کند. این مدل یک استراتژی برای تشویق ردیابی قله‌ها در توابع چندقله‌ای است؛ درحالی‌که از تجمع ذرات در قله‌ها نیز جلوگیری می‌کند.

در [۲۴] یزدانی و همکاران روش‌های مختلفی برای حل DOP ارائه کرده‌اند. یکی از روش‌های این مقاله افزایش حافظه برای DOPها است؛ همچنین در این مقاله روش‌های دیگری ارائه شده‌است؛ از جملهٔ این روش‌ها می‌توان به روش‌های خوشه‌محور، چندجمعیتی و روش‌هایی که تنوع را در جمعیت حفظ می‌کند، اشاره کرد. این بخش به بررسی برخی از آخرین روش‌های ذکرشده در این مقاله می‌پردازد.

همان‌طورکه در مقالهٔ [۳۰] بیان شد، استفاده از حافظهٔ صریح تنها برای کلاسی از DOPها مناسب است که در آن بازده بهینهٔ آن به مکان پیشین یا محیط‌های پیشین به طور دوره‌ای ظاهر می‌شود. برای مقابله با چنین DOPها، اطلاعات تاریخی حفظ‌شده باعث می‌شود DOPها بلافاصله به بهینهٔ سراسری در محیط جدید تبدیل شوند[۳۱]؛ علاوه‌بر این، اگر بتوان حرکت نواحی یا قله‌های امیدوارکننده را پیش‌بینی کرد، می‌توان از راه‌حل‌های حفظ‌شده به‌عنوان مجموعه‌دادهٔ آموزشی برای روش‌های پیش‌بینی استفاده کرد[۳۱-۳۴]؛ علاوه‌بر مدیریت حافظهٔ کامل و حذف راه‌حل‌های اضافی، راه‌حل‌های ذخیره‌شده پس از هر تغییر قدیمی می‌شوند؛ درنتیجه، هر راه‌حل ذخیره‌شده که نشان‌دهندهٔ موقعیت یک منطقهٔ امیدوارکننده است، باید با عملیات بهره‌برداری به‌روز شود. این فرایند بهره‌برداری را می‌توان با تزریق راه‌حل‌های ذخیره‌شده به جمعیت و به‌روزرسانی آن‌ها با حذف راه‌حل‌های قدیمی مشابه [۳۵]، [۳۶]، یا با انجام یک جست‌وجوی محلی مستقل در راه‌حل‌های ذخیره‌شده پس از ایجاد هرگونه تغییر محیطی مورد بهره‌برداری قرار داد[۳۷-۴۰]. در برخی از DOPهای چند دسته [۴۰]، هنگامی که یک زیرجمعیت هم‌گرا کشف شد، بهترین مکان آن در حافظه ذخیره می‌شود؛ زیرجمعیت به‌صورت تصادفی جست‌وجو یا حذف می‌شود تا از هدر رفتن منابع محاسباتی جلوگیری شود؛ بااین‌حال، یک روش غیرفعال‌سازی ساده می‌تواند جایگزین اجزای حافظهٔ صریح شود که در آن زیرجمعیت‌های هم‌گرا تا تغییرات محیطی بیشتر غیرفعال می‌شوند[۴۱-۴۳].

در [۴۴]، چندین راه‌حل برای مدیریت مسائل بهینه‌سازی مبتنی بر حافظهٔ پویا پیشنهاد شده‌است. این مکانیسم‌ها عبارت‌اند از: استفاده از حافظهٔ صریح، راه‌اندازی مجدد، بهبود جست‌وجوی محلی با ذخیرهٔ راه حل‌های آرشیوشده و حافظهٔ ترکیبی.

پژوهش‌گران در [۴۵] یک روش دو جمعیتی مبتنی بر حافظه را پیشنهاد کرده‌اند. این روش از دو زیرجمعیت تشکیل شده‌است: زیرجمعیت کاوش‌گر و زیرجمعیت بهره‌بردار. در این روش، زیرجمعیت‌ها از طریق یک حافظهٔ صریح با هم کار می‌کنند. هنگامی‌که کاوش‌گر هم‌گرا می‌شود، بهترین موقعیت پیدا شدهٔ خود را به حافظه می‌فرستد و زیرجمعیت بهره بردار می‌تواند از بهترین موقعیت ذخیره‌شده در حافظه استفاده کند.

در [۴۶]، یک روش چند جمعیتی مبتنی بر بایگانی اطلاعات مفید قدیمی معرفی شده‌است. در این روش بهترین موقعیت یافت‌شده به وسیلهٔ هر زیرجمعیت به بایگانی در انتهای هر محیط ارسال می‌شود. جمعیت‌های فرعی از طریق آرشیوها با هم کار می‌کنند. هر زیرجمعیت بهترین موقعیت موجود در حافظه را ذخیره می‌کند تا سایر زیرجمعیت‌ها بتوانند با سرعت بالاتری به بهینهٔ جهانی هم‌گرا شوند.

رویکردهای مختلفی بر اساس ذخیرهٔ اطلاعات در حافظه توسط پژوهش‌گران مختلف ارائه شده‌است. در [۴۷]، بهترین موقعیت یافت‌شده به وسیلهٔ هر زیرجمعیت پس از هم‌گرایی به حافظه ارسال می‌شود. این رویکرد راه‌حل‌های مناسب گذشته را ذخیره می‌کند و با استفاده از مکانیزمی این راه‌حل‌ها را از حافظه بازیابی می‌کند. راه‌حل‌های بازیابی‌شده برای پیش‌بینی آینده بسیار مناسب‌اند. این رویکرد در برخی از آثار بعدی مانند [۴۸-۵۱] به‌کار گرفته شده‌است. در [۵۲]، بهترین مکان‌های یافت‌شده به وسیلهٔ زیرجمعیت‌ها پس از هر تغییر محیطی بایگانی می‌شوند.

روش FTMPSO [۵۳] توسط یزدانی و همکاران ارائه شد. سه روش PSDR-hPSO، oTMO و cTMO نسخه‌های ساده‌شدهٔ FTMPSO هستند. در این روش‌ها مکانیسم‌های ذرات بهره‌بردار و بیداری خواب غیرفعال می‌شوند. فرض بر این است که تمام الگوریتم‌ها از تغییرات محیطی مطلع‌اند[۵۳]. یزدانی و همکاران از روش TMO در FTMPSO استفاده کرد که راه‌حل‌ها را به بهترین موقعیت در هر محیط تغییر می‌دهد. آن‌ها روش خود را TFTmPSO [۵۳] نامیدند؛ همچنین نسخهٔ ROOT FTmPSO پیشنهادشده توسط یزدانی و همکاران به‌عنوان یک روش ROOT استفاده می‌شود و RFTmPSO [۵۳] نامیده می‌شود. در [۵۳]، یک الگوریتم جدید انتخاب‌گر راه‌حل تطبیقی (ASC) ارائه شده‌است. این روش به روشی مشابه الگوریتم‌های TMO عمل می‌کند که در آن SC کوچک است و مانند الگوریتم‌های ROOT زمانی که SC بالا است، عمل می‌کند[۵۳]. الگوریتم ASC می‌تواند تصمیم بگیرد که آیا راه‌حل‌های دیگر را بر اساس مقادیر

تناسب راه‌حل فعلی تغییر دهد یا نگه دارد. روش RFTmPSO-s4[۵۳] یک الگوریتم TMO مبتنی بر FTmPSO است که در آن، زمانی که راه‌حل قوی فعلی قابل قبول نیست، الگوریتم به سادگی بهترین موقعیت پیداشده را به‌عنوان راه‌حل قوی بعدی انتخاب می‌کند. RAmQSO-s4 و RMNAFSA-s4، همان مکانیزم حذف مانند RFTmPSO-s4 با مقدار فاکتور excl یکسان استفاده می‌شوند. روش‌های مختلف FTmPSO(TMO)، ROOT-PV، ROOT-TFV، RAmQSO-s4، RmNAFSA-s4، TFTmPSO، RFTmPSO و ASC در رسالهٔ دکترای یزدانی ارائه شده‌است[۵۳].

۳- روش پیشنهادی

در این مقاله یک الگوریتم رقابت استعماری آشوب‌گونه[1] مبتنی بر حافظه و خوشه‌بندی برای محیط‌های پویا پیشنهاد داده شده‌است. مراحل روش پیشنهادی همراه با جزئیات در ادامه تشریح شده‌است.

۳-۱- ایجاد جمعیت اولیه بر اساس تئوری آشوب

در الگوریتم رقابت استعماری استاندارد جمعیت اولیه به‌صورت تصادفی ایجاد می‌شود. در این روش جمعیت اولیه بر خلاف الگوریتم استاندارد، طبق تئوری آشوب (نگاشت ترابری) مقداردهی اولیه می‌شوند. در طبیعت رفتار بسیاری از سامانه‌ها به‌صورت آشوب‌گونه‌اند و در جهان واقعی سامانه‌ها به جای 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.

مقداردهی اولیهٔ جمعیت طبق رابطهٔ (۱۸) است:

$$x(i) = LB_i + (UB_i - LB_i) \times A\tau_n(1 - \tau_n), \forall j \in (1,2,\dots,D), \forall i \in (1,2,\dots,N) \quad (۱۸)$$

همان‌گونه که در رابطهٔ (۱۸) مشاهده می‌شود به جای مقدار تصادفی $Rand$ از تابع نگاشت لجستیک استفاده شده‌است؛ بنابراین جمعیت اولیه بر اساس تئوری آشوب ایجاد می‌شود. N تعداد جمعیت و D ابعاد فضای مسئله است.

۳-۲-مقداردهی اولیهٔ جمعیت در حافظهٔ صریح

در روش پیشنهادی از یک حافظهٔ صریح برای ذخیرهٔ بهترین راه‌حل‌های خوب جهت استفادهٔ مجدد در محیط جدید استفاده می‌شود. در یک محیط پویا به‌دلیل چرخه‌ای‌بودن این محیط، ممکن است بهینه که در چند نسل پیش ظاهر شده مجدد در همان نقطه ظاهر شود و در این‌صورت یک حافظه می‌تواند به‌راحتی این بهینه را شناسایی کرده و سرعت هم‌گرایی را در الگوریتم افزایش دهد.

حافظه در ابتدا همانند جمعیت اصلی باید مقداردهی اولیه شود و طبق همان رابطهٔ آشوب که جمعیت اصلی را مقداردهی اولیه می‌کند، جمعیت حافظه نیز مقداردهی اولیه می‌شود. مقداردهی جمعیت برای جمعیت حافظه طبق رابطهٔ (۱۹) صورت می‌گیرد.

$$m_j(i) = LB_i + (UB_i - LB_i) \times A\tau_n(1 - \tau_n), \forall j \in (1,2,\dots,D), \forall i \in (1,2,\dots,N) \quad (۱۹)$$

۳-۳-خوشه‌بندی جمعیت

عناصر در جمعیت اصلی و حافظه خوشه‌بندی می‌شوند که هر خوشه (C) می‌تواند شامل یک یا چند دسته (G) باشد یا این‌که هر دسته می‌تواند شامل یک یا چند خوشه باشد. برای خوشه‌بندی از روش خوشه‌بندی اقلیدسی استفاده شده‌است. خوشه‌بندی در روش پیشنهادی باعث می‌شود تنوع در حین اجرای الگوریتم برای جمعیت حفظ شود؛ همان‌طورکه پیش‌تر بیان شد، حفظ تنوع بعد از هر تغییر در محیط یکی از اساسی‌ترین چالش‌ها در حل مسائل بهینه‌سازی پویا است. در ادامه با ذکر شکل روش خوشه‌بندی به طور واضح تشریح خواهد شد.

۳-۳-۱-ذخیره در حافظه (store)

ذخیره و نگهداری راه‌حل‌های کاندید در حافظه یکی از مهم‌ترین مسائل در کارهای پیشین است. ابتدا باید تصمیم گرفته شود که داده‌های موجود در حافظه چگونه به‌روز می‌شوند و دوم این‌که کدام داده‌ها را می‌توان به‌عنوان راه‌حل ورودی به حافظه اضافه کرد. با توجه به این‌که حافظه‌ها فضای ذخیره‌سازی محدودی دارند، اگر الگوریتم قصد ذخیرهٔ یک راه‌حل نامزد را داشته باشد، راه‌حل جدید باید با یکی از راه‌حل‌های موجود جایگزین شود. استراتژی انتخاب داده‌هایی که باید از حافظهٔ صریح حذف شوند (از بین موارد پیشین) سازوکار جایگزینی نامیده می‌شود. در این راه‌حل به هر یک از افراد موجود در حافظه صفتی به نام سن نسبت داده می‌شود. در مرحلهٔ ورود به حافظه، افرادی که در هر نسل سنی برابر با صفر دارند، یک واحد به سن آن‌ها اضافه می‌شود؛ برای مثال، اگر بیشینهٔ سن افراد حافظه صد در نظر گرفته شود، اگر سن فرد به صد برسد، آن فرد به‌عنوان قدیمی‌ترین داده در حافظه در نظر گرفته می‌شود و اگر حافظه پر است و نیاز به به‌روزرسانی دارد، قدیمی‌ترین داده از حافظه حذف می‌شود و دادهٔ جدید جایگزین آن می‌شود و همچنین سن آن صفر می‌شود. برای ذخیرهٔ بهترین داده در حافظه، باید موقعیت بهترین داده و خوشهٔ مربوط به آن تعیین شود.

