الگوریتم جمعیت ذرات اطلاع‌دهنده‌ی محلی گرانشی برای حل مسائل بهینه‌سازی چندمُدی

محمدباقر دولتشاهی[۱]، دانشجوی دکتری؛ ولی درهمی[۲]، دانشیار؛ حسین نظام‌آبادی‌پور[۳]، استاد

۱- پردیس فنی و مهندسی، گروه مهندسی کامپیوتر - دانشگاه یزد - یزد - ایران - mb.dowlatshahi@stu.yazd.ac.ir
۲- پردیس فنی و مهندسی، گروه مهندسی کامپیوتر - دانشگاه یزد - یزد - ایران - vderhami@yazd.ac.ir
۳- دانشکده فنی و مهندسی، گروه مهندسی برق - دانشگاه شهید باهنر کرمان - کرمان - ایران - nezam@uk.ac.ir

چکیده: الگوریتم جمعیت ذراتِ اطلاع‌دهنده‌ی محلی، یک روش ساده و مؤثر است که اخیراً برای حل مسائل بهینه‌سازی چندمُدی ارائه شده است. این الگوریتم دارای یک ضعف اساسی است: برای محاسبه سرعت یک ذره، "شایستگی" و " فاصله‌ی" ذرات همسایه‌ی آن ذره را در نظر نمی‌گیرد، درصورتیکه در نظر گرفتن این دو پارامتر در محاسبه سرعت می‌تواند به الگوریتم برای ایجاد یک تعادل مناسب بین همگرایی و تنوع راه‌حل‌ها کمک زیادی کند. در این مقاله، یک نسخه جدید از این الگوریتم با نام "الگوریتم جمعیت ذراتِ اطلاع‌دهنده‌ی محلی گرانشی" ارائه شده است، که در آن هر ذره موقعیت خود را با استفاده از قوانین گرانش و حرکت به سمت بهترین موقعیت همسایگان محلی‌اش تنظیم می‌کند. در الگوریتم پیشنهادی، هر چه همسایه‌ی محلی یک ذره دارای کیفیت بیشتری باشد یا دارای فاصله‌ی کمتری با ذره باشد، جرم گرانشی بیشتری به آن همسایه تعلق می گیرد و در نتیجه آن همسایه مجاز به اعمال نیروی گرانشی بیشتری به آن ذره می‌شود. برای بررسی کارایی الگوریتم پیشنهادی، یک ارزیابی تجربی روی چندین تابع محک استاندارد صورت گرفته است. نتایج این آزمایشات نشان می‌دهد که الگوریتم پیشنهادی می‌تواند نتایج بهتری نسبت به الگوریتم جمعیت ذراتِ اطلاع‌دهنده‌ی محلی و سایر الگوریتم‌های بهینه‌ساز چندمُدی به دست آورد.

واژه‌های کلیدی: بهینه‌سازی جمعیت ذرات، الگوریتم جستجوی گرانشی، قاعده به‌روزرسانی سرعت، بهینه‌سازی چندمُدی.

Gravitational Locally Informed Particle Swarm Algorithm for solving Multimodal Optimization Problems

M. B. Dowlatshahi[1], PhD student; V. Derhami[2], Associated Professor; H. Nezamabadi-pour[3], Professor

1- Faculty of Engineering, Computer Engineering Department, Yazd University, Yazd, Iran, Email: mb.dowlatshahi@stu.yazd.ac.ir
2- Faculty of Engineering, Computer Engineering Department, Yazd University, Yazd, Iran, Email: vderhami@yazd.ac.ir
3 - Department of Electrical Engineering, Shahid Bahonar University of Kerman, Kerman, Iran, Email: nezam@uk.ac.ir

Abstract: Locally Informed Particle Swarm (LIPS) is a simple and effective method for solving multimodal optimization problems. Despite the good performance of LIPS's velocity updating rule, the quality (fitness) of this local neighbors is not considered in calculating the velocity. Considering the quality of neighbors to update the particle velocity can reinforce the search power of LIPS. In this paper, a new version of LIPS with Gravitational velocity updating rule (GLIPS) is proposed. In GLIPS each particle successively adjusts its position towards the best positions of its local neighbors using laws of gravity and motion. In proposed GLIPS, local neighbors with a higher quality get a greater gravitational mass and therefore are allowed to apply the higher gravity force to other particles to attract them. In this case, the particles near good solutions try to attract the other particles which are exploring the search space. We perform a detailed empirical evaluation on the several commonly used multimodal benchmark functions. Our results demonstrate that the new velocity updating rule for LIPS can obtain better results for multimodal function optimization.

Keywords: Particle Swarm Optimization, Gravitational Search Algorithm, Velocity updating rule, Multimodal optimization.

تاریخ ارسال مقاله: ۱۳۹۵/۱۱/۰۷
تاریخ اصلاح مقاله: ۱۳۹۶/۰۱/۱۰
تاریخ پذیرش مقاله: ۱۳۹۶/۰۴/۲۴
نام نویسنده مسئول: ولی درهمی
نشانی نویسنده مسئول: ایران – یزد، صفاییه،دانشگاه یزد، پردیس فنی و مهندسی، گروه مهندسی کامپیوتر.

۱- مقدمه

در ریاضیات و علم کامپیوتر، یک مسأله بهینه‌سازی، عبارت است از مسأله‌ی یافتن بهترین راه‌حل از میان همه راه‌حل‌های ممکن [۱]. ساده‌ترین نوع مسائل بهینه‌سازی، مسائل بهینه‌سازی تک‌هدفه-تک‌مُدی هستند که هدف از حل این نوع مسائل، پیداکردن یک بردار در فضای تصمیم است که متناظر با نقطه‌ی بهینه‌ی سراسری تابع هدف است [۲]. دسته‌ی دیگری از مسائل بهینه‌سازی وجود دارند که با نام مسائل بهینه‌سازی تک‌هدفه-چندمُدی (یا به اختصار مسائل بهینه‌سازی چندمُدی[۱]) شناخته می‌شوند. برخلاف مسائل بهینه‌سازی تک‌هدفه-تک‌مُدی، هدف از حل مسائل بهینه‌سازی چندمُدی پیدا کردن فقط یک بردار در فضای تصمیم که متناظر با نقطه‌ی بهینه‌ی سراسری تابع هدف باشد نیست، بلکه هدف پیداکردن تمام بردارهایی در فضای تصمیم است که متناظر با تمام نقاط بهینه‌ی سراسری و بهینه‌ی محلی تابع هدف هستند [۳].

الگوریتم‌های فرا ابتکاری، دسته‌ای از روش‌های جستجوی هوشمند در فضای جستجوی مسأله هستند که می‌توانند جواب‌های نزدیک به بهینه مسائل بهینه‌سازی پیچیده را به‌دست آورند. این الگوریتم‌ها هیچ تضمینی برای پیداکردن مقادیر بهینه ارائه نمی‌دهند. با این وجود، با توجه به قابلیت‌هایی که این الگوریتم‌ها در عمل از خود نشان داده‌اند، امروزه از این الگوریتم‌ها به‌وفور استفاده می‌شود [۴].

به‌طور کلی الگوریتم‌های فرا ابتکاری را می‌توان به دو گروه تقسیم کرد: مبتنی بر جمعیت[۲] در مقابل مبتنی بر تک‌راه‌حل[۳]. الگوریتم‌های فرا ابتکاری تک‌راه‌حله در خلال فرآیند جستجو فقط یک راه‌حل را دستکاری می‌کنند، درصورتیکه در الگوریتم‌های فرا ابتکاری مبتنی بر جمعیت در خلال فرآیند جستجو یک جمعیت از راه‌حل‌ها دستکاری می‌شود. این دو دسته از الگوریتم‌های فرا ابتکاری مکمل یکدیگر هستند: الگوریتم‌های فرا ابتکاری تک‌راه‌حله معمولاً بر قابلیت "بهره‌گیری" تاکید دارند، درصورتیکه الگوریتم‌های فرا ابتکاری مبتنی بر جمعیت معمولاً بر قابلیت "کاوش" تاکید دارند [۴]. مشهورترین الگوریتم‌های فرا ابتکاری مبتنی بر جمعیت عبارتند از: الگوریتم‌های تکاملی[۴] [۵] و الگوریتم‌های هوش جمعی[۵] [۶].