فرض کنید که خوشه‌ها به‌صورت $\zeta^{\mathrm{POP}} = \{\zeta_1^{\mathrm{POP}}, \zeta_2^{\mathrm{POP}}, \dots, \zeta_K^{\mathrm{POP}}\}$ در نظر گرفته شوند، که در آن K تعداد خوشه‌ها در جمعیت است، $\zeta_j^{\mathrm{POP}} \subset \left\{1,2,\dots,\frac{\mathrm{CS}}{2}\right\}$ از خوشه j ام در ازدحام نشان می‌دهد که $\forall i,j \in$

[1] Chaotic artificial bee colony algorithm

$\{1,2,\dots,K\}: i \neq j \rightarrow \zeta_j^{\mathrm{POP}} \cap \zeta_i^{\mathrm{POP}} = \emptyset$؛ همچنین فرض کنید که بهترین شاخص به‌عنوان گروه best_ind در نظر گرفته شود. اکنون مرکز خوشه‌ها را می‌توان به صورت زیر به‌دست آورد:

$$Cluster_{ji}^{POP} = \frac{1}{\left|\zeta_j^{POP}\right|} \sum_{j \in \pi_i^{POP}} POP_{ji} \qquad (۲۰)$$

که $Cluster_{ji}^{POP}$ مؤلفهٔ i-ام خوشه j ، برابر با میانگین جزء i-امین آن خوشه است. فاصلهٔ بهترین موقعیت داده از مرکز همی خوشه‌ها مشخص می‌کند که موقعیت بهترین داده متعلق به کدام خوشه است (یعنی کدام خوشه به مرکز نزدیکتر است) و به‌صورت زیر به‌دست می‌آید:

$$\begin{aligned} &\boldsymbol{best_{ind}} \in \text{ .i} \\ &\boldsymbol{\zeta_{\bar{j}}^{POP} if\ f\ \forall u \in} \\ &\{1,2,\dots,k\}\backslash\{\bar{j}\}: \\ &\left(\sum_{q=1}^{D} \left|Cluster_{\bar{j}q}^{POP} - POP_{best_{ind}}\right|^2\right) \\ &\leq \left(\sum_{q=1}^{D} \left|Cluster_{U_q}^{POP}\right.\right. \\ &\left.\left. - POP_{best_{ind}}\right|^2\right) \end{aligned} \qquad (۲۱)$$

اکنون هر عنصر حافظه به یک خوشه اختصاص داده می‌شود. عناصر موجود در حافظهٔ N با استفاده از (۲۲) خوشه‌بندی می‌شوند (که در آن $N_{\bar{j}:}$ عنصر $\bar{j}$امین حافظه‌است). مشخص می‌شود که کدام یک از عناصر حافظه عضوی از خوشهٔ $\zeta_{\bar{j}}^{POP}$ است. این مجموعه با $N\left(\zeta_{\bar{j}}^{POP}\right)$ نشان داده می‌شود؛

$$\begin{aligned} &\bar{j}|\forall u \in \{1,2,\dots,k\}\backslash\{\bar{j}\}: \\ &M\left(\zeta_{\bar{j}}^{POP}\right) = \left(\sum_{q=1}^{D} \left|C_{\bar{j}_q}^{POP} - N_{J_q}\right|^2\right) \\ &\leq \left(\sum_{q=1}^{D} \left|C_{U_q}^{POP} - N_{J_q}\right|^2\right) \end{aligned} \qquad (۲۲)$$

سپس یک عنصر از حافظهٔ $N\left(\zeta_{\bar{j}}^{POP}\right)$ که از مرکز خوشهٔ $Cluster_{\bar{j}:}^{POP}$ فاصله دارد انتخاب شده و از حافظه حذف می‌شود؛ یعنی در مجموعهٔ $N\left(\zeta_{\bar{j}}^{POP}\right)$ بدترین عنصر انتخاب شده‌است (بدترین عنصر دورترین عنصر از مرکز خوشه است که بهترین موقعیت را دارد). اکنون $POP_{Best\_ind,:}$ با عضوی از $N\left(\zeta_{\bar{j}}^{POP}\right)$ جایگزین می‌شود که بیشترین فاصله را با $Cluster_{\bar{j}:}^{POP}$ دارد و این باعث می‌شود داده از بدترین عضو ذخیره‌شده در حافظه و عضو مجاور خود در بهترین موقعیت باشد. اگر عنصر حافظه در خوشهٔ $\zeta_{\bar{j}}^{POP}$ وجود نداشته باشد؛ یعنی $N\left(\zeta_{\bar{j}}^{POP}\right) = \emptyset$ باشد آنگاه نزدیکترین عنصر حافظه با مرکز خوشه $Cluster_{\bar{j}:}^{POP}$ انتخاب می‌شود. شبه‌کد روش ذخیره‌سازی حافظه در شکل (۱۰) نشان داده شده‌است.

Memory Update in the proposed method:

01. cluster POP into K clusters;
    $best_{ind} \in \zeta_{\bar{j}}^{POP}$ iff $\forall u \in \{1,2,\dots,K\}\backslash\{\bar{j}\}: \left(\sum_{q=1}^{D} \left|Cluster_{\bar{j}q}^{POP} - POP_{best_{ind},q}\right|^2\right) \leq \left(\sum_{q=1}^{D} \left|Cluster_{uq}^{POP} - POP_{best_{ind},q}\right|^2\right)$
02. find the cluster $\zeta_{\bar{j}}^{\mathrm{POP}}$ with the best food source $Pop_{best_{ind},:}$;
03. find $N\left(\zeta_{\bar{j}}^{POP}\right)$;
04. if $N\left(\zeta_{\bar{j}}^{POP}\right) \neq \emptyset$
    I_{FM}= Index of the memory element in $N\left(\zeta_{\bar{j}}^{POP}\right)$ that is furthest from the cluster center $Cluster_{\bar{j},:}^{POP}$;
    else
    I_{FM}= Index of the memory element that is nearest to the cluster center $Cluster_{\bar{j}}^{Pop}$;
05. $N_{I_{FM}:} = Pop_{best_{ind},:}$;

(شکل-۱۰): شبه‌کد برای روش ذخیره‌سازی حافظه
(Figure-10): pseudo-code for the storing memory method

۳-۳-۲-بازیابی حافظه

الگوریتم‌های بهینه‌سازی تکاملی می‌توانند حافظه‌های ذخیره‌سازی را به دو روش بررسی کنند؛ در حین اجرا یا پس از هر تغییر. اگر اطلاعات حافظه پس از هر تغییر بازیابی شود، بدترین عضو ازدحام با یک رونوشت از ورودی در حافظهٔ اصلی جایگزین می‌شود؛ در غیر این صورت، اگر بازیابی در طول زمان اجرا اتفاق بیافتد، ورودی‌های حافظهٔ اصلی را می‌توان به‌صورت دوره‌ای بازیابی کرد. یکی از راه‌بردهای بازیابی از حافظه، شناسایی بهترین داده از حافظه و جایگزینی بدترین داده از جمعیت اصلی است. برای بازیابی بهترین ورودی از حافظه، باید محل بهترین ورودی ذخیره‌شده در حافظه و خوشهٔ مربوط به آن مشخص شود. فرض کنید خوشه‌ها به‌صورت $\zeta^N = \{\zeta_1^N, \zeta_2^N, \dots, \zeta_{\ddot{K}}^N\}$ در نظر گرفته می‌شوند که $\ddot{K}$ تعداد خوشه‌ها در جمعیت است و ζ_j^N نشان‌دهندهٔ j-امین خوشه در حافظه است. اجازه دهید $\zeta_j^N \subset \left\{1,2,\dots,\alpha \times \frac{CS}{2}\right\}$ و $\forall i,j \in \{1,2,\dots,\ddot{K}\}: i \neq j \rightarrow \zeta_j^N \cap \zeta_i^N = \emptyset$ که در آن α نرخ اندازهٔ حافظه (به‌طور معمول ۰.۱) و $(1+\alpha) \times \frac{CS}{2}$

اندازهٔ کل جمعیت است؛ درنتیجه، شاخص بهترین ورودی حافظه با $best_{ind}$ تعیین می‌شود و مرکز خوشه‌ها به‌صورت زیر به‌دست می‌آید:

$$Cluster_{i,j}^{N} = \frac{1}{|\zeta_j^N|}\sum_{j\in\zeta_j^M} N_{ji} \qquad (۲۳)$$

$$best_{ind} \in \zeta_{\bar{j}}^{N} \; if \; f \; \forall u \in \{1,2,\dots,\bar{k}\}\backslash\{\bar{j}\}$$
$$\left(\sum_{q=1}^{D} |Cluster_{\bar{j}_q}^{N} - N_{best_{ind}}|^2\right) \leq \left(\sum_{q=1}^{D} |Cluster_{U_q}^{N} - N_{best_{ind}}|^2\right) \qquad (۲۴)$$

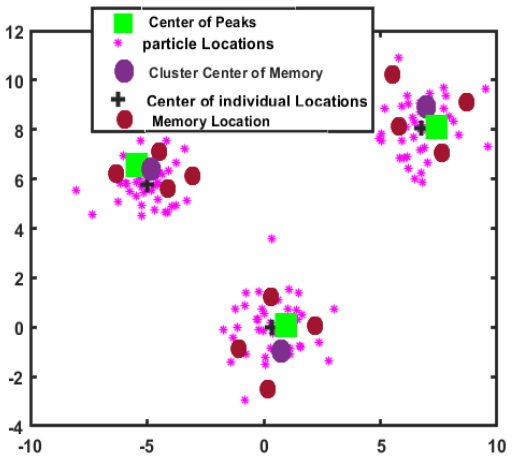
اکنون هر داده به یک خوشهٔ حافظه اختصاص داده می‌شود. برای $POP_{\bar{j}:}$ با استفاده از (۲۴)، بررسی می‌کنیم که کدام ذره در ازدحام عضوی از خوشه $\zeta_{\bar{j}}^{N}$ است. این مجموعه با $POP(\zeta_{\bar{j}}^{N})$ نشان داده می‌شود؛ سپس از بین داده‌ها $POP(\zeta_{\bar{j}}^{N})$، داده‌ای که از مرکز خوشه $Cluster_{\bar{j}:}^{N}$ دورتر است برای حذف از گروه انتخاب می‌شود؛ یعنی در مجموعهٔ $POP(\zeta_{\bar{j}}^{N})$ بدترین عنصر را انتخاب می‌کنیم (بدترین عنصر دورترین عنصر از مرکز خوشه است که بهترین موقعیت را دارد). اکنون $M_{Best\_ind,:}$ را با عضو $(POP(\zeta_{\bar{j}}^{N}))$ جایگزین می‌کنیم که بیشترین فاصله را از $Cluster_{\bar{j}:}^{N}$ دارد و این بهترین داده را عضو ذخیره شده در جمعیت و در مجاورت خود در حافظه در بدترین موقعیت قرار می‌دهد. اگر هیچ عنصری از گروه در خوشهٔ $\zeta_{\bar{j}}^{N}$ وجود نداشته باشد؛ یعنی $POP(\zeta_{\bar{j}}^{N}) = \emptyset$، آنگاه عنصری از گروه که نزدیکترین به مرکز خوشه $Cluster_{\bar{j}:}^{N}$ است، انتخاب شده‌است. شبه‌کد روش بازیابی حافظه در شکل (۱۱) نشان داده شده‌است. شکل (۱۲) فضای مسئله را نشان می‌دهد که در آن مراکز خوشه برای حافظه مشخص شده‌اند. شکل (۱۲) نشان می‌دهد که داده‌ها در حافظه خوشه هستند و مرکز حافظه و قله‌های موجود در فضای مسئله نیز نشان داده شده‌اند.

موضوع دیگری که در محیط‌های پویا به‌عنوان یک چالش مطرح می‌شود، تشخیص تغییرات در محیط است. روش‌های مختلفی برای تشخیص تغییرات در محیط وجود دارد که به طور کامل به نوع تغییرات محیط بستگی دارد. در این مقاله، هدف ما حل مسائلی است که محیط را به شیوه‌ای سراسری تغییر می‌دهد؛ یعنی میزان برازش تمام نقاط محیط تغییر می‌کند. تنها با آزمایش یک نقطه در محیط و مقایسهٔ میزان برازش به‌دست‌آمده با مقدار ذخیره‌شدهٔ پیشین آن، می‌توان به تغییر در محیط پی برد. تشخیص لحظهٔ تغییر در محیط در شکل (۱۳) نشان داده شده‌است. شبه‌کد روش پیشنهادی در شکل (۱۴) نشان داده شده‌است.

Memory Retrieval in the proposed method:

01. cluster M into $\ddot{K}$ clusters;
    $best_{ind} \in \zeta_{\bar{j}}^{N} \; if \; f \; \forall u \in \{1,2,\dots,\ddot{K}\}\backslash\{\bar{j}\}: \left(\sum_{q=1}^{D} \left|Cluster_{\bar{j}_q}^{N} - N_{best_{ind},q}\right|^2\right) \leq \left(\sum_{q=1}^{D} \left|Cluster_{U_q}^{N} - N_{best_{ind},q}\right|^2\right)$
02. find the cluster $\zeta_{\bar{j}}^{M}$ with the best memory element $N_{best_{ind},:}$;
03. find $POP(\zeta_{\bar{j}}^{N})$;
04. if $POP(\zeta_{\bar{j}}^{N}) \neq \emptyset$;
    I_{FP}= Index of the population in $POP(\zeta_{\bar{j}}^{N})$ that is furthest from the cluster center $Cluster_{\bar{j}:}^{N}$;
    else
    I_{FP}= Index of the individual that is nearest to the cluster center $Cluster_{\bar{j}:}^{N}$;
05. $POP_{I_{FP}:} = N_{best_{ind},:}$;

(شکل-۱۱): شبه‌کد روش بازیابی حافظه
(Figure-11): pseudo-code of memory retrieval method



(شکل-۱۲): فضای مسئله با مراکز خوشه برای جمعیت و حافظه
(Figure-12): The problem space with the cluster centers for the particle swarm and memory

Calculating shift severity and fitness variance;
Introducing diversity;

initialize x_i^d, v_i^d
$pbest_{i=}x_i^d$
endfor
$gbest = \text{argmin}_{pbest_i} f(pbest_i)$
repeat
Update memory with (01 to 05);
01. cluster POP into K clusters;

$$best_{ind} \in \zeta_j^{POP} \text{ iff } \forall u \in \{1,2,\dots,K\} \setminus \{\bar{j}\}: \left(\sum_{q=1}^{D} \left|Cluster_{jq}^{POP} - POP_{best_{ind},q}\right|^2\right) \leq \left(\sum_{q=1}^{D} \left|Cluster_{uq}^{POP} - POP_{best_{ind},q}\right|^2\right)$$

02. find the cluster $\zeta_{\bar{j}}^{\text{POP}}$ with the best source $Pop_{best_{ind},:}$;
03. find $N(\zeta_j^{POP})$;
04. if $N(\zeta_j^{POP}) \neq \emptyset$
I_{FM}= Index of the memory element in $N(\zeta_j^{POP})$ that is furthest from the cluster center $Cluster_{j,:}^{POP}$;
else
I_{FM}= Index of the memory element that is nearest to the cluster center $Cluster_j^{Pop}$;
05. $N_{I_{FM}:} = Pop_{best_{ind},:}$;
Recall Modified PSO algorithm procedure;
if an environmental **change** happens then:
Recall **Memory Update in the proposed method;**
Recall **Memory Retrieval in the proposed method;**
endif
Execute exclusion mechanism for **PCPMC**;
Update excel if the number of trackers is changed;
Execute an iteration of PSO for **PCPMC**;
until stopping criterion is met;

(شکل-۱۴): شبه‌کد روش پیشنهادی
(Figure-14): Pseudo code of the proposed method

محاسبهٔ شایستگی برای جمعیت اولیه: برای هر کشور، مقدار تابع هدف محاسبه می‌شود. پیچیدگی $O(N \times C_f)$ است، که C_f نشان‌دهندهٔ پیچیدگی محاسبهٔ تابع هدف است.

خوشه‌بندی جمعیت و حافظه: الگوریتم پیشنهادی از خوشه‌بندی برای مدیریت تنوع و ساختاردهی جمعیت استفاده می‌کند. پیچیدگی خوشه‌بندی به طور معمول $O(N \times K \times D)$ است، که K تعداد خوشه‌ها است. پیچیدگی این مرحله $O(N \times K \times D)$ است.

حرکت مستعمرات به سمت استعمارگر (جذب): در این مرحله، هر مستعمره به سمت استعمارگر خود حرکت می‌کند که شامل محاسبات برداری و جابه‌جایی است (پیچیدگی $O(N \times D)$).

به‌روزرسانی حافظه: مکانیزم حافظه شامل جست‌وجو و جایگزینی داده‌ها در حافظه است. این فرایند به طور

Chang detection method:

if an environmental **change** happens then (**a to e**)

a. cluster M into $\ddot{K}$ clusters;

$$best_{ind} \in \zeta_j^N \; if \; f \;\; \forall u \in \{1,2,\dots,\ddot{K}\} \setminus \{\bar{j}\}: \left(\sum_{q=1}^{D} \left|Cluster_{j_q}^N - N_{best_{ind},q}\right|^2\right) \leq \left(\sum_{q=1}^{D} \left|Cluster_{U_q}^N - N_{best_{ind},q}\right|^2\right)$$

b. find the cluster $\zeta_{\bar{j}}^{\text{M}}$ with the best memory element $N_{best_{ind},:}$;
c. find $POP(\zeta_j^N)$;
d. if $POP(\zeta_j^N) \neq \emptyset$;
I_{FP}= Index of the population in $POP(\zeta_j^N)$ that is furthest from the cluster center $Cluster_{j:}^N$;
else
I_{FP}= Index of the individual that is nearest to the cluster center $Cluster_{j:}^N$;
e. $POP_{I_{FP}:} = N_{best_{ind},:}$;

(شکل-۱۳): تشخیص لحظهٔ تغییر در محیط
(Figure-13): Detecting the moment of change in the environment

۳-۴-پیچیدگی محاسباتی روش پیشنهادی

در این بخش، پیچیدگی محاسباتی الگوریتم پیشنهادی با درنظرگرفتن مراحل مختلف آن محاسبه و تحلیل می‌شود. الگوریتم پیشنهادی شامل مراحل زیر است که هرکدام از نظر پیچیدگی محاسباتی جداگانه بررسی شده‌اند:

مقداردهی اولیهٔ جمعیت: در این مرحله، N کشور (ذرات) با D بعد مقداردهی اولیه می‌شوند. پیچیدگی این مرحله $O(N \times D)$ است.

Proposed Algorithm

Input:
$\alpha, D, CS, I, LB_1, UB_1, LB_2, UB_2, \dots, LB_D, UB_D$
Output:
BEST-Solution, BEST-Fitness

1. Initialize POP by equations (14 and 15) and set $fP(i) = F(Pop_{i:})$ % initialization %
2. Initialize M by $M_{ji} = LB_i + (UB_i - LB_i) \times \tau_j$ and set $fm(i) = F(M_{i:})$ % initialization %
3. $FlgEnvCh = 0$; $NUTime = rand(5,10)$; % Environment change flag
4. **For** $Iteration$=1 to I
5. **Return** BEST-Solution and BEST-Fitness

Initialize multi − swarm method;
for each Particle i [1 … N]
forall sub − swarms do
Update database;

خطی با تعداد عناصر حافظه (M) و ابعاد مسئله مقیاس می‌شود (پیچیدگی $O(M \times D)$).

رقابت استعماری و بازتخصیص مستعمرات: در این مرحله ضعیف‌ترین مستعمره حذف شده و به امپراتوری قوی‌تر اختصاص داده می‌شود. این فرایند شامل محاسبات برای تعیین امپراتوری برتر و بازتخصیص است (پیچیدگی $O(N \times \log(N))$، با فرض استفاده از مرتب‌سازی برای مقایسه).

کنترل معیار توقف و انجام تکرارها: تمام مراحل بالا به‌طور مکرر برای T تکرار الگوریتم انجام می‌شود.

پیچیدگی کلی: با توجه به تکرارها $O(T \times$ ((مراحل پیشین)).

پیچیدگی کلی:

با ترکیب مراحل فوق، پیچیدگی محاسباتی کل الگوریتم پیشنهادی به صورت زیر است:

$$O(T \times [N \times (D + C_f + K \times D) + M \times D + N \times \log(N)])$$

در مسائل با ابعاد بالا، پیچیدگی به طور عمده تحت تأثیر تعداد ابعاد D و جمعیت N قرار می‌گیرد. خوشه‌بندی، به‌ویژه با تعداد بالای خوشه‌ها K، می‌تواند تأثیر قابل‌ت وجهی بر پیچیدگی داشته باشد؛ بااین‌حال، این فرایند برای حفظ تنوع و جلوگیری از هم‌گرایی زودهنگام ضروری است. حافظه و ساروکارهای مربوط به آن نیز بر پیچیدگی اثرگذارند، اما این فرایند نقش مهمی در مدیریت تغییرات محیطی دارد.

الگوریتم پیشنهادی، با وجود داشتن پیچیدگی محاسباتی وابسته به ابعاد و جمعیت، به‌دلیل طراحی مؤلفه‌های حافظه، خوشه‌بندی و دافعه توانسته است، تعادل مناسبی بین اکتشاف و بهره‌برداری ایجاد کند. این ویژگی، همراه با قابلیت انطباق با تغییرات محیطی، موجب شده‌است که الگوریتم در مسائل بهینه‌سازی پویا عملکرد مناسبی از نظر کارایی و هزینهٔ محاسباتی داشته باشد.

۴- آزمایش‌ها و نتایج تجربی

به منظور ارزیابی الگوریتم‌های پیشنهادی و مقایسهٔ آن‌ها با الگوریتم‌های دیگر در محیط‌های پویا، از محک قله‌های متحرک استفاده شده‌است[۱۷]؛ محک قله‌های متحرک به‌عنوان یک محیط پویای سراسری برای آزمایش کارایی الگوریتم‌های مختلف در نظر گرفته شده‌است. تمام آزمایش‌ها بر روی الگوریتم‌ها در شرایط مشابه صورت گرفته‌است و الگوریتم‌های مورد مقایسه همگی بر اساس پیاده‌سازی‌های موجود بر روی محک ارزیابی شده‌اند. در جدول (۱) تنظیمات استاندارد پارامترهای تابع محک قله‌های متحرک نشان داده شده‌است. تنظیمات مربوط به روش پیشنهادی در جدول (۲) آورده شده‌است.

(جدول-۱): تنظیمات استاندارد تابع محک قله‌های متحرک [۵۹]
(Table-1): Standard Setting for moving peaks benchmark [59]

پارامتر	مقدار
تعداد قله ها(M)	۱۰
تغییرات فرکانس(U)	۵۰۰۰
شدت قد(α)	۷.۰
شدت عرض(β)	۱.۰
شکل قله	Con
عملکرد پایه	No
(s) طول شیفت	۱.۰
(D) تعداد ابعاد	۵
ضریب همبستگی(λ)	۰
درصد تغییر قله ها	۱.۰
(S)محدوده مکان قله	[۰،۱۰۰]
(H) ارتفاع قله	[۳۰،۰،۷۰،۰]
ضخامت قله(W)	[۱،۱۲]
مقدار ارتفاع اولیه	۵۰.۰
مقدار ضخامت اولیه	۶.۰
تعداد محیط ها	۱۰۰

جدول ۲: تنظیمات استاندارد برای الگوریتم پیشنهادی
(Table-2): The standard setting for the proposed algorithm

پارامتر	مقدار
تعداد امپراتوری‌ها	۷۰
تعداد کلونی	۷
نرخ انقلاب	۰.۲
نرخ جذب	۲
تعداد دهه	۱۰۰
فاکتور دفع	۰.۵
کران بالا	۰
کران پایین	۱۰۰
معیار توقف الگوریتم	حداکثر تعداد ارزیابی‌ها

در این پژوهش تمام الگوریتم‌ها با استفاده از Matlab 2021 پیاده‌سازی و روی رایانهٔ شخصی با پردازندهٔ Core i7 و رم هشت گیگابایت اجرا شدند. الگوریتم پیشنهادی با FTmPSO(TMO)، RAmQSO-s4، RmNAFSA-s4، TFTmPSO، RFTmPSO، mQSO10 (5+5q)، FMSO، CellularPSO، Multi-SwarmPSO، AmQSO، FTMPSO، CDEPSO، و DPSABC مقایسه شده‌است.