الگوریتم‌های هوش جمعی از رفتارهای جمعی موجود در طبیعت الهام گرفته‌اند. مهم‌ترین مؤلفه در این الگوریتم‌ها، عناصری به نام ذره[۶] هستند که این ذره‌ها توسط یک رسانه ارتباطی، به‌طور غیرمستقیم با هم همکاری می‌کنند و بر همین اساس، به طور مداوم مشغول حرکت‌کردن در فضای جستجوی مسأله هستند [۴]. الگوریتم بهینه‌ساز جمعیت ذرات[۷] [۷]، الگوریتم جستجوی گرانشی[۸] [۸ و ۹]، الگوریتم بهینه‌سازی گروه ذرات تعاونی[۹] [۱۰] از جمله الگوریتم‌های هوش جمعی ای هستند که تا امروز در کاربردهای مختلفی استفاده شده‌اند.

الگوریتم جمعیت ذراتِ اطلاع‌دهنده‌ی محلی[۱۰] [۱۱]، یک الگوریتم هوش جمعی ساده و مؤثر است که اخیراً برای حل مسائل بهینه‌سازی چندمُدی ارائه شده است. این الگوریتم دارای یک ضعف اساسی است: برای محاسبه سرعت یک ذره، "شایستگی" و " فاصله‌ی" ذرات همسایه‌ی آن ذره را در نظر نمی‌گیرد، درصورتیکه در نظرگرفتن این دو پارامتر در محاسبه سرعتِ یک ذره، می‌تواند به الگوریتم برای ایجاد یک تعادل مناسب بین همگرایی و تنوع راه‌حل‌ها کمک زیادی کند. در این مقاله، یک نسخه جدید از این الگوریتم با نام "الگوریتم جمعیت ذراتِ اطلاع‌دهنده‌ی محلی گرانشی[۱۱]" ارائه شده است، که در آن هر ذره موقعیت خود را با استفاده از قوانین گرانش و حرکت به سمت بهترین موقعیت همسایگان محلی‌اش تنظیم می‌کند. در الگوریتم پیشنهادی، هر چه همسایه‌ی محلی یک ذره دارای کیفیت بیشتری باشد یا دارای فاصله ی کمتری با ذره باشد، جرم گرانشی بیشتری به آن همسایه تعلق می گیرد و در نتیجه آن همسایه مجاز به اعمال نیروی گرانشی بیشتری به آن ذره می‌شود. نتایج آزمایشات نشان می‌دهد که الگوریتم پیشنهادی می‌تواند نتایج بهتری نسبت به الگوریتم جمعیت ذراتِ اطلاع‌دهنده‌ی محلی و سایر الگوریتم‌های بهینه‌ساز چندمُدی به‌دست آورد.

ادامه مقاله به این صورت سازماندهی شده است: در بخش ۲ مروری بر الگوریتم بهینه‌ساز جمعیت ذرات، الگوریتم جمعیت ذراتِ اطلاع‌دهنده ی محلی، الگوریتم جستجوی گرانشی، و مشهورترین الگوریتم‌های هوش جمعی برای حل مسائل بهینه‌سازی چندمُدی خواهیم داشت. در بخش ۳ روش پیشنهادی شرح داده خواهد شد. در بخش ۴ نتایج حاصل از پیاده‌سازی الگوریتم پیشنهادی در حل برای حل مسائل بهینه‌سازی چندمُدی و همچنین مقایسه این نتایج با سایر الگوریتم‌های حل مسائل بهینه‌سازی چندمُدی آورده شده است. در بخش ۵ نیز نتیجه‌گیری آورده شده است.

۲- کارهای مرتبط

۲-۱- الگوریتم بهینه‌ساز جمعیت ذرات

کندی و ابرهارت در سال ۱۹۹۵ [۷] برای اولین بار الگوریتم بهینه‌ساز جمعیت ذرات را برای حل مسائل بهینه‌سازی تک‌هدفه پیوسته ارائه دادند. هر ذره از این الگوریتم، دارای سه عنصر زیر است: یک بردار موقعیت در فضای جستجو که نشان‌دهنده موقعیت فعلی ذره است، یک بردار سرعت که جهت تغییرات ذره را نشان می‌دهد، و یک بردار موقعیت در فضای جستجو که نشان‌دهنده بهترین موقعیتی است که ذره تابه‌حال در آن قرار گرفته است. همچنین در این الگوریتم یک بردار موقعیت در فضای جستجو وجود دارد که نشان‌دهنده بهترین موقعیتی است که تمام ذرات توانسته‌اند تابه‌حال پیدا کنند.

فرض کنید هر ذره را به‌صورت یک سه تایی $Particle_i=\left(\vec{X}_i(t),\vec{V}_i(t),\overrightarrow{Pbest_i}(t)\right)$ نشان دهیم که در آن $\vec{X}_i(t)$، $\vec{V}_i(t)$، و $\overrightarrow{Pbest_i}(t)$ به‌ترتیب موقعیت، سرعت، و بهترین موقعیت پیدا شده توسط عامل i-اُم هستند و داریم:

$$\vec{X}_i(t)=\left(x_i^1(t),x_i^2(t),\ldots,x_i^d(t),\ldots,x_i^n(t)\right), \quad (۱)$$

$$\vec{V}_i(t)=\left(v_i^1(t),v_i^2(t),\ldots,v_i^d(t),\ldots,v_i^n(t)\right), \quad (۲)$$

$$\overrightarrow{Pbest_i}(t)=\left(pbest_i^1(t), pbest_i^2(t),\ldots, pbest_i^n(t)\right), \quad (۳)$$

که در آن n تعداد بُعد (متغیر) مسئله و N تعداد اعضای جمعیت است.

در این حالت، سرعت بعدی عامل i–اُم به‌صورت زیر محاسبه می‌شود [۹]:

$$\begin{aligned} v_i^d(t+1) &= w(t)\times v_i^d(t) \\ &+ c_1\times r_1\times\left(pbest_i^d(t)-x_i^d(t)\right) \\ &+ c_2\times r_2\times\left(gbest^d(t)-x_i^d(t)\right), \end{aligned} \quad (۴)$$

که در آن $w(t)$ ضریب اینرسی است، $c_1>0$ و $c_2>0$ ضرایب شتاب‌دهی هستند، و r_1 و r_2 دو عدد تصادفی بین صفر و یک هستند. برای به‌روزرسانی موقعیت عامل i–اُم از رابطه زیر استفاده می‌شود:

$$x_i^d(t+1)=x_i^d(t)+v_i^d(t+1). \quad (۵)$$

شبه‌کد الگوریتم بهینه‌ساز جمعیت ذرات در الگوریتم (۱) آمده است.

الگوریتم (۱): شبه‌کد الگوریتم بهینه‌ساز جمعیت ذرات

۱. مقداردهی به پارامترهای الگوریتم.
۲. تولید جمعیت اولیه از ذرات به‌صورت تصادفی.
۳. تازمانی‌که شرایط پایانی برآورده نشده است، مراحل (۴) تا (۹) را انجام بده:
۴. ارزیابی جمعیت فعلی.
۵. تعیین بهترین موقعیت هر ذره تاکنون.
۶. تعیین بهترین موقعیت پیدا شده توسط کل ذرات تاکنون.
۷. محاسبه سرعت هر ذره.
۸. به‌روزرسانی موقعیت هر ذره.
۹. به‌روزرسانی پارامترهای الگوریتم.
۱۰. برگرداندن بهترین راه‌حل پیداشده توسط الگوریتم به‌عنوان خروجی.