جداول (۳)، (۴)، (۵) و (۶) میانگین خطای برون‌خطی روش پیشنهادی را در مقایسه با سایر روش‌ها با فرکانس تغییر پانصد، هزار، پنج‌هزار و ده‌هزار و همچنین تعداد قله‌های مختلف نشان می‌دهد. سایر پارامترهای مسئله به‌طور پیش‌فرض طبق طرح دوم برانک تنظیم می‌شوند. نتایج سایر روش‌ها از منابع هر روش آورده شده‌است. توجه داشته باشید که بهترین نتایج به‌صورت پررنگ نشان داده شده‌است.

شکل (۱۱) نشان می‌دهد که جمعیت می‌توانند تمام قله‌های موجود در فضای مسئله را پوشش دهند. این نشان می‌دهد که الگوریتم با استفاده از حافظهٔ پیشنهادی بیشترین تنوع را در فضای مسئله ایجاد کرده و درنهایت جمعیت با سرعت بالا هم‌گرا می‌شوند. با افزایش ابعاد، فضای جست‌وجو برای الگوریتم بسیار چالش‌برانگیز می‌شود و درنتیجه الگوریتم بهینه‌سازی باید بتواند کارایی قابل قبولی را در منظرهٔ شایستگی با ابعاد بزرگ نشان دهد. شکل (۱۲) نشان می‌دهد که الگوریتم توانسته هم در ابعاد کوچک (پنج بعد) و هم در ابعاد به نسبت بزرگ (سی بعد) عملکرد قابل توجهی داشته باشد.

شکل (۱۳) عملکرد روش پیشنهادی را در دو حالت، با استفاده از حافظهٔ استاندارد با استفاده از حافظهٔ خوشه‌ای پیشنهادی نشان می‌دهد. در این شکل مشاهده می‌شود که روش حافظهٔ خوشه‌ای خطای برون‌خطی بسیار کمتری نسبت به حافظهٔ استاندارد دارد. شدت تغییرات یکی از پارامترهای محیطی است که با افزایش این پارامتر قله‌های متحرک با شدت بیشتری تغییر می‌کنند و درنتیجه ردیابی آن‌ها برای الگوریتم مشکل می‌شود. شکل (۱۴) نشان می‌دهد که الگوریتم پیشنهادی با افزایش پارامتر شدت تغییر، نسبت به روش‌های پیشرفتهٔ دیگر عملکرد بهتری داشته است.

(جدول-۳): میانگین خطای برون‌خطی برای الگوریتم‌های مختلف در مسئلهٔ MPB با تعداد قله‌های مختلف و فرکانس پانصد

(Table-3): Average offline error for different algorithms on MPB problem with different numbers of peaks and frequency 500

تعداد قله	روش پیشنهادی	Mohamad pour et. al	mQSO10 (5+5q)	FMSO	Cellular PSO	CPSO	FTMPSO	DPSABC	Multi-SwarmPSO	AmQSO
۱	۱.۰۳(۰.۱۲)	۲.۸۵(۰.۲۲)	۳۳.۶۷(۳.۴)	۷.۵۸(۰.۹)	۱۳.۴۶(۰.۳)	۱۴.۲۵(-)	۱.۷۶(۰.۰۹)	۲.۷۷ (۰.۰۰)	۵.۴۶ (۰.۳۰)	۳.۰۲(۰.۳۲)
۵	۱.۸۰(۰.۱۴)	۳.۵۷(۰.۲۵)	۱۱.۹۱(۰.۷)	۹.۴۵(۰.۴)	۹.۶۳(۰.۴۹)	۳۶.۴۰(-)	۲.۹۳(۰.۱۸)	-	۵.۴۸ (۰.۱۹)	۵.۷۷(۰.۵۶)
۱۰	۱.۵۵(۰.۱۶)	۳.۹۶(۰.۲۱)	۹.۶۲(۰.۳۴)	۱۸.۲۶(۰.۳)	۹.۳۵(۰.۳۷)	۲۰.۹۱(-)	۳.۹۱(۰۱۹)	۳.۴۲ (۰.۰۰)	۵.۹۵ (۰.۰۹)	۵.۳۷(۰.۴۲)
۲۰	۱.۸۳(۰.۱۶)	۴.۰۵(۰.۱۸)	۹.۰۷(۰.۲۵)	۱۷.۳۴(۰.۳)	۸.۸۴(۰.۲۸)	۱۳.۱۱(-)	۴.۸۳(۰.۱۹)	۳.۱۲ (۰.۰۰)	۶.۴۵ (۰.۱۶)	۶.۸۲(۰.۳۴)
۳۰	۲.۰۶(۰.۱۸)	۴.۶۷(۰.۲۰)	۸.۸۰(۰.۲۱)	۱۶.۳۹(۰.۴)	۸.۸۱(۰.۲۴)	۱۰.۸۳(-)	۵.۰۵(۰.۲۱)	۳.۶۹ (۰.۰۰)	۶.۶۰ (۰.۱۴)	۷.۱۰(۰.۳۹)
۴۰	۲.۱۴(۰.۱۸)	۴.۹۵(۰.۱۵)	۸.۵۵(۰.۲۱)	۱۵.۳۴(۰.۴)	۸.۹۴(۰.۲۴)	۱۰.۱۲(-)	-	-	۶.۸۵ (۰.۱۳)	۷.۰۵(۰.۴۱)
۵۰	۲.۲۰(۰.۱۸)	۵.۲۳(۰.۱۷)	۸.۷۲(۰.۲۰)	۵.۵۴(۰.۲)	۸.۶۲(۰.۲۳)	۹.۲۸(-)	۴.۹۸(۰.۱۵)	۳.۳۲ (۰.۰۰)	۷.۰۴ (۰.۱۰)	۸.۹۷(۰.۳۲)
۱۰۰	۳.۱۰(۰.۲۰)	۵.۰۶(۰.۱۶)	۸.۵۴(۰.۱۶)	۲.۸۷(۰.۶)	۸.۵۴(۰.۲۱)	۷.۷۷(-)	۵.۳۱(۰.۱۱)	۳.۰۱ (۰.۰۰)	۷.۳۹ (۰.۱۳)	۷.۳۴(۰.۳۱)
۲۰۰	۳.۸۵(۰.۲۳)	۴.۸۱(۰.۱۳)	۸.۱۹(۰.۱۷)	۱۱.۵۲(۰.۶)	۸.۲۸(۰.۱۸)	۶.۸۳(-)	۵.۵۲(۰.۲۱)	۳.۱۶ (۰.۰۰)	۷.۵۲ (۰.۱۲)	۷.۴۸(۰.۱۹)

(جدول-۴): میانگین خطای برون‌خطی برای الگوریتم‌های مختلف در مسئلهٔ MPB با تعداد قله‌های مختلف و فرکانس هزار

(Table-4): Average offline errors for different algorithms on MPB problem with different numbers of peaks and frequency 1000

تعداد قله	روش پیشنهادی	Mohamadpour et. al	mQSO10 (5+5q)	FMSO	Cellular PSO	CPSO	AmQSO	Multi-SwarmPSO	mCPSO*
۱	۰.۵۵(۰.۸)	۱.۱۰(۰.۱۰)	۱۸.۶۰(۱.۳)	۱۴.۴۲(۰.۹)	۶.۷۷(۰.۳۸)	۸.۹۳(-)	۲.۳۳(۰.۳۱)	۲.۹۰ (۰.۱۸)	۴.۹۳(-)
۵	۰.۶۰(۰.۸)	۱.۱۲(۰.۱۱)	۶.۵۶(۰.۳۸)	۱۰.۵۹(۰.۴)	۵.۳۰(۰.۳۲)	۸.۶۲(-)	۲.۹۰(۰.۳۲)	۳.۳۵ (۰.۱۸)	۲.۰۷(-)
۱۰	۰.۸۰(۰.۹)	۱.۲۸(۰.۱۳)	۵.۷۱(۰.۲۲)	۱۰.۴۰(۰.۳)	۵.۱۵(۰.۱۹)	۷.۴۸(-)	۴.۵۶(۰.۴۰)	۳.۹۴ (۰.۰۸)	۲.۰۵(-)
۲۰	۰.۹۶(۰.۱۰)	۱.۷۶(۰.۹)	۵.۸۵(۰.۱۵)	۱۰.۳۳(۰.۳)	۵.۲۳(۰.۱۸)	۶.۱۰(-)	۵.۳۶(۰.۴۷)	۴.۳۳ (۰.۱۲)	۲.۹۵(-)
۳۰	۱.۰۳(۰.۱۲)	۲.۰۱(۰.۱۴)	۵.۸۱(۰.۱۵)	۱۰.۰۶(۰.۴)	۵.۳۳(۰.۱۶)	۵.۴۴(-)	۵.۲۰(۰.۳۸)	۴.۴۱ (۰.۱۱)	۳.۳۸(-)
۴۰	۱.۱۰(۰.۱۴)	۲.۲۳(۰.۱۶)	۵.۷۰(۰.۱۴)	۹.۸۵(۰.۴)	۵.۶۱(۰.۱۶)	۵.۵۷(-)	۵.۲۵(۰.۳۷)	۴.۵۲ (۰.۰۹)	-
۵۰	۱.۱۷(۰.۱۳)	۲.۵۶(۰.۱۰)	۵.۸۷(۰.۱۳)	۹.۵۴(۰.۲)	۵.۵۵(۰.۱۴)	۵.۱۷(-)	۶.۰۶(۰.۱۴)	۴.۵۷ (۰.۰۸)	۳.۶۸(-)
۱۰۰	۱.۵۰(۰.۱۵)	۲.۴۲(۰.۱۴)	۵.۸۳(۰.۱۳)	۸.۷۷(۰.۶)	۵.۵۷(۰.۱۲)	۴.۲۶(-)	۴.۷۷(۰.۴۵)	۴.۷۷ (۰.۰۸)	۳.۹۳(-)
۲۰۰	۱.۸۴(۰.۱۵)	۲.۲۰(۰.۱۱)	۵.۵۴(۰.۱۱)	۸.۰۶(۰.۶)	۵.۵۰(۰.۱۲)	۳.۷۴(-)	۵.۷۵(۰.۲۶)	۴.۷۶ (۰.۰۷)	۳.۸۶(-)

(جدول-۵): میانگین خطای برون‌خطی برای الگوریتم‌های مختلف در مسئلهٔ MPB با تعداد قله‌های مختلف و فرکانس پنج هزار

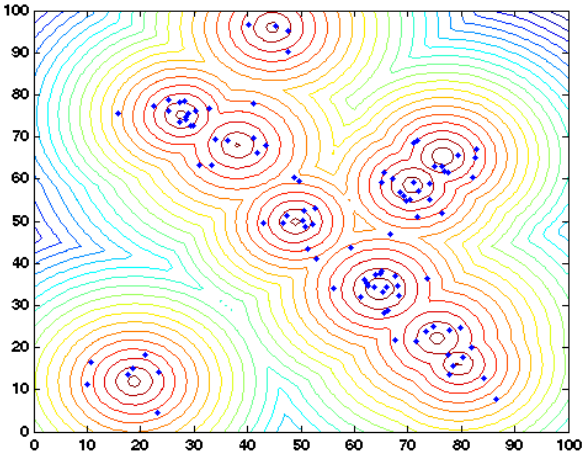
(Table-5): Average offline errors for different algorithms on MPB Problem with different numbers of peaks and frequency 5000