۲-۲- الگوریتم جمعیت ذراتِ اطلاع‌دهنده‌ی محلی

برخلاف الگوریتم بهینه‌ساز جمعیت ذرات، در الگوریتم جمعیت ذراتِ اطلاع‌دهنده‌ی محلی که در سال ۲۰۱۳ ارائه شده است [۱۱]، فرایند به‌روزرسانی سرعت ذره‌ی i–اُم با استفاده از رابطه زیر انجام می‌شود (و رابطه به‌روزرسانی موقعیت ذره‌ی i–اُم همانند رابطه به‌روزرسانی موقعیت در الگوریتم بهینه‌ساز جمعیت ذرات است):

$$v_i^d(t+1)=w(t)\times\left(v_i^d(t)+\varphi\left(P_i^d(t)-x_i^d(t)\right)\right), \quad (۶)$$

که در آن داریم:

$$P_i^d(t)=\frac{\sum_{j=1}^{nsize}\left(\varphi_j\times nbest_j\right)/nsize}{\varphi}. \quad (۷)$$

که در آن φ_j یک عدد تصادفی در بازه‌ی $[0, 4.1/nsize]$ است، φ برابر با مجموع φ_j ها است، $nbest_j$ به معنای j–اُمین همسایه ذره i–اُم است، و $nsize$ اندازه همسایگی هر ذره است (یعنی تعداد ذراتی که به عنوان همسایه یک ذره شناخته می‌شوند و بر فرایند به‌روزرسانی موقعیت فعلی ذره تاثیرگذار هستند).

در الگوریتم جمعیت ذرات اطلاع‌دهنده‌ی محلی، انتخاب یک مقدار کوچک برای پارامتر اندازه همسایگی باعث بهبود تنوع و انتخاب یک مقدار بزرگ برای این پارامتر باعث بهبود همگرایی می‌شود. در نسخه اصلی این الگوریتم، اندازه‌ی همسایگی پویا در نظر گرفته شده است، به‌طوری‌که در طول فرایند جستجو به‌طور خطی از مقدار ۲ به سمت مقدار ۵ رشد می‌کند.

شبه‌کد الگوریتم جمعیت ذرات اطلاع‌دهنده‌ی محلی در الگوریتم (۲) آمده است.

الگوریتم (۲): شبه‌کد الگوریتم جمعیت ذرات اطلاع‌دهنده‌ی محلی

۱. مقداردهی به پارامترهای الگوریتم.
۲. تولید یک جمعیت اولیه از ذرات (راه‌حل‌ها) به‌صورت تصادفی.
۳. تازمانی‌که شرایط پایانی برآورده نشده است، مراحل (۴) تا (۹) را انجام بده:
۴. به‌روزرسانی پارامتر $nsize$ بین ۲ تا ۵.
۵. ارزیابی جمعیت فعلی.
۶. تعیین بهترین موقعیت هر ذره تاکنون.
۷. تعیین همسایه‌های هر ذره.
۸. محاسبه سرعت هر ذره.
۹. به‌روزرسانی موقعیت هر ذره.
۱۰. برگرداندن جمعیت نهایی پیداشده توسط الگوریتم به‌عنوان خروجی.

دقت شود که پیچیدگی محاسباتی هر تکرار از حلقه‌ی الگوریتم (۲) از درجه‌ی $O(nsize\times n\times N^2)$ است، چراکه در هر تکرار از حلقه باید برای هر عامل از جمعیت، $nsize$ نزدیک‌ترین همسایه‌ی آن عامل را در فضای n–بُعدی پیدا کنیم. لازم به ذکر است که سایر دستورهای حلقه ی تکرار دارای درجه‌ی کوچکتری از $O(nsize\times n\times N^2)$ هستند و بنابراین از نظر تئوری پیچیدگی محاسباتی در نظر گرفته نمی‌شوند.

۲-۳- الگوریتم جستجوی گرانشی

الگوریتم جستجوی گرانشی [۸] یک الگوریتم فرا ابتکاری که برای حل مسائل بهینه‌سازی پیوسته معرفی شده است [۸]. ایده اصلی این الگوریتم، شبیه‌سازی قانون گرانش نیوتون و قوانین حرکت بر روی یک جمعیت از جرم‌ها در یک فضای پیوسته n–بُعدی می‌باشد.

اولین گام از الگوریتم جستجوی گرانشی، تشکیل یک سیستم مصنوعی n–بُعدی با N عامل جستجوگر که به‌صورت تصادفی مقداردهی شده‌اند، است. بر اساس [۱۰]، جرم گرانشی عامل i در زمان t (که با نماد $M_i(t)$ نشان داده می‌شود)، پس از محاسبه شایستگی جمعیت فعلی جرم‌ها با استفاده از رابطه‌های (۸) و (۹) محاسبه می‌شود.

$$q_i(t)=\frac{fit_i(t)-worst(t)}{best(t)-worst(t)}, \quad (۸)$$

$$M_i(t)=\frac{q_i(t)}{\sum_{j=1}^{N}q_j(t)}, \quad (۹)$$

که در آن $M_i(t)$ و $fit_i(t)$ به‌ترتیب جرم گرانشی و میزان شایستگی عامل i را در زمان t نشان می‌دهند. همچنین $best(t)$ و $worst(t)$ به‌ترتیب نماینده بهترین و بدترین جرم‌های جمعیت در زمان t هستند؛ مقدار این دو پارامتر برای مسائل کمینه‌سازی، به‌ترتیب با استفاده از رابطه‌های (۱۰) و (۱۱) محاسبه می‌شود (برای مسائل بیشینه‌سازی جای این دو عوض می‌شود).

$$best(t)=\min_{j\in\{1,\ldots,N\}} fit_j(t), \qquad (۱۰)$$

$$worst(t)=\max_{j\in\{1,\ldots,N\}} fit_j(t). \qquad (۱۱)$$

پس از محاسبه جرم گرانشی برای هر عامل، حال می‌توان شتاب، سرعت و موقعیت جدید هر عامل را در فضای جستجو محاسبه کرد. برای محاسبه شتاب عامل i در زمان t، برآیند نیروهای واردشده از جانب K عامل برتر جمعیت بر روی عامل i بر اساس قانون گرانش محاسبه می شود. سپس برای محاسبه سرعت یا میزان جابجایی عامل i در زمان $t+1$، کسری از سرعت این عامل در زمان t با شتاب این عامل در زمان t جمع می‌شود (رابطه (۱۴)). در نهایت موقعیت جدید عامل i، می‌تواند بر اساس رابطه (۱۵) محاسبه شود.

$$F_i^d(t)=\sum_{j\in Kbest,\, j\neq i} rand_j G(t)\frac{M_j(t)M_i(t)}{R_{ij}(t)+\varepsilon}(x_j^d(t)-x_i^d(t)), \qquad (۱۲)$$

$$a_i^d(t)=\frac{F_i^d(t)}{M_i(t)}=\sum_{j\in Kbest,\, j\neq i} rand_j G(t)\frac{M_j(t)}{R_{ij}(t)+\varepsilon}(x_j^d(t)-x_i^d(t)), \qquad (۱۳)$$

$$v_i^d(t+1)=rand_i\times v_i^d(t)+a_i^d(t), \qquad (۱۴)$$

$$x_i^d(t+1)=x_i^d(t)+v_i^d(t+1). \qquad (۱۵)$$

شبه‌کد الگوریتم جستجوی گرانشی در الگوریتم (۳) آمده است.

الگوریتم (۳): شبه‌کد الگوریتم جستجوی گرانشی

۱. مقداردهی اولیه به پارامترها.
۲. مقداردهی اولیه به عامل‌ها به‌صورت تصادفی.
۳. تازمانی‌که شرایط پایانی برآورده نشده است، مراحل (۴) تا (۹) انجام بده:
۴. ارزیابی شایستگی عامل‌ها.
۵. محاسبه M برای هر عامل.
۶. به‌روزرسانی پارامترهای G، K، و $Kbest$.
۷. محاسبه شتاب وارده به هر عامل.
۸. محاسبه سرعت هر عامل.
۹. به‌روزرسانی موقعیت هر عامل.
۱۰. برگرداندن بهترین راه‌حل پیداشده.

۲-۴- هوش جمعی و حل مسائل بهینه‌سازی چندمُدی

تابه‌حال محققان زیادی بر روی حل مسائل بهینه‌سازی چندمُدی با استفاده از الگوریتم بهینه‌ساز جمعیت ذرات کار کرده‌اند. در این بخش به بررسی و تشریح برخی از مهمترین کارهای انجام‌شده در این زمینه خواهیم پرداخت.

بریتس و همکاران [۱۲] الگوریتمی به نام NichePSO را برای حل مسائل بهینه‌سازی چندمُدی معرفی کردند که در آن چندین زیرجمعیت از جمعیت اولیه با توجه به شایستگی ذرات ساخته می‌شود. سپس، اوزکان و همکاران [۱۳] عملکرد الگوریتم NichePSO را با استفاده از مکانیزم تپه‌نوردی بهبود دادند. علاوه بر کارایی نه چندان مناسب، ضعف دیگر این الگوریتم‌ها این است که نیازمند مجموعه‌ای از پارامترهای مرتبط با هر نیچ[۱۲] هستند.

پاسارو و اِستاریتا [۱۴] یک الگوریتم بهینه‌ساز جمعیت ذرات به نام الگوریتم k-means-PSO را برای حل مسائل بهینه‌سازی چندمُدی ارائه دادند که در آن با استفاده از الگوریتم خوشه‌بندی k-means به خوشه‌بندی جمعیت به‌منظور شناسایی نیچ‌ها می‌پردازد. از طریق حل مجموعه‌ای از توابع آزمون استاندارد، الگوریتم k-means-PSO عملکرد بهتری نسبت به برخی از الگوریتم‌های موجود تا آن زمان را نشان می داد. اگر چه این الگوریتم جواب‌های خوبی در مقایسه با کارهای زمان خود به‌دست آورد، اما نیازمند تنظیم پارامتر تعداد خوشه‌ها و همچنین تعداد مراحل تکرار الگوریتم خوشه‌بندی k-means می‌باشد، که این موضوع استفاده از آن را برای حل مسائل دنیای واقعی با مشکل مواجه می‌سازد.

لی [۱۵] یک الگوریتم بهینه‌ساز جمعیت ذرات به نام الگوریتم RPSO بر اساس توپولوژی حلقه‌ای را برای حل مسائل بهینه‌سازی چندمُدی ارائه داد که مزیت اصلی آن این است که به هیچ پارامتر اضافه‌ای (مثل تعداد نیچ‌ها یا تعداد خوشه‌ها) وابسته نیست. لی نشان داد که الگوریتم RPSO قادر است به‌خوبی مسائل چندمُدی را حل کند. یک نقطه ضعف بزرگ الگوریتم RPSO این است که در توپولوژی حلقه‌ای امکان ایجاد ارتباط بین ذرات موجود در نیچ‌های مختلف نیز وجود دارد که این موضوع باعث کُندشدن قدرت همگرایی الگوریتم می‌شود. برای مثال، اگر دو ذره در دو نیچ مختلف قرار گرفته باشند، حرکت آنها به‌سمت هم باعث ایجاد نوسان[۱۳] در حرکت ذره می‌شود که نوسان بین دو نیچ تأثیر نامطلوبی در کاوش و بهره‌گیری الگوریتم دارد. با این حال، لی نشان داده است که الگوریتم RPSO در برخی مسائل قادر به تشکیل نیچ‌های پایدار است.

کیو و همکاران [۱۱] یک الگوریتم بهینه‌ساز جمعیت ذرات به نام الگوریتم جمعیت ذرات اطلاع‌دهنده‌ی محلی یا به اختصار LIPS را برای حل مسائل بهینه‌سازی چندمُدی ارائه دادند که در آن موقعیت جدید یک ذره با استفاده از موقعیت تعدادی از نزدیکترین همسایه‌هایش محاسبه می‌شود. در این الگوریتم از فاصله‌ی اقلیدسی برای تعیین نزدیکترین همسایه‌های هر ذره استفاده شده است. کیو و همکاران [۱۱] به مقایسه الگوریتم LIPS با چندین الگوریتم مشهور هوش جمعی برای حل مسائل بهینه‌سازی چندمُدی پرداخته‌اند که نتایج تجربی نشان می دهد که الگوریتم LIPS قادر به ارائه عملکرد آماری بهتری خواهد بود. با وجود اینکه الگوریتم LIPS یکی از بهترین الگوریتم‌هایی است که تا امروز

برای حل مسائل بهینه‌سازی چندمُدی ارائه شده است، اما این الگوریتم دارای یک ضعف اساسی است: برای محاسبه سرعت ذره‌ی i، "شایستگی ذرات همسایه ذره‌ی i" و "فاصله‌ی ذرات همسایه ذره‌ی i" را در نظر نمی گیرد، درصورتیکه برای ایجاد یک تعادل مناسب بین همگرایی و تنوع، در نظرگرفتن این دو پارامتر در محاسبه سرعت می‌تواند کمک زیادی کند.

کیو و همکاران [۳] تعدادی تابع محک جدید برای مسائل بهینه‌سازی چندمُدی ارائه دادند که نتایج پیاده‌سازی تعدادی از مشهورترین الگوریتم‌های فرا ابتکاری چندمُدی بر روی این توابع نشان می‌دهد که در حال حاضر الگوریتم‌های فرا ابتکاری فعلی در حل این مسائل کارایی چندان مناسبی ندارند. بنابراین، ابداع الگوریتم‌های فرا ابتکاری جدید به‌منظور حل مؤثر و کارامد مسائل بهینه‌سازی چندمُدی یک نیاز ضروری در مبحث الگوریتم‌های فرا ابتکاری به حساب می‌آید.

به‌طور کلی، برای حل مسائل بهینه‌سازی چندمُدی با استفاده از الگوریتم بهینه‌ساز جمعیت ذرات (و معمولاً تمام الگوریتم‌های هوش جمعی) بایستی به دو سؤال اساسی زیر پاسخ داده شود:

- چگونه ذرات رهبر انتخاب شوند که باعث ایجاد تنوع در جمعیت به‌منظور جلوگیری از همگرایی به یک راه‌حل واحد شوند؟
- چگونه با استفاده از ذرات رهبر و موقعیت فعلی یک ذره، سرعت جدید آن ذره محاسبه شود؟

۳- الگوریتم پیشنهادی: الگوریتم جمعیت ذرات اطلاع‌دهنده‌ی محلی گرانشی

در این بخش به تشریح الگوریتم پیشنهادی یعنی الگوریتم جمعیت ذراتِ اطلاع‌دهنده‌ی محلی گرانشی یا به اختصار GLIPS پرداخته می‌شود. مشابه با الگوریتم LIPS، الگوریتم GLIPS از نزدیکترین همسایه‌های خود (از نظر فاصله اقلیدسی) برای هدایت ذرات جستجوگر استفاده می کند. تفاوت اصلی بین دو الگوریتم LIPS و GLIPS این است که در الگوریتم پیشنهادی هر ذره موقعیت خود را به‌سمت موقعیت همسایگان محلی‌اش با استفاده از قوانین گرانش و حرکت به‌روزرسانی می‌کند. بنابراین، در الگوریتم GLIPS هرچه همسایه‌های محلی یک ذره کیفیت بالاتری داشته باشند یا فاصله‌ی کمتری با آن ذره داشته باشند، یک جرم گرانشی بیشتر به آن همسایه‌ها اختصاص داده می‌شود و در نتیجه آن همسایه‌ها مجاز به اعمال نیروی گرانش بیشتری به آن ذره هستند. در این حالت، ذراتی از الگوریتم که در نزدیکی راه‌حل‌های خوب قرار دارند به جذب ذرات همسایه خود در فضای جستجو می‌پردازند.