تعداد قله	روش پیشنهادی	Mohamadpour et. al	mQSO10 (5+5q)	FMSO	Cellular PSO	CPSO	FTMPSO	DPSABC	Multi-SwarmPSO	Adaptive mQSO	mCPSO*
۱	۰.۱۰(۰.۰۰)	۰.۹۲(۰.۰۹)	۳.۸۲(۰.۳۵)	۳.۴۴(۰.۱)	۲.۵۴(۰.۱)	۰.۱۴(۰.۱۱)	۰.۱۸ (۰۰۱)	۳.۲۵ (۰.۰۰)	۰.۹۰ (۰.۰۵)	۰.۵۱(۰.۰)	۴.۹۳(-)
۵	۰.۱۴(۰.۰۰)	۱.۰۶(۰.۷)	۱.۹۰(۰.۰۸)	۲.۹۴(۰.۰)	۱.۷۲(۰.۱)	۰.۷۲(۰.۷۲)	۰.۴۷ (۰.۰۵)	-	۱.۲۱ (۰.۱۲)	۱.۰۱(۰.۰)	۲.۰۷(-)
۱۰	۰.۱۷(۰.۰۰)	۱.۱۵(۰.۱۰)	۱.۹۱(۰.۰۸)	۳.۱۱(۰.۰)	۱.۷۶(۰.۱)	۱.۰۵(۰.۲۴)	۰.۶۷ (۰.۰۴)	۲.۱۳ (۰.۰۰)	۱.۶۶ (۰.۰۸)	۱.۵۱(۰.۱)	۲.۰۵(-)
۲۰	۰.۳۰(۰.۰۱)	۱.۱۸(۰.۰۶)	۲.۵۶(۰.۱۰)	۳.۳۶(۰.۰)	۲.۵۹(۰.۱)	۱.۵۹(۰.۲۲)	۰.۹۳ (۰.۰۴)	۲.۰۷ (۰.۰۰)	۲.۰۵ (۰.۰۸)	۲.۰۰(۰.۱)	۲.۹۵(-)
۳۰	۰.۳۷(۰.۰۱)	۱.۳۵(۰.۰۵)	۲.۶۸(۰.۱۰)	۳.۲۸(۰.۰)	۲.۹۵(۰.۱)	۱.۵۸(۰.۱۷)	۱.۱۴ (۰.۰۴)	۱.۸۸(۰.۰۰)	۲.۱۸ (۰.۰۶)	۲.۱۹(۰.۱)	۳.۳۸(-)
۴۰	۰.۵۴(۰.۰۰)	۱.۵۳(۰.۰۹)	۲.۶۵(۰.۰۸)	۳.۲۶(۰.۰)	۳.۱۱(۰.۱)	۱.۵۱(۰.۱۲)	-	-	۲.۲۳ (۰.۰۶)	۲.۲۸(۰.۱)	-
۵۰	۰.۶۳(۰.۰۵)	۱.۶۵(۰.۰۷)	۲.۶۳(۰.۰۸)	۳.۲۲(۰.۰)	۳.۲۲(۰.۱)	۱.۵۴(۰.۱۲)	۱.۳۲ (۰.۰۴)	۱.۹۱ (۰.۰۰)	۲.۳۰ (۰.۰۴)	۲.۴۳(۰.۱)	۳.۶۸(-)
۱۰۰	۰.۸۲(۰.۰۷)	۱.۸۰(۰.۰۶)	۲.۵۲(۰.۰۶)	۳.۰۶(۰.۰)	۳.۳۹(۰.۱)	۱.۴۱(۰.۰۸)	۱.۶۱ (۰.۰۳)	۱.۸۹ (۰.۰۰)	۲.۳۲ (۰.۰۴)	۲.۶۸(۰.۱)	۳.۹۳(-)
۲۰۰	۱.۱۰(۰.۰۷)	۱.۷۱(۰.۰۵)	۲.۳۰(۰.۰۵)	۲.۸۴(۰.۰)	۳.۳۶(۰.۰)	۱.۲۴(۰.۰۶)	۱.۶۷ (۰.۰۳)	۱.۸۷ (۰.۰۰)	۲.۳۴ (۰.۰۳)	۲.۶۲(۰.۱)	۳.۸۶(-)

(جدول-۶): میانگین خطای برون‌خطی برای الگوریتم‌های مختلف در مسئلهٔ MPB با تعداد قله‌های مختلف و فرکانس ده هزار

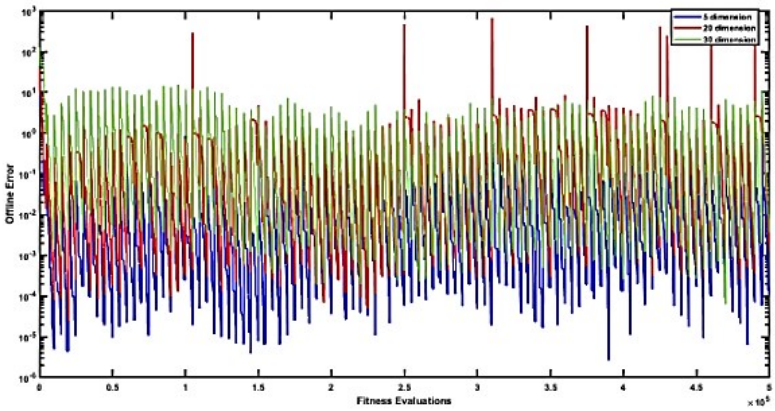
(Table-6): Average offline errors for different algorithms on MPB Problem with different numbers of peaks and frequency 10000

تعداد قله	روش پیشنهادی	mQSO10 (5+5q)	FMSO	Cellular PSO	FTMPSO	DPSABC	Multi Swarm PSO	AmQSO
۱	۰.۰۳(۰.۱۰)	۱.۹۰ ±۰.۱۸	۱.۹۰ ±۰.۰۶	۱.۵۳ ±۰.۱۲	۰.۰۹ ±۰.۰۰	۲.۶۷ ±۰.۰۰	۰.۲۷ ±۰.۰۲	۰.۱۹ ±۰.۰۲
۲	۰.۰۵(۰.۰۸)	۱.۰۳ ±۰.۰۶	۱.۷۵ ±۰.۰۶	۰.۹۲ ±۰.۱۰	۰.۳۱ ±۰.۰۴	-	۰.۷۰ ±۰.۱۰	۰.۴۵ ±۰.۰۴
۱۰	۰.۰۷(۰.۰۶)	۱.۱۰ ±۰.۰۷	۱.۹۱ ±۰.۰۴	۱.۱۹ ±۰.۰۷	۰.۴۳ ±۰.۰۳	۹.۰۱ ±۰.۰۱	۰.۹۷ ±۰.۰۴	۰.۷۶ ±۰.۰۶
۲۰	۰.۱۰(۰.۰۷)	۱.۸۴ ±۰.۰۸	۲.۱۶ ±۰.۰۴	۲.۲۰ ±۰.۱۰	۰.۵۶ ±۰.۰۱	۶.۶۰ ±۰.۰۱	۱.۳۴ ±۰.۰۸	۱.۲۸ ±۰.۱۲
۳۰	۰.۱۲(۰.۰۵)	۲.۰۰ ±۰.۰۹	۲.۱۸ ±۰.۰۴	۲.۶۰ ±۰.۱۳	۰.۶۹ ±۰.۰۹	۷.۷۰ ±۰.۰۱	۱.۴۳ ±۰.۰۵	۱.۷۸ ±۰.۰۹
۴۰	۰.۱۴(۰.۰۴)	۱.۹۹ ±۰.۰۷	۲.۲۱ ±۰.۰۳	۲.۷۳ ±۰.۱۱	-	-	۱.۴۷ ±۰.۰۶	-
۵۰	۰.۱۵(۰.۰۳)	۱.۹۹ ±۰.۰۷	۲.۶۰ ±۰.۰۸	۲.۸۴ ±۰.۱۲	۰.۸۶ ±۰.۰۲	۸.۱۰ ±۰.۰۱	۱.۴۷ ±۰.۰۴	۱.۵۵ ±۰.۰۸
۱۰۰	۰.۲۵(۰.۰۳)	۱.۸۵ ±۰.۰۵	۲.۲۰ ±۰.۰۳	۲.۹۳ ±۰.۰۹	۱.۰۸ ±۰.۰۱	۸.۳۴ ±۰.۰۱	۱.۵۰ ±۰.۰۳	۱.۸۹ ±۰.۱۴
۲۰۰	۰.۴۰(۰.۰۳)	۱.۷۱ ±۰.۰۴	۲.۰۰ ±۰.۰۲	۲.۸۸ ±۰.۰۷	-	-	-	-



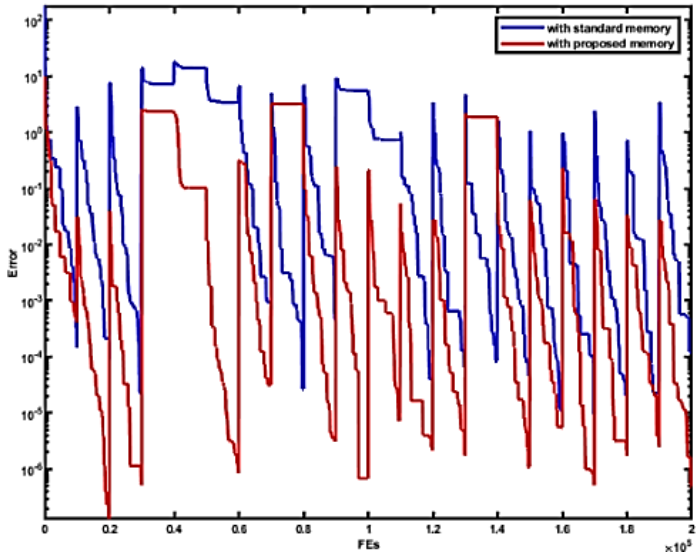
(شکل -۱۱): پوشش قله‌های متحرک توسط جمعیت

(Figure-11): Coverage of moving peaks by population

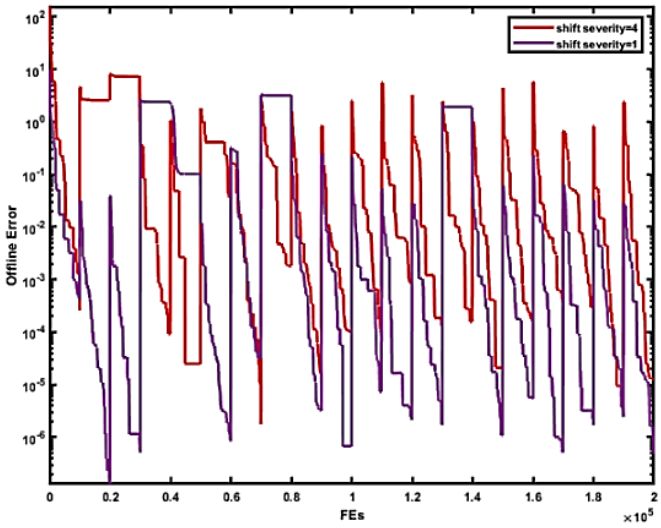


(شکل-۱۲): خطای برون‌خطی روش پیشنهادی برای ابعاد پنج، بیست و سی

Figure (12): Offline error of the proposed method for dimensions of sizes 5, 20 and 30



(شکل-۱۳): خطای برون‌خطی روش پیشنهادی با حافظهٔ استاندارد و حافظهٔ خوشه‌ای پیشنهادی
(Figure-13): Offline error of the proposed method with standard memory and clustered memory



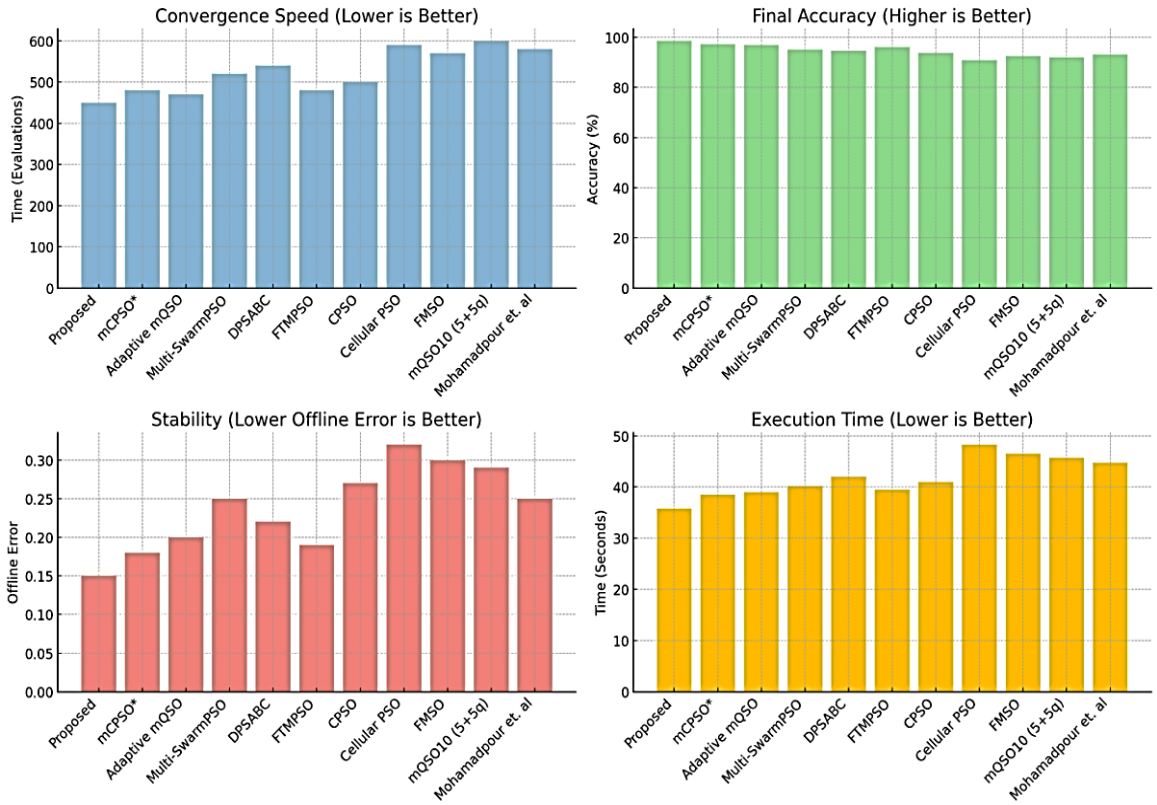
(شکل-۱۴): خطای برون‌خطی روش پیشنهادی با شدت تغییرات یک و چهار
(Figure-14): Offline error of the proposed method with shift severity=1 and shift severity=4

شکل (۱۵) نتایج مقایسهٔ عملکرد الگوریتم پیشنهادی را در مقایسه با سایر روش‌های بهینه‌سازی نشان می‌دهد. این مقایسه بر اساس چهار معیار کلیدی صورت گرفته‌است که در ادامه به صورت دقیق‌تر تحلیل می‌شوند:

سرعت هم‌گرایی: الگوریتم پیشنهادی با استفاده از ترکیب سازوکار حافظه و خوشه‌بندی توانسته است فرایند جست‌وجو را تسریع کند. این ویژگی باعث شده‌است که الگوریتم در مدت زمان کوتاه‌تری به نقاط بهینه برسد؛ در حالی که روش‌های دیگر مانند Multi-SwarmPSO و mQSO10 عملکرد کندتری داشته‌اند.