در الگوریتم GLIPS، به‌روزرسانی سرعت با استفاده از رابطه زیر انجام می‌شود (و رابطه به‌روزرسانی موقعیت بدون تغییر باقی می‌ماند):

$$V_i^d(t)=\sum_{j\in K_i^{local}} rand_j G(t)\frac{M_j^{local}(t)}{R_{ij}(t)+\varepsilon}\left(x_j^d(t)-x_i^d(t)\right), \qquad (۱۶)$$

که در آن K_i^{local} مجموعه‌ی همسایه‌های ذره i–اُم است. بر اساس رابطه ی بالا، ذره i–اُم قادر است از همسایه‌های محلی‌اش بر اساس قانون گرانش و حرکت تأثیر بپذیرد. در رابطه (۱۶)، $M_i^{local}(t)$ به معنای جرم گرانشی ذره i–اُم است که برای محاسبه آن داریم:

$$q_i^{local}(t)=\frac{fit_i(t)-worst^{local}(t)}{best^{local}(t)-worst^{local}(t)}\ , \qquad (۱۷)$$

$$M_i^{local}(t)\ =\ \frac{q_i^{local}(t)}{\sum_{j\in K_i^{local}} q_j^{local}(t)}\ , \qquad (۱۸)$$

که در آن $worst^{local}(t)$ و $best^{local}(t)$ برای مسائل کمینه‌سازی به‌صورت زیر محاسبه می‌شوند:

$$best^{local}(t)=\min_{j\in K_i^{local}} fit_j(t), \qquad (۱۹)$$

$$worst^{local}(t)=\max_{j\in K_i^{local}} fit_j(t)\ . \qquad (۲۰)$$

توجه داشته باشید که در طول فرایند بهینه‌سازی چندمُدی دو نیاز اصلی عبارتند از: تنوع (حفظ مجموعه‌ای متنوع از راه‌حل‌ها) و همگرایی (هدایت ذرات به‌سمت راه‌حل‌های بهینه واقعی). در الگوریتم GLIPS، دو پارامتر K و G اجزاء اصلی الگوریتم برای ایجاد تعادل بین تنوع و همگرایی هستند. بنابراین، انتخاب مقدار مناسب برای این پارامترها یک مسأله اساسی است که باید مورد بحث قرار گیرد. در ادامه، به تشریح نحوه به‌روزرسانی مقدار این پارامترها در طول زمان خواهیم پرداخت.

در الگوریتم GLIPS، انتخاب مقدار کوچک برای پارامتر K موجب می‌شود که یک ذره موقعیتش را بر اساس موقعیت تعداد کمی از همسایگان خود به‌روزرسانی کند و در نتیجه آن دسته از ذرات که در فضای جستجو به‌صورت محلی بهترین هستند، در نهایت تشکیل تعداد بسیار زیادی نیچ می‌دهند. لازم به ذکر است که در این شرایط، برخی از این نیچ‌ها ممکن است نیچ واقعی نباشند. در این حالت، می‌توان گفت که الگوریتم توانسته است تنوع را به‌دست آورد اما در پاره‌ای از نقاط همگرایی را از دست داده است. به‌طور عکس، انتخاب مقدار بزرگ برای پارامتر K اجازه می‌دهد تا یک ذره موقعیتش را براساس موقعیت تعداد زیادی از همسایگان خود به‌روزرسانی کند و در نتیجه با خطر از دست‌دادن برخی از نیچ‌های واقعی مواجه شویم. در این حالت، می‌توان گفت که الگوریتم قادر خواهد بود همگرایی به برخی از نیچ‌ها را به‌دست آورد اما در عوض تنوع راه‌حل‌ها را از دست داده است. به نظر ما، یک استراتژی خوب برای به‌دست‌آوردن یک مجموعه خوب از نیچ‌های واقعی با الگوریتم GLIPS این است که از استراتژی سه مرحله‌ای زیر استفاده کنیم: (۱) مقداردهی اولیه K با یک مقدار کوچک در شروع کار الگوریتم GLIPS به‌منظور ساختن یک مجموعه متنوع از نیچ‌های واقعی و غیرواقعی، (۲) افزایش مقدار K به‌منظور حرکت‌کردن نیچ‌های غیرواقعی ساخته‌شده به‌سمت نزدیک‌ترین نیچ‌های واقعی، و (۳) کاهش مقدار K برای ساخت یک مجموعه متنوع از نیچ‌ها در نزدیکی نیچ‌های واقعی. با توجه به این توضیحات، مقدار پارامتر K را می‌توان به‌صورت پویا در طول اجرای الگوریتم به‌صورت زیر به‌روزرسانی کرد:

$$K=\begin{cases}3 & , \quad if\ \%0\le FE\le\%25\\ \frac{N}{100}+2, & if\ \%25<FE\le\%50\\ 3 & , \quad if\ \%50<FE\le\%75\\ 2 & , \quad if\ \%75<FE\le\%100\end{cases}, \qquad (۲۱)$$

که در آن FE به معنای درصدِ ارزیابی شایستگی سپری شده است.

همانند الگوریتم جستجوی گرانشی، در الگوریتم GLIPS انتخاب مقدار بزرگ برای پارامتر G تحرک ذره را افزایش می‌دهد و ازاین‌رو تنوع افزایش می‌یابد. همچنین، انتخاب مقدار کوچک برای پارامتر G تحرک هر ذره را کاهش می‌دهد و ازاین‌رو همگرایی افزایش خواهد یافت. با توجه به این موارد، مقدار پارامتر G به‌صورت پویا در طول اجرای الگوریتم به‌روز می‌شود:

$$G=\begin{cases}100\times\sqrt{n}, & if\ \%0\le FE\le\%25\\ 15\times\sqrt{n} & , \quad if\ \%25<FE\le\%50\\ 2\times\sqrt{n} & , \quad if\ \%50<FE\le\%75\\ 0.5\times\sqrt{n} & , \quad if\ \%75<FE\le\%100\end{cases}, \qquad (۲۲)$$

که در آن n به معنای تعداد ابعاد فضای جستجو است.

شبه‌کد الگوریتم جمعیت ذرات اطلاع‌دهنده‌ی محلی گرانشی در الگوریتم (۴) آمده است.

الگوریتم (۴): شبه‌کد الگوریتم جمعیت ذرات اطلاع‌دهنده‌ی محلی گرانشی

۱. مقداردهی به پارامترهای الگوریتم.
۲. تولید یک جمعیت اولیه از ذرات (راه‌حل‌ها) به‌صورت تصادفی.
۳. تازمانی‌که شرایط پایانی برآورده نشده است، مراحل (۴) تا (۹) را انجام بده:
۴. ارزیابی جمعیت فعلی.
۵. تعیین بهترین موقعیت هر ذره تاکنون.
۶. محاسبه جرم گرانشی هر عامل با رابطه (۱۸).
۷. به‌روزرسانی پارامترهای G، K و K^{local}
۸. محاسبه سرعت هر ذره.
۹. به‌روزرسانی موقعیت هر ذره.
۱۰. برگرداندن جمعیت نهایی پیداشده توسط الگوریتم به‌عنوان خروجی.

توجه داشته باشید که نحوه‌ی ترکیب‌کردن دو الگوریتم LIPS و GSA به‌منظور پیشنهاد الگوریتم GLIPS، دارای سه ویژگی زیر است [۴]:

- ترکیب سطح پایین: چراکه قابلیت‌های دو الگوریتم جمعیت ذراتِ اطلاع‌دهنده‌ی محلی و الگوریتم جستجوی گرانشی را با هم ترکیب می‌کند.
- ترکیب مبتنی بر کار تیمی: چراکه دو الگوریتم مذکور با هم ترکیب شده و در کنار هم و به‌صورت موازی اجرا می‌شوند و به حل مسئله می‌پردازند.
- ترکیب ناهمگن: چراکه دو الگوریتم مختلف با هم ترکیب شده‌اند و ترکیب دو الگوریتم مختلف منجر به تولید نتایج نهایی می‌شود.

پیچیدگی محاسباتی هر تکرار از حلقه‌ی الگوریتم (۴) از درجه‌ی $O(K\times n\times N^2)$ است، چراکه در هر تکرار از حلقه باید برای هر عامل از جمعیت، K نزدیک‌ترین همسایه‌ی آن عامل را در فضای n-بُعدی پیدا کنیم. چون سایر دستورهای حلقه‌ی تکرار دارای درجه‌ی کوچکتری از $O(K\times n\times N^2)$ هستند، بنابراین از نظر تئوری پیچیدگی محاسباتی در نظر گرفته نمی‌شوند. با این اوصاف، دو الگوریتم LIPS و GLIPS دارای مرتبه ی رشد پیچیدگی یکسانی هستند.

۴- نتایج شبیه‌سازی و مقایسات

در این بخش، به بررسی نتایج شبیه‌سازی و ارزیابی اثربخشی الگوریتم GLIPS پیشنهادی برای حل توابع استاندارد بهینه‌سازی چندمُدی خواهیم پرداخت. در ادامه، ابتدا به معرفی و بررسی ویژگی‌های توابع محک انتخاب‌شده خواهیم پرداخت. سپس، تنظیمات شبیه‌سازی الگوریتم GLIPS شرح داده خواهد شد. در نهایت، نتایج به‌دست آمده از اجرای الگوریتم GLIPS پیشنهادی و سایر الگوریتم‌های مشهور بهینه‌سازی چندمُدی را آورده و به بررسی و تحلیل آن‌ها خواهیم پرداخت.

۴-۱- توصیف توابع محک

در این مقاله، دو مجموعه از توابع محک استاندارد برای ارزیابی عملکرد روش پیشنهادی استفاده می‌شود. جدول (۱) اطلاعات عمومی ۱۵ تابع محک کلاسیک که در مقالات منتشر شده‌ی مختلف استفاده شده است را لیست می‌کند. توجه داشته باشید که این توابع همگی توابع بیشینه‌سازی هستند، یعنی هدف پیداکردن نقاطی از تابع هدف است که به‌ازای آن‌ها مقدار تابع هدف بیشینه می‌شود. جدول (۲) اطلاعات عمومی ۸ تابع محک پیچیده که اخیراً در [۳] ارائه شده‌اند را نشان می دهد. توجه داشته باشید که این توابع محک، همگی توابع کمینه‌سازی هستند، یعنی هدف پیداکردن نقاطی از تابع هدف است که به‌ازای آن‌ها مقدار تابع هدف کمینه می‌شود. در جدول‌های (۱) و (۲)، تعداد ابعاد و تعداد قله‌های سراسری (یا حتی محلی) توابع محک استفاده‌شده، آمده است. چنانکه دیده می‌شود توابع جدول (۲) برای ابعاد مختلفی قابل تعریف هستند، درصورتیکه توابع جدول (۱) چنین امکانی را به ما نمی دهند.

۴-۲- تنظیمات شبیه‌سازی

به‌دلیل این‌که پیداکردن راه‌حل‌های بهینه در توابع هدف پیوسته کار بسیار دشواری است، در تمام تحقیقات قبلی انجام‌شده، معمولاً یک سطح دقت (ε) در نظر گرفته می‌شود که نشان می‌دهد راه‌حل‌های پیداشده توسط الگوریتم چقدر به راه‌حل‌های بهینه‌ی واقعی نزدیک هستند. همچنین، برای اطمینان از این‌که بهینه‌های پیداشده با هم متفاوت

هستند، یک پارامتر شعاع نیچینگ (r) در نظر گرفته می‌شود. به‌عبارت دیگر، یک راه‌حل پیداشده توسط الگوریتم به‌عنوان یک راه‌حل بهینه گزارش می‌شود اگر همزمان دو شرط زیر برقرار باشد: (۱) اختلاف بین مقدار هدف آن راه‌حل با مقدار هدف یکی از راه‌حل‌های بهینه‌ی واقعی کمتر از اپسیلون (ε) باشد، و (۲) فاصله اقلیدسی بین آن راه‌حل با یکی از راه‌حل‌های بهینه‌ی واقعی کمتر از r باشد.

جدول (۱): مشخصات ۱۵ تابع محک کلاسیک

نام تابع محک	ابعاد	تعداد قله‌های سراسری
F1: Equal maxima [11]	۱	۵
F2: Decreasing maxima [11]	۱	۱
F3: Uneven maxima [11]	۱	۵
F4: Uneven decreasing maxima [11]	۱	۱
F5: Himmelblau's function [11]	۲	۴
F6: Six-hump camel back [11]	۲	۲
F7: Shekel's foxholes [11]	۲	۱
F8: Inverted Shubert function [11]	۲	۱۸
F9: Waves [11]	۲	۱۰
F10: Sphere	۱۰	۱
F11: Branin RCOS [11]	۲	۳
F12: Ackley [11]	۲	۱
F13: Michalewicz [11]	۲	۱
F14: Ursem F1 [11]	۲	۱
F15: Ursem F3 [11]		

جدول (۲): مشخصات ۸ تابع محک پیچیده [۱۶]

نام تابع محک	ابعاد	تعداد قله‌های سراسری/ محلی
F16: Shifted and rotated expanded two-peak trap	۵	۱۵/۱
	۱۰	۵۵/۱
	۲۰	۲۱۰/۱
F17: Shifted and rotated expanded Five-Uneven-Peak Trap	۲	۲۱/۴
	۵	۰/۳۲
	۸	۰/۲۵۶
F18: Shifted and rotated Expanded Equal Minima	۲	۰/۲۵
	۳	۰/۱۲۵
	۴	۰/۶۲۵
F19: Shifted and rotated Expanded Decreasing Minima	۵	۱۵/۱
	۱۰	۵۵/۱
	۲۰	۲۱۰/۱
F20: Shifted and rotated expanded uneven minima	۲	۰/۲۵
	۳	۰/۱۲۵
	۴	۰/۶۲۵
F21: Shifted and rotated expanded Himmelblau's function	۴	۰/۱۶
	۶	۰/۶۴
	۸	۰/۲۵۶
F22: Shifted and rotated expanded six-hump camel back	۶	۰/۸
	۱۰	۰/۳۲
	۱۶	۰/۲۵۶
F23: Shifted and rotated modified Vincent function	۲	۰/۳۶
	۳	۰/۲۱۶
	۴	۰/۱۲۹۶

در جدول (۳)، سطح دقت (ε)، شعاع نیچینگ (r)، اندازه جمعیت، و حداکثر تعداد ارزیابی تابع شایستگی برای هر یک از توابع محک فهرست شده است (این تنظیمات برای تمام الگوریتم‌های مورد مقایسه نیز معتبر است). به طور کلی، اندازه جمعیت و حداکثر تعداد ارزیابی تابع شایستگی به تعداد نقاط بهینه‌ی هر تابع وابسته هستند، به‌طوریکه تعداد زیاد نقاط بهینه نیازمند اندازه جمعیت بزرگتر و تعداد ارزیابی شایستگی بیشتری است. توجه داشته باشید که اندازه‌گیری عملکرد هر یک از الگوریتم‌ها به سطح دقت مشخص‌شده و شعاع نیچینگ وابسته است.

جدول (۳): تنظیمات شبیه‌سازی توابع محک

شماره تابع	تعداد ارزیابی تابع شایستگی	اندازه جمعیت	r	ε
F1	۱۰۰۰۰	۵۰	۰/۰۱	۰/۰۰۰۰۰۱
F2	۱۰۰۰۰	۵۰	۰/۰۱	۰/۰۰۰۰۰۱
F3	۱۰۰۰۰	۵۰	۰/۰۱	۰/۰۰۰۰۰۱
F4	۱۰۰۰۰	۵۰	۰/۰۱	۰/۰۰۰۰۰۱
F5	۱۰۰۰۰	۵۰	۰/۵	۰/۰۰۰۵
F6	۱۰۰۰۰	۵۰	۰/۵	۰/۰۰۰۰۰۱
F7	۱۰۰۰۰	۵۰	۰/۵	۰/۰۰۰۰۱
F8	۱۰۰۰۰۰	۲۵۰	۰/۵	۰/۰۵
F9	۳۰۰۰۰	۲۰۰	۰/۲	۰/۰۰۱
F10	۱۲۵۰۰	۵۰	۰/۲	۰/۰۱
F11	۲۰۰۰۰	۲۰۰	۰/۵	۰/۰۰۱
F12	۱۰۰۰۰	۱۰۰	۰/۵	۰/۰۱
F13	۱۰۰۰۰	۱۰۰	۰/۵	۰/۰۰۰۱
F14	۱۰۰۰۰	۱۰۰	۰/۵	۰/۰۰۰۰۰۱
F15	۲۰۰۰۰	۱۰۰	۰/۵	۰/۰۰۰۰۱
F16-F23	$۲۰۰۰ * D * \sqrt{q}$	$۵۰۰ * Round(\sqrt{D})$	۰/۱* D	۰/۱

۴-۳- نتایج و مقایسات

در این بخش، نتایج تجربی الگوریتم پیشنهادی بر روی توابع محک و همچنین نتایج سایر الگوریتم‌های بهینه‌سازی چندمُدی ارائه خواهد شد. جدول (۴) نتایج حاصل از شبیه‌سازی الگوریتم GLIPS پیشنهادی و ۱۰ الگوریتم بهینه‌ساز چندمُدی شامل الگوریتم‌های LIPS [۱۱]، r2PSO [۱۱]، r3PSO [۱۱]، r2PSO-LHC [۱۱]، r3PSO-LHC [۱۱]، SPSO [۱۶]، FER-PSO [۱۷]، SDE [۱۸]، CDE [۱۹]، و SACMA-ES [۱۶] را برای توابع کلاسیک F1-F15 نشان می‌دهد.