دقت نهایی: دقت نهایی به‌عنوان یکی از معیارهای کلیدی در مسائل بهینه‌سازی، به میزان نزدیکی راه‌حل نهایی به مقدار بهینه واقعی اشاره دارد. نتایج نشان می‌دهند که الگوریتم پیشنهادی دقت بالاتری در مقایسه با سایر روش‌ها ازجمله DPSABC و FTMPSO داشته است.

پایداری: پایداری الگوریتم به توانایی آن در کاهش خطاهای برون‌خطی و مدیریت تغییرات محیطی اشاره دارد. الگوریتم پیشنهادی کمترین خطای برون‌خطی را در تمامی طرح‌ها نشان داده‌است که بیان‌کنندهٔ توانایی بالا در تطبیق با تغییرات محیط‌های پویا است.



(شکل-۱۵): مقایسهٔ عملکرد الگوریتم پیشنهادی با روش‌های دیگر در معیارهای مختلف

(Figure-15): Performance comparison of the proposed algorithm with other methods across different metrics

زمان اجرا: یکی دیگر از مزایای الگوریتم پیشنهادی، زمان اجرای مطلوب آن است. این الگوریتم با حفظ تعادل بین اکتشاف و بهره‌برداری توانسته‌است پیچیدگی محاسباتی را کاهش دهد. در مقایسه با روش‌های mCPSO* و CellularPSO الگوریتم پیشنهادی زمان اجرا را بهبود بخشیده‌است.

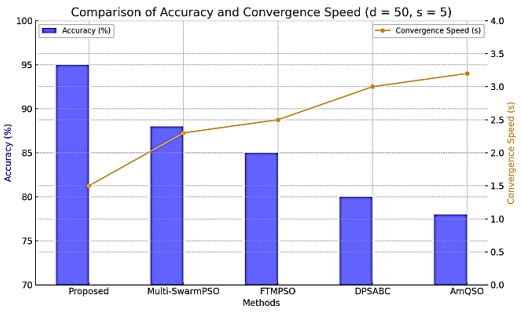
نتایج و تحلیل روش پیشنهادی با معیار GMPB: معیار GMPB یک فضای مسئلهٔ پیچیده‌تر و پویاتر از MPBارائه می‌دهد و برای ارزیابی عملکرد الگوریتم‌های بهینه‌سازی در شرایط پویا و با ابعاد بزرگ ایده‌آل است. روش پیشنهادی با چندین الگوریتم پیشرفته در تنظیمات مختلف GMPB، طبق جدول (۷) مورد ارزیابی قرار گرفته‌است. نتایج در شکل‌های (۱۶) تا (۱۹) ارائه شده‌است.

دقت و سرعت هم‌گرایی: روش پیشنهادی به‌طور مداوم در دقت عملکرد، به‌ویژه در طرح‌های با ابعاد بالا مانند $d = 50$ ، بهتر از سایر روش‌ها عمل کرده‌است. سازوکار حافظه در روش پیشنهادی به خوبی اطلاعات کلیدی محیط‌های پیشین را حفظ کرده و امکان هم‌گرایی سریع‌تر به نقاط بهینه را فراهم آورده‌است. این عملکرد به ویژه در شدت تغییرات بالا ($s = 5$) مشهود بود؛ زیرا روش پیشنهادی توانست به سرعت با تغییرات محیطی تطبیق یابد. شکل (۱۶) مقایسه‌ای از دقت (به درصد) و سرعت هم‌گرایی (به ثانیه) روش پیشنهادی و سایر روش‌ها را نشان می‌دهد؛ همان‌طورکه مشخص است، روش پیشنهادی بالاترین دقت را با سریع‌ترین زمان هم‌گرایی ارائه می‌دهد، که برتری آن را در شرایط با ابعاد بالا ($d = 50$) و تغییرات شدید ($s = 5$) تأیید می‌کند.

پایداری در برابر بی‌قاعدگی‌ها: معیار GMPB با اعمال بی‌قاعدگی‌هایی مانند شدت ارتفاع، عرض و زاویه، فضای مسئله را پیچیده‌تر می‌کند. روش پیشنهادی در برابر این بی‌قاعدگی‌ها ($s = 5$ و $\tau = 0.2$) عملکرد پایداری از خود نشان داده‌است. استفاده از استراتژی خوشه‌بندی و به‌روزرسانی حافظه در الگوریتم موجب حفظ تنوع جمعیت شده و از گیرافتادن در بهینه‌های محلی جلوگیری کرده‌است. شکل (۱۷) نشان می‌دهد که روش پیشنهادی در برابر بی‌قاعدگی‌هایی مانند شدت ارتفاع، عرض و زاویه، عملکرد پایدارتری نسبت به سایر روش‌ها داشته‌است. استفاده از سازوکار حافظه و استراتژی خوشه‌بندی کمک کرده‌است تا تنوع جمعیت حفظ شود و از گیرافتادن در بهینه‌های محلی جلوگیری شود.

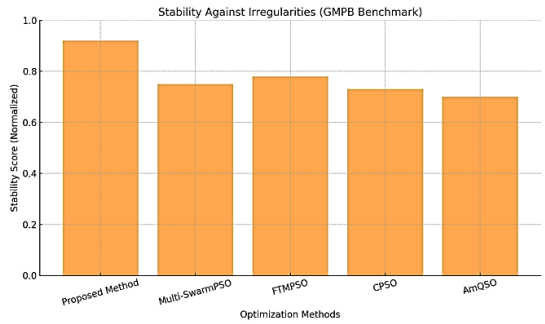
مقیاس‌پذیری : روش پیشنهادی با افزایش تعداد اجزا و ابعاد مسئله به خوبی مقیاس‌پذیری خود را حفظ کرده‌است. مدیریت بهینهٔ منابع محاسباتی و استفادهٔ کارآمد از ساختار حافظه باعث شده‌است که حتی در شرایط افزایش پیچیدگی مسئله، عملکرد الگوریتم کاهش نیابد. شکل (۱۸) نشان‌دهندهٔ عملکرد مقیاس‌پذیری روش پیشنهادی در مقایسه با سایر روش‌ها است. همان طور که مشاهده می‌شود، روش پیشنهادی در تمامی ابعاد نسبت به سایر الگوریتم‌ها عملکرد بهتری داشته‌است.

تأثیر فرکانس تغییرات (CF): در شرایط تغییر فرکانس‌های مختلف روش پیشنهادی به خوبی با تغییرات سریع و آهسته سازگار شده‌است. راهکار بازیابی و جایگزینی حافظه در این الگوریتم باعث شده‌است که جمعیت با کارایی بیشتری به تغییرات فرکانس‌های مختلف پاسخ دهند. شکل (۱۹) تأثیر فرکانس تغییرات (CF) بر عملکرد الگوریتم‌ها را نشان می‌دهد. الگوریتم پیشنهادی در تمامی فرکانس‌ها عملکرد بهتری از خود نشان داده و مداوم نسبت به سایر روش‌ها، دقت بیشتری دارد. این نشان‌دهندهٔ انطباق بالای الگوریتم پیشنهادی با تغییرات محیطی سریع و آهسته است.



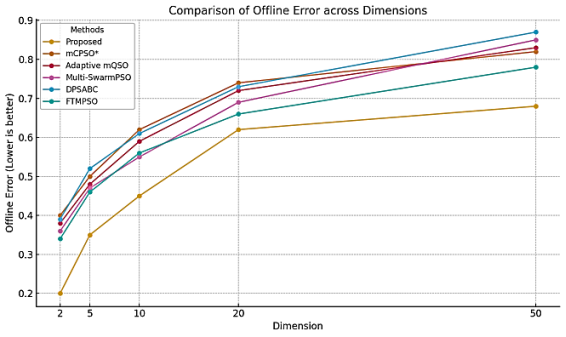
(شکل-۱۶): مقایسهٔ سرعت هم‌گرایی الگوریتم پیشنهادی با روش‌های دیگر در معیار GMPB

(Figure-16): Comparison of the convergence speed of the proposed algorithm with other methods based on the GMPB benchmark



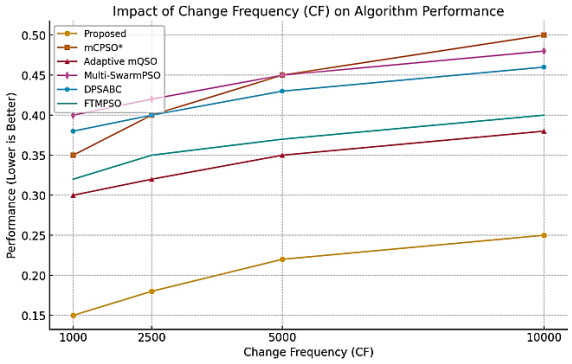
(شکل-۱۷): مقایسهٔ پایداری الگوریتم پیشنهادی با روش‌های دیگر در معیار GMPB

(Figure-17): Comparison of the stability of the proposed algorithm with other methods based on the GMPB benchmark



(شکل-۱۸): مقایسهٔ مقیاس‌پذیری الگوریتم پیشنهادی با روش‌های دیگر در معیار GMPB

(Figure 18): Comparison of the Scalability of the Proposed Algorithm with Other Methods Based on the GMPB Benchmark.



(شکل-۱۹): تأثیر فرکانس تغییرات (CF) بر عملکرد الگوریتم‌ها در معیار GMPB

(Figure-19): The impact of change frequency (CF) on the performance of algorithms based on the GMPB benchmark

جدول (۸) مقایسهٔ آماری شش روش را با یکدیگر نشان می‌دهد. برای مقایسهٔ آماری الگوریتم‌ها از آزمون t استفاده شده‌است. در این آزمون، از ۹۸ درجه آزادی با سطح معناداری ۰.۵ استفاده شد. نوع آزمون به صورت زوجی و یک‌طرفه تعیین شد. پس از انجام آزمایش، نتایج به صورت زیر ارائه شده‌است: اگر الگوریتم نخست به‌طور معناداری بهتر از الگوریتم دوم باشد، از نماد «+» استفاده می‌شود. اگر الگوریتم نخست به‌طور معناداری ضعیف‌تر از الگوریتم دوم باشد، از نماد «-» استفاده می‌شود. اگر شواهد معناداری برای برتری هیچ‌یک از الگوریتم‌ها وجود نداشته باشد، از نماد «~» استفاده می‌شود. گفتنی است که برای آزمون معناداری آماری، روش پیشنهادی سی‌بار اجرا شده‌است. در این آزمون نماد + نشان‌دهنده تفاوت آماری معنادار است. نماد − نشان‌دهندهٔ نبود تفاوت آماری معنادار است و نماد ~ نشان‌دهندهٔ نبود تفاوت آماری معنادار بین دو الگوریتم است. در این تحلیل، روش پیشنهادی در بسیاری از موارد برتری معناداری نسبت به الگوریتم‌های دیگر نشان داده‌است. در برخی موارد، تفاوت‌ها معنادار نیست و (نماد ~) یا الگوریتم دوم بهتر عمل کرده‌است.