برای ارزیابی یک تابع محک توسط یک الگوریتم، از معیار نرخ موفقیت[۱۴] استفاده شده است. این معیار نشان می‌دهد که الگوریتم مورد نظر قادر است در چند درصد از اجراهای مستقلش توانسته است تمام نقاط بهینه‌ی تابع محک مورد بررسی را پیدا کند. مقدار ۱ برای نرخ موفقیت به این معناست که الگوریتم قادر به پیداکردن تمام بهینه‌ها در تمام اجراهای مستقلش است، و مقدار صفر برای نرخ موفقیت نشان می دهد که الگوریتم قادر نبوده است در هیچ یک از اجراهای مستقلش تمام نقاط بهینه‌ی تابع را پیدا کند. علاوه بر معیار نرخ موفقیت، رتبه‌ی هر الگوریتم در حل هر تابع محک نیز در درون پرانتز آمده است. همچنین، رتبه‌ی کلی هر الگوریتم در ردیف آخر جدول (۴) گزارش شده است. همان‌طور که از نتایج رتبه کلی الگوریتم‌ها دیده می‌شود، الگوریتم GLIPS بهترین کارایی را نسبت به سایر الگوریتم‌های بهینه‌سازی

چندمُدی مورد مقایسه خواهد داشت، به‌طوری‌که از نظر کارایی (همراه با الگوریتم LIPS) دارای اختلاف زیادی با سایر الگوریتم‌ها است.

جدول (۵) نتایج حاصل از پیاده‌سازی الگوریتم GLIPS و الگوریتم LIPS را توسط معیار متوسط تعداد بهینه‌های پیداشده برای توابع محک F16-F23 نشان می‌دهد. باید دقت داشت که توابع محک چندمُدی F16-F23 بسیار پیچیده‌تر از توابع F1-F15 هستند، به‌طوری‌که برای این توابع محک هیچ‌کدام از الگوریتم‌ها قادر به تولید نرخ موفقیت غیر صفر نیستند. به‌عبارت دیگر، معیار اندازه‌گیری نرخ موفقیت معیار مناسبی برای بررسی عملکرد الگوریتم‌های چندمُدی فعلی بر روی این توابع محک نیست، چراکه برای تمام الگوریتم‌ها مقدار صفر را گزارش می‌کند و در این حالت کارایی الگوریتم‌ها به هیچ وجه قابل مقایسه نیست. در این حالت، یک معیار اندازه‌گیری بهتر معیار متوسط تعداد بهینه‌های پیداشده توسط هر یک از الگوریتم‌ها است. این معیار میانگین تعداد بهینه‌هایی که یک الگوریتم در اجراهای مختلفش روی یک تابع محک پیدا می‌کند را گزارش می‌کند. همان‌طور که در جدول (۵) دیده می شود، الگوریتم GLIPS کارایی بهتری نسبت به الگوریتم LIPS دارد، به‌طوری‌که در اکثر توابع محک این الگوریتم توانسته است مقدار متوسط تعداد بهینه‌های بزرگتری را تولید کند.

در شکل‌های ۱ و ۲ توزیع ذرات جمعیت نهایی پیداشده توسط دو الگوریتم GLIPS و LIPS به‌ترتیب برای دو تابع محک F18 و F20 با تعداد ابعاد ۲ نمایش داده شده است. در این شکل‌ها نقاط آبی رنگ اعضای جمعیت نهایی پیداشده توسط هر الگوریتم را نشان می‌دهند و نقاط قرمز رنگ نقاط بهینه‌ی تابع محک مورد نظر را به تصویر می‌کشد. چنان‌که در هر دو شکل دیده می‌شود، الگوریتم پیشنهادی توانسته است یک تعادل بهتر بین همگرایی و تنوع راه‌حل‌ها در جمعیت نهایی پیداشده، ایجاد کند.

۵- نتیجه‌گیری

در این مقاله، با الهام از قاعده‌ی به‌روزرسانی سرعت الگوریتم جستجوی گرانشی، قاعده‌ی به‌روزرسانی سرعت در الگوریتم جمعیت ذرات اطلاع‌دهنده‌ی محلی بازتعریف شده و یک نسخه‌ی جدید از این الگوریتم با نام الگوریتم جمعیت ذراتِ اطلاع‌دهنده‌ی محلی گرانشی ارائه شده است. نتایج تجربی به‌دست‌آمده از شبیه‌سازی الگوریتم، عملکرد خوب آن را در مقایسه با الگوریتم جمعیت ذراتِ اطلاع‌دهنده‌ی محلی و دیگر الگوریتم‌های مشهور بهینه‌سازی چندمُدی نشان می‌دهد. به‌طور خلاصه، ویژگی اصلی الگوریتم پیشنهادی این است که در آن هر ذره موقعیت خود را به‌سمت موقعیت همسایگان محلی‌اش با استفاده از قوانین گرانش و حرکت به‌روزرسانی می‌کند. بنابراین، هرچه همسایه‌های محلی یک ذره کیفیت بالاتری داشته باشند یا فاصله‌ی کمتری با آن ذره داشته باشند، یک جرم گرانشی بیشتر به آن همسایه‌ها اختصاص داده می‌شود و در نتیجه آن همسایه‌ها مجاز به اعمال نیروی گرانش بیشتری به آن ذره هستند. در این حالت، ذراتی که در نزدیکی راه‌حل‌های خوب قرار دارند به جذب ذرات همسایه خود در فضای جستجو می‌پردازند. در نتیجه، الگوریتم پیشنهادی دارای قاعده‌ی به‌روزرسانی هوشمندانه‌تری در مقایسه با الگوریتم جمعیت ذراتِ اطلاع‌دهنده‌ی محلی است.

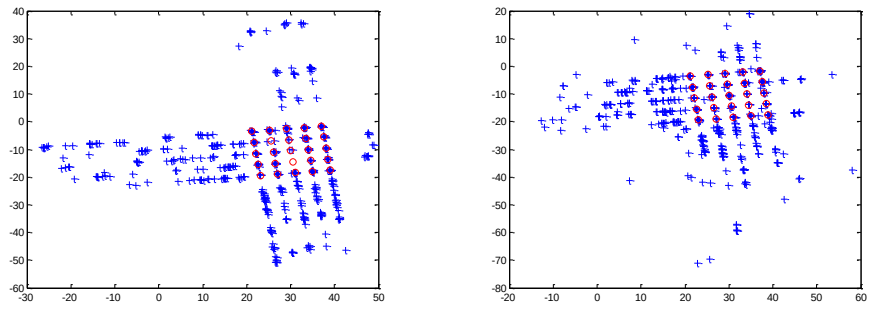
برای تحقیقات آینده، الگوریتم پیشنهادی می‌تواند بر روی مسائل بهینه‌سازی چندمُدی دنیای واقعی مانند انتخاب ویژگی، خوشه‌بندی داده‌ها، و غیره مورد بررسی قرار گیرد. همچنین، اثر بخشی الگوریتم پیشنهادی در حل مسائل بهینه‌سازی چندمُدی پویا می‌تواند مورد بررسی قرار گیرد. در نهایت، می‌توان قواعد به‌روزرسانی هوشمندانه‌تری برای به‌روزرسانی موقعیت ذرات در فضای جستجو تعریف کرد به‌نحوی که سایر پارامترهای مؤثر برای تنظیم تنوع و همگرایی را نیز در خود داشته باشند.

جدول (۴): مقایسه الگوریتم پیشنهادی با ۱۰ الگوریتم دیگر بر اساس معیار نرخ موفقیت

شماره تابع	SACMA-ES	CDE	SDE	FERPSO	SPSO	r3pso-lhc	r2pso-lhc	r3pso	r2pso	LIPS	GLIPS
F1	۰ (۱۱)	۰/۲۸ (۱۰)	۰/۷۲ (۹)	۰/۸۴ (۸)	۰/۸۸ (۶)	۰/۹۲ (۴)	۱ (۱)	۰/۸۸ (۶)	۰/۹۲ (۴)	۱ (۱)	۱ (۱)
F2	۰ (۷)	۰/۴۸ (۵)	۰ (۷)	۰ (۷)	۱ (۱)	۰/۰۴ (۶)	۰/۶۴ (۴)	۰ (۷)	۰ (۷)	۰/۹۶ (۳)	۱ (۱)
F3	۰ (۱۱)	۰/۲۸ (۱۰)	۰/۶۰ (۹)	۱ (۱)	۰/۹۲ (۴)	۰/۹۲ (۴)	۰/۹۲ (۴)	۰/۷۲ (۸)	۰/۸۸ (۷)	۱ (۱)	۱ (۱)
F4	۰/۹۶ (۱۱)	۱ (۱)	۱ (۱)	۱ (۱)	۱ (۱)	۱ (۱)	۱ (۱)	۱ (۱)	۱ (۱)	۱ (۱)	۱ (۱)
F5	۰/۴۴ (۵)	۰ (۱۰)	۰/۷۲ (۳)	۰/۷۲ (۳)	۰ (۱۰)	۰/۲۴ (۸)	۰/۲۸ (۶)	۰/۲۴ (۸)	۰/۲۸ (۶)	۱ (۱)	۱ (۱)
F6	۱ (۱)	۰ (۱۰)	۱ (۱)	۰/۹۶ (۵)	۰ (۱۰)	۰/۵۲ (۹)	۰/۵۶ (۷)	۰/۶۰ (۶)	۰/۵۶ (۷)	۱ (۱)	۱ (۱)
F7	۰ (۸)	۰ (۸)	۰ (۸)	۰ (۸)	۰/۹۲ (۳)	۰/۷۶ (۵)	۰/۸۴ (۴)	۰/۵۲ (۷)	۰/۶۰ (۶)	۱ (۱)	۱ (۱)
F8	۰ (۹)	۰/۷۲ (۳)	۰ (۹)	۰/۵۲ (۴)	۰ (۹)	۰/۲۰ (۵)	۰/۰۴ (۶)	۰/۰۴ (۶)	۰/۰۴ (۶)	۰/۸۴ (۲)	۸۸/ (۱)
F9	۰ (۱)	۰ (۱)	۰ (۱)	۰ (۱)	۰ (۱)	۰ (۱)	۰ (۱)	۰ (۱)	۰ (۱)	۰ (۱)	۰ (۱)
F10	۱ (۱)	۱ (۱)	۰ (۵)	۰ (۵)	۰ (۵)	۰ (۵)	۰ (۵)	۰ (۵)	۰ (۵)	۱ (۱)	۱ (۱)
F11	۱ (۱)	۰/۰۸ (۱۱)	۱ (۱)	۱ (۱)	۰/۶۴ (۱۰)	۰/۸۰ (۶)	۰/۷۶ (۹)	۰/۸۰ (۶)	۰/۸۰ (۶)	۱ (۱)	۱ (۱)
F12	۱ (۱)	۰ (۱۱)	۰/۹۶ (۵)	۱ (۱)	۰/۰۸ (۱۰)	۰/۷۲ (۷)	۰/۵۶ (۹)	۰/۸۸ (۶)	۰/۷۲ (۷)	۱ (۱)	۱ (۱)
F13	۰/۴۴ (۹)	۱ (۱)	۱ (۱)	۰/۰۴ (۱۰)	۰ (۱۱)	۰/۹۶ (۶)	۰/۹۲ (۸)	۱ (۱)	۰/۹۶ (۶)	۱ (۱)	۱ (۱)
F14	۰ (۶)	۰ (۶)	۰ (۶)	۰ (۶)	۰ (۶)	۰/۶۰ (۴)	۰/۷۶ (۳)	۰ (۶)	۰/۱۲ (۵)	۱ (۱)	۱ (۱)
F15	۰ (۳)	۰ (۳)	۰ (۳)	۰ (۳)	۰ (۳)	۰ (۳)	۰ (۳)	۰ (۳)	۰ (۳)	۱ (۱)	۱ (۱)
میانگین رتبه	۵/۶۶	۶/۰۶	۴/۶۰	۴/۲۶	۶/۰۰	۴/۹۳	۴/۷۳	۵/۱۳	۲/۲۰	۱/۲۰	**۱/۰۰**

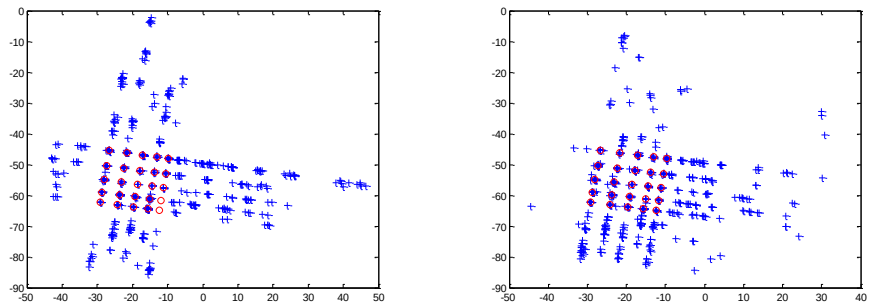
جدول (۵): مقایسه الگوریتم پیشنهادی با الگوریتم LIPS توسط معیار متوسط تعداد بهینه‌های پیداشده

شماره تابع	ابعاد	LIPS				GLIPS			
		بهترین	بدترین	میانگین	انحراف معیار	بهترین	بدترین	میانگین	انحراف معیار
F16	۵	۰	۰	۰	۰	۰	۰	۰	۰
	۱۰	۰	۰	۰	۰	۰	۰	۰	۰
	۲۰	۰	۰	۰	۰	۰	۰	۰	۰
F17	۲	۴	۳	۴	۰/۴۷	۵	۴	۴/۱	۰/۳۲
	۵	۰	۰	۰	۰	۰	۰	۰	۰
	۸	۰	۰	۰	۰	۰	۰	۰	۰
F18	۲	۲۳	۱۹	۲۱/۲۰	۱/۲۳	۲۵	۲۵	۲۵	۰
	۳	۳۰	۱۶	۲۲/۸۰	۴/۷۳	۷۹	۶۸	۷۵/۵۰	۳/۲۷
	۴	۲۵	۱۹	۲۳/۱۰	۱/۶۶	۱۱۷	۹۹	۱۱۰/۶۰	۵/۰۲
F19	۵	۰	۰	۰	۰	۰	۰	۰	۰
	۱۰	۰	۰	۰	۰	۰	۰	۰	۰
	۲۰	۰	۰	۰	۰	۰	۰	۰	۰
F20	۲	۲۵	۲۰	۲۱/۷۰	۱/۷۰	۲۵	۲۴	۲۴/۹۰	۰/۳۲
	۳	۳۱	۲۱	۲۵/۹۰	۲/۸۵	۸۵	۶۹	۷۸/۱۰	۵/۳۶
	۴	۳۱	۱۶	۲۳/۸۰	۵/۴۹	۱۲۷	۱۰۴	۱۱۳/۹۰	۸/۰۵
F21	۴	۲	۰	۰/۵۰	۰/۷۱	۹	۳	۶/۴۰	۱/۹۰
	۶	۴۵	۳۸	۴۰/۱۰	۱/۹۷	۳۸	۳۰	۳۳/۶۰	۲/۴۱
	۸	۷۰	۶۲	۶۵/۷۰	۲/۹۱	۴۵	۳۷	۴۱/۴۰	۲/۲۷
F22	۶	۰	۰	۰	۰	۱	۰	۰/۳۰	۰/۴۸
	۱۰	۱	۰	۰	۰/۳۲	۱۱	۵	۷/۹۰	۱/۹۱
	۱۶	۱	۰