(جدول-۷): تنظیمات استاندارد تابع محک قله‌های متحرک [۲۴]
(Table-7): The set of Generalized Moving Peaks Benchmark (GMPB) parameters[24]

پارامتر	نماد	مقدار
بعد	d	۲,۵,۱۰,۲۰, ۵۰
شدت شیفت	$\tilde{s}$	۱,۲,۵
تعداد اجزا	m	۱,۵,۱۰ ,۲۵,۵۰,۱۰۰,۲۰۰
شدت زاویه	$\tilde{\theta}$	$\pi/۹$
شدت ارتفاع	$\tilde{h}$	۷
شدت ضخامت	$\tilde{w}$	۱
شدتτ پارامتر بی نظمی	$\tilde{\tau}$	۰.۲
شدت ηپارامتر بی نظمی	$\tilde{\eta}$	۲
محدودهٔ جست‌وجو	$[Lb, Ub]^d$	$[-۱۰۰,۱۰۰]^d$
ارتفاع جست‌وجو	$[h_{min}, h_{max}]$	[۳۰,۷۰]
عرض جست‌وجو	$[w_{min}, w_{max}]^d$	$[۱,۱۲]^d$
محدودهٔ زاویه	$[\theta_{min}, \theta_{max}]$	$[-\pi, \pi]$
پارامتر بی نظمی τ محدوده	$[\tau_{min}, \tau_{max}]$	$[-۱, ۱]$
پارامتر بی نظمیη محدوده	$[\eta_{min}, \eta_{max}]$	$[-۲۰, ۲۰]$
موقعیت مرکزی اولیه	$c_k^{(0)}$	$\mathcal{U}[Lb, Ub]^d$
ارتفاع اولیه	$h_k^{(0)}$	$\mathcal{U}[h_{min}, h_{max}]$
ضخامت اولیه	$w_k^{(0)}$	$\mathcal{U}[w_{min}, w_{max}]^d$
محدودهٔ اولیه	$\theta_k^{(0)}$	$\mathcal{U}[\theta_{min}, \theta_{max}]$
τ پارامتر بی نظمی اولیه	$\tau_k^{(0)}$	$\mathcal{U}[\tau_{min}, \tau_{max}]$
η پارامتر بی نظمی اولیه	$\eta_k^{(0)}$	$\mathcal{U}[\tau_{min}, \tau_{max}]$
ماتریس چرخش اولیه	$\mathrm{R}_k^{(0)}$	$GS(N(۰,۱)^{d\times d})^{\mathrm{t}}$
تغییرات فرکانس	CF	۱۰۰۰,۲۵۰۰,۵۰۰۰ ,۱۰۰۰۰
تعداد محیط ها	T	۱۰۰

(جدول-۸): تحلیل آماری آزمون t برای مقایسهٔ الگوریتم‌ها
(Table-8): Statistical Analysis of t-test for Algorithm Comparison

جفت الگوریتم‌ها	فرکانس ۵۰۰	فرکانس ۱۰۰۰	فرکانس ۱۵۰۰	فرکانس ۵۰۰۰	فرکانس ۱۰۰۰۰
mQSO, AmQSO	+	−	−	−	~
❋mQSO, mCPSO	−	−	−	−	+
mQSO, FMSO	−	−	~	+	−
mQSO, CellularPSO	+	−	−	+	~
❋AmQSO, mCPSO	+	−	−	+	~
AmQSO, FMSO	−	−	+	−	+
AmQSO, CellularPSO	−	−	~	+	−
Proposed, AmQSO	−	−	−	−	+
❋Proposed, mCPSO	+	+	+	+	+
Proposed, CPSO	+	+	+	+	+
Proposed, CellularPSO	+	+	+	+	+
Proposed, FMSO	+	−	+	+	+

۵-نتیجه‌گیری

مسائل بهینه‌سازی پویا به‌دلیل ارتباط مستقیم با کاربردهای متعدد دنیای واقعی، همواره یکی از موضوعات مهم و چالش‌برانگیز در زمینهٔ پژوهشی علوم مهندسی و محاسباتی محسوب می‌شوند. این مسائل به‌دلیل تغییرات پیوسته و شرایط پویا در محیط، نیازمند رویکردهایی هستند که بتوانند به طور مؤثر با تغییرات سازگار شوند و هم‌زمان تنوع کافی را در جمعیت راه‌حل‌ها حفظ کنند. نمونه‌هایی از این کاربردها شامل زمان‌بندی کارهای پویا، مسیریابی بهینه در شبکه‌های حس‌گر بی‌سیم و مسائل پیچیدهٔ تصمیم‌گیری در محیط‌های نامطمئن است.

یکی از رویکردهای پرکاربرد در این حوزه، الگوریتم‌های رقابت استعماری هستند که با الهام از فرایندهای تاریخی و سیاسی استعمار و جذب، به‌عنوان یکی از الگوریتم‌های تکاملی کارآمد شناخته می‌شوند؛ بااین‌حال، این الگوریتم‌ها در مواجهه با مسائل پویا با چالش‌های متعددی روبه‌رو هستند؛ از جمله کاهش تنوع جمعیت، افت کارایی در شرایط تغییرات سریع محیط و محدودیت در هم‌گرایی بهینه. این موارد، لزوم توسعهٔ نسخه‌های بهبودیافته و تطبیق‌پذیرتر این الگوریتم‌ها را نشان می‌دهند. در این مقاله، با درک دقیق این چالش‌ها، یک الگوریتم رقابت استعماری بهبودیافته و سازگار با محیط‌های پویا معرفی شده‌است. این الگوریتم با استفاده از مفاهیمی نظیر حافظه، خوشه‌بندی جمعیت و سازوکار دافعه توانسته است تنوع جمعیت را در تمام مراحل حفظ کند و در عین حال سرعت هم‌گرایی را در مواجهه با تغییرات محیطی افزایش دهد. ویژگی کلیدی الگوریتم پیشنهادی استفاده از حافظه برای ذخیرهٔ راه‌حل‌های بهینهٔ گذشته، مکانیزم خوشه‌بندی برای مدیریت تنوع جمعیت و دافعه برای جلوگیری از تجمع غیرضروری راه‌حل‌ها در مناطق خاص است. این سه مؤلفه هماهنگ عمل می‌کنند تا الگوریتم بتواند در مواجهه با تغییرات پیچیده و غیرمنتظرهٔ محیطی، عملکردی پایدار و کارآمد ارائه دهد.

نتایج شبیه‌سازی‌ها و آزمایش‌های انجام‌شده در این پژوهش نشان داد که الگوریتم پیشنهادی در مقایسه با سایر روش‌های موجود عملکرد بهتری دارد. در مقایسه با الگوریتم‌های مشابه، الگوریتم پیشنهادی توانست با حفظ تنوع بیشتر در جمعیت، بهبود قابل‌توجهی در کاهش خطای برون‌خطی ارائه دهد. این ویژگی موجب شد که الگوریتم پیشنهادی بتواند کمابیش تمام قله‌های فضای جست‌وجو را پوشش دهد و از افتادن در بهینه‌های محلی جلوگیری کند؛ همچنین نتایج نشان داد که این الگوریتم در فرکانس‌های مختلف تغییرات محیطی، توانایی بیشتری در انطباق با شرایط پویا و دستیابی سریع‌تر به راه‌حل‌های بهینه دارد.

برتری الگوریتم پیشنهادی به وضوح در کاهش خطاهای برون‌خطی و بهبود سرعت هم‌گرایی مشاهده شد. این الگوریتم توانست در زمان کمتری به نتایج قابل قبولی برسد و کارایی خود را در محیط‌های پویا با تغییرات شدید و یا جزئی نشان دهد؛ همچنین استفاده از راه‌کار خوشه‌بندی، علاوه‌بر مدیریت مؤثر تنوع، به جلوگیری از هم‌گرایی زودهنگام کمک کرد و باعث شد که جمعیت راه‌حل‌ها همچنان قابلیت اکتشاف در فضای جست‌وجو را حفظ کنند. این دستاوردها نشان‌دهندهٔ توانایی الگوریتم پیشنهادی در حل مسائل پویا با ابعاد بالا و پیچیدگی‌های ذاتی بیشتر است. با توجه به نتایج به‌دست‌آمده می‌توان نتیجه گرفت که الگوریتم پیشنهادی یک ابزار قدرتمند و مؤثر برای حل مسائل بهینه‌سازی پویا است که می‌تواند به‌طور معناداری بهبود عملکرد و کارایی سامانه‌های بهینه‌سازی را تضمین کند. این الگوریتم با حفظ تعادل مناسب بین اکتشاف و بهره‌برداری، توانایی بالایی در سازگاری با تغییرات محیطی دارد و این ویژگی، آن را به گزینه‌ای مناسب برای کاربردهای دنیای واقعی تبدیل می‌کند؛ درنهایت، با توجه به قابلیت‌های این الگوریتم پیشنهاد می‌شود که پژوهش‌های آینده بر روی توسعه و گسترش این رویکرد در حوزه‌هایی نظیر زمان‌بندی پویا، مدیریت انرژی در سامانه‌های هوشمند، مسیریابی در شبکه‌های حس‌گر بی‌سیم و سایر مسائل مشابه تمرکز کنند؛ همچنین استفاده از روش‌های پیش‌بینی برای تغییرات محیطی و تلفیق آن با حافظه و خوشه‌بندی می‌تواند مسیرهای جدیدی را برای بهبود عملکرد الگوریتم فراهم کند؛ علاوه‌بر این، بررسی نحوهٔ تعامل این الگوریتم با دیگر روش‌های یادگیری ماشین، مانند شبکه‌های عصبی و الگوریتم‌های یادگیری تقویتی، می‌تواند کاربردهای جدیدی برای آن در مسائل دنیای واقعی ایجاد کند.

6-Reference ۶-مراجع

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

[2] J. Branke. Evolutionary Optimization in Dynamic Environments. Springer US, 2002.

[3] T. T. Nguyen, Z. Yang, and S. Bonsall. Dynamic time-linkage problems - the challenges. In Computing and Communication Technologies, Research, *Innovation, and Vision for the Future (RIVF)*, pages 1–6. IEEE, 2012b. doi: 10.1109/rivf.2012.6169823.

[4] M. N. Omidvar, B. Kazimipour, X. Li, and X. Yao. CBCC3 – a contribution-based cooperative co-evolutionary algorithm with improved exploration/exploitation balance. In Evolutionary Computation (CEC), 2016 *IEEE Congress on*, pages 3541–3548. IEEE, 2016.

[5] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. Time series analysis: forecasting and control. Wiley, 2015.

[6] E. Atashpaz-Gargari and C. Lucas, "Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition," *2007 IEEE Congress on Evolutionary Computation*, 2007, pp. 4661-4667, doi: 10.1109/CEC.2007.4425083.

[7] H. Nakano, M. Kojima, and A. Miyauchi, "An artificial bee colony algorithm with a memory scheme for dynamic optimization problems," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '15)*, pp. 2657–2663, IEEE, May 2015.

https://doi.org/10.1007/s10489-018-1197-z, 2018.
[21] M. Mohammadpour, H. Parvin, M. Sina "Chaotic Genetic Algorithm based on Explicit Memory with a New Strategy for Updating and Retrieval of Memory in Dynamic Environments," In Journal of AI and Data Mining. (Vol 6, No 1), Pages, 191-205, 2018.
[22] X. Chen, D. Zhang and X. Zeng, "Matching-Based Selection and Memory Enhanced MOEDA/D for Evolutionary Dynamic Multiobjective Optimization," Tools with Artificial intelligence (ICTAI), 2015 IEEE 27th International Conference on, pages 478 485.
[23] Yang S (2008) Genetic algorithms with memory- and elitism-based immigrants in dynamic environments. Evol Comput 16(3):385 416.
[24] D. Yazdani., R. Cheng., D. Yazdani, J. Branke, Y. Jin., & X. Yao. A Survey of Evolutionary Continuous Dynamic Optimization Over Two Decades—Part A. In IEEE Transactions on Evolutionary Computation (Vol. 25, Issue 4, pp. 609–629). Institute of Electrical and Electronics Engineers (IEEE), 2021. https://doi.org/10.1109/tevc.2021.3060014.
[25] B. Niu, Q. Liu, and J. Wang, "Bacterial foraging optimization with memory and clone schemes for dynamic environments," in Advances in Swarm Intelligence, Y. Tan et al., Ed. Springer International Publishing, 2019, pp. 352–360.
[26] Y. Bravo, G. Luque, and E. Alba, "Global memory schemes for dynamic optimization," Natural Computing, vol. 15, no. 2, pp. 319–333, 2015.
[27] M. Moradi, S. Nejatian, H. Parvin and V. Rezaie, "CMCABC: Clustering and Memory-Based Chaotic Artificial Bee Colony Dynamic Optimization Algorithm", International Journal of Information Technology & Decision MakingVol. 17, No. 04, pp. 1007-1046, 2018.
[28] Yang, S., and Li, C., "A General Framework of Multipopulation Methods with Clustering in Undetectable Dynamic Environments". IEEE Transactions on Evolutionary Computation, vol. 16, no. 4, pp. 556-577, 2012.
[29] B. Niu, Q. Liu, and J. Wang, "Bacterial foraging optimization with memory and clone schemes for dynamic environments," in Advances in Swarm Intelligence, Y. Tan et al., Ed. Springer International Publishing, 2019, pp. 352–360.
[30] Y. Bravo, G. Luque, and E. Alba, "Global memory schemes for dynamic optimization," Natural Computing, vol. 15, no. 2, pp. 319–333, 2015.
[31] S. A. van der Stockt and A. P. Engelbrecht, "Analysis of selection hyper-heuristics for population-based meta-heuristics in real-valued dynamic optimization," Swarm Evol. Comput., vol. 43, pp. 127–146, 2018.
[32] W. Zhang, M. Zhang, W. Zhang, Y. Meng, and H. Wu, "Innate-adaptive response and
[8] T. Blackwell and J. Branke, "MultiswarmT Exclusion, and Anti-Convergence in Dynamic Environment", 2006.
[9] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.
[10] S. Yang, "Memory-based immigrants for genetic algorithms in dynamic environments," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '05)*, pp. 1115–1122, ACM, June 2005.
[11] R. Vafashoar and M. R. Meybodi, "A multi-population differential evolution algorithm based on cellular learning automata and evolutionary context information for optimization in dynamic environments," Appl. Soft Comput., p. 106009, 2019.
[12] M. Yasrebi, A. Eskandar-Baghban, H. Parvin, M. Mohammadpour, "Optimisation inspiring from behaviour of raining in nature: droplet optimisation algorithm," International Journal of Bio-Inspired Computation, vol. 12, no. 3, pp. 152-163, 2018.
[13] D. Yazdani, T. T. Nguyen, J. Branke, and J. Wang. A new multi swarm particle swarm optimization for robust optimization over time. In G. Squillero and K. Sim, editors, Applications of Evolutionary Computation, volume 10200, pages 99{109. Springer Lecture Notes in Computer Science, 2017.
[14] D. Yazdani, T. T. Nguyen, J. Branke, and J. Wang. A new multi swarm particle swarm optimization for robust optimization over time. In G. Squillero and K. Sim, editors, Applications of Evolutionary Computation, volume 10200, pages 99–109. Springer Lecture Notes in Computer Science, 2017.
[15] D. Yazdani, T. T. Nguyen, and J. Branke. Robust optimization over time by learning problem space characteristics. IEEE Transactions on Evolutionary Computation, 2018a.
[16] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in Proc. Congr. Evol. Comput., vol. 3. 1999, pp. 1875–1882.
[17] S. Yang, "Associative memory scheme for genetic algorithms in dynamic environments," in Proc. EvoWorkshops: Appl. Evol. Comput., LNCS 3907. 2006, pp. 788–799.
[18] S. Yang, "Genetic algorithms with memory and elitism-based immigrants in dynamic environment," Evol. Comput., vol. 16, no. 3, pp. 385–416, 2008.
[19] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems", *In IEEE Congress on Evolutionary Computation*, pages 1875–1882. IEEE, 1999. doi: 10.1109/CEC.1999.785502.
[20] H., Parvin, S., Nejatian M., Mohammadpour, "Explicit memory based ABC with a clustering strategy for updating and retrieval of memory in dynamic environments", *Applied Intelligence*, Volume 48 , V, 11., pages, 4317-4337, doi:

K. Panigrahi et al., Ed. Springer International Publishing, 2013, pp. 222–235.
[44] .Barlow GJ, Improving memory for optimization and learning in dynamic environments. Doctoral dissertation, Carnegie Mellon University, Pittsburgh, 2011.
[45] J. K. Kordestani, A. Rezvanian, and M. R. Meybodi, “Cdepso: a bi-population hybrid approach for dynamic optimization problems,” Applied Intelligence, vol. 40, no. 4, pp. 682–694, 2014.
[46] U. Halder, D. Maity, P. Dasgupta, and S. Das, “Self-adaptive cluster-based differential evolution with an external archive for dynamic optimization problems,” in Swarm, Evolutionary, and Memetic Computing, B. K. Panigrahi et al., Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 19–26.
[47] H. Wang, S. Yang, W. Ip, and D. Wang, “A memetic particle swarm optimisation algorithm for dynamic multi-modal optimisation problems,” International Journal of Systems Science, vol. 43, no. 7, pp. 1268–1283, 2012.
[48] R. Mukherjee, G. R. Patra, R. Kundu, and S. Das, “Cluster-based differential evolution with crowding archive for niching in dynamic environments,” Inf. Sci., vol. 267, pp. 58 – 82, 2014.
[49] W. Wu, D. Xie, and L. Liu, “Heterogeneous differential evolution with memory enhanced brownian and quantum individuals for dynamic optimization problems,” International Journal of Pattern Recognition and Artificial Intelligence, vol. 32, no. 02, p. 1859003, 2018.
[50] R. Vafashoar and M. R. Meybodi, “A multi-population differential evolution algorithm based on cellular learning automata and evolutionary context information for optimization in dynamic environments,” *Appl. Soft Comput*., p. 106009, 2019.
[51] D. Yazdani, T. T. Nguyen, J. Branke, and J. Wang. A multi-objective time-linkage approach for dynamic optimization problems with previous-solution displacement restriction. *In European Conference on the Applications of Evolutionary Computation. Lecture Notes in Computer Science*, 2018b.
[52] D. Yazdani, B. Nasiri, R. Azizi, A. Sepas-Moghaddam, and M. R. Meybodi. Optimization in dynamic environments utilizing a novel method based on particle swarm optimization. International Journal of Artificial Intelligence, 11:170–192, 2013a.
[53] D. Yazdani. “Particle swarm optimization for dynamically changing environments with particular focus on scalability and switching cost”, *Doctoral thesis, Liverpool John Moores University*, (2018).
memory based artificial immune system for dynamic optimization,” International Journal of Performability Engineering, vol. 14, no. 9, p. 2048, 2018.
[33] W. Luo, J. Sun, C. Bu, and H. Liang, “Species-based particle swarm optimizer enhanced by memory for dynamic optimization,” Appl. Soft Comput., vol. 47, pp. 130 – 140, 2016.
[34] T. Zhu, W. Luo, and L. Yue, “Combining multipopulation evolutionary algorithms with memory for dynamic optimization problems,” in IEEE Congr. Evol. Comput. IEEE, 2014, pp. 2047–2054.
[35] J. K. Kordestani, A. E. Ranginkaman, M. R. Meybodi, and P. Novoa-Hernandez, “A novel framework for improving multi-population algorithms for dynamic optimization problems: A scheduling approach,” Swarm Evol. Comput., vol. 44, pp. 788 – 805, 2019.
[36] Zhu, T., Luo, W., & Yue, L. (2014). Dynamic optimization facilitated by the memory tree. In Soft Computing (Vol. 19, Issue 3, pp. 547–566). Springer Science and Business Media LLC. https://doi.org/10.1007/s00500-014-1273-1.
[37] Yang S (2008) Genetic algorithms with memory- and elitism-based immigrants in dynamic environments. Evol Comput 16(3):385 416.
[38] Yang S, Yao X (2008) Population-based incremental learning with associative memory for dynamic environments. IEEE Trans Evol Comput 12(5):542–561.Simöes A, Costa E (2007b) Variable-size memory evolutionary algorithm to deal with dynamic environments. In: Applications of evolutionary computing, vol. 4448. Springer, Berlin, pp 617–626.
[39] Tinós R, Yang S (2007) A self-organizing random immigrants genetic algorithm for dynamic optimization problems. Gen Progr Evolv Mach 8(3):255–286.
[40] Urbanowicz RJ, Moore JH (2009) Learning classifier systems: a complete introduction, review, and roadmap. J Artif Evol Appl 2009.
[41] S. Sadeghi, H. Parvin and F. Rad, "Particle Swarm Optimization for Dynamic Environments," in *Springer International Publishing, 14th Mexican International Conference on Artificial intelligence, MICAI*, 2015.
[42] D. Yazdani, B. Nasiri, A. Sepas-Moghaddam and M. Meybodi, "a novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization," *Applied Soft Computing,* 2013.
[43] S. Kundu, D. Basu, and S. S. Chaudhuri, “Multipopulation-based differential evolution with speciation-based response to dynamic environments,” in Swarm, Evolutionary, and Memetic Computing, B.

مهدی صادقی مقدم دانشجوی دکترای کامپیوتر مهندسی نرم‌افزار دانشگاه آزاد اسلامی واحد یاسوج و مدرس حق‌التدریس دانشگاه آزاد اسلامی واحد گچساران است.
نشانی رایانامهٔ ایشان عبارت است از:
Mahdi.s.m.1366@gmail.com

صمد نجاتیان مدرک کارشناسی خود را در سال ۱۳۸۲ در رشتهٔ مهندسی برق گرایش الکترونیک از دانشگاه سیستان‌وبلوچستان و مدرک کارشناسی‌ارشد خود را در سال ۱۳۸۶ در رشتهٔ مهندسی برق گرایش مخابرات از دانشگاه مشهد و در سال ۱۳۹۳ مدرک دکتری خود را در رشتهٔ مهندسی برق گرایش مخابرات از دانشگاه یو تی ام کوالالامپور مالزی دریافت کرد. وی عضو هیأت علمی دانشگاه آزاد اسلامی یاسوج، معاون پژوهش و فن‌آوری دانشگاه آزاد اسلامی واحد یاسوج و رئیس باشگاه پژوهش‌گران جوان دانشگاه آزاد اسلامی واحد یاسوج است. حوزه‌های تخصصی ایشان برق، مخابرات و هوش مصنوعی است. وی تاکنون بیش از هفتاد مقالهٔ علمی در نشریات و کنفرانس‌های معتبر داخلی و خارجی به چاپ رسانیده‌است.
نشانی رایانامهٔ ایشان عبارت است از:
samad.nej.2007@gmail.com

حمید پروین مدرک کارشناسی خود را در سال ۱۳۸۵ در رشتهٔ مهندسی کامپیوتر گرایش نرم‌افزار از دانشگاه شهید چمران اهواز و مدرک کارشناسی ارشد و دکتری خود را در سال ۱۳۸۷ و ۱۳۹۲ در رشتهٔ مهندسی کامپیوتر گرایش هوش مصنوعی از دانشگاه علم و صنعت ایران دریافت کرد. وی تاکنون بیش از صد مقالهٔ علمی در نشریات و کنفرانس‌های معتبر داخلی و خارجی به چاپ رسانیده‌است و چندین کتاب چاپ کرده‌اند.
نشانی رایانامهٔ ایشان عبارت است از:
parvin@iust.ac.ir

کرم‌الله باقری‌فرد مدرک کارشناسی خود را در سال ۱۳۸۴ در رشتهٔ مهندسی کامپیوتر گرایش نرم‌افزار از دانشگاه اصفهان و مدرک کارشناسی ارشد و دکتری خود را به ترتیب در سال‌های ۱۳۸۷ و ۱۳۹۵ از دانشگاه نجف آباد و اراک در رشتهٔ مهندسی کامپیوتر گرایش نرم‌افزار دریافت کرد. وی از سال ۱۳۸۵ تاکنون عضو هیأت علمی بخش مهندسی کامپیوتر دانشگاه آزاد اسلامی واحد یاسوج است. حوزه‌های تخصصی ایشان داده‌کاوی، یادگیری ماشین و سامانه‌های پیشنهاددهنده است. وی تاکنون بیش از هفتاد مقالهٔ علمی در نشریات و کنفرانس‌های معتبر داخلی و خارجی به چاپ رسانیده‌است.
نشانی رایانامهٔ ایشان عبارت است از:
k.bagheri@iauyasooj.ac.ir

سید هادی یعقوبیان عضو باشگاه پژوهش‌گران جوان و نخبگان دارای مدرک دکترای کامپیوتر و عضو هیئت علمی دانشگاه آزاد اسلامی واحد یاسوج با بیش از بیست مقالهٔ علمی در نشریات و کنفرانس‌های معتبر داخلی و خارجی است.
نشانی رایانامهٔ ایشان عبارت است از:
Yaghoobian.h@gmail.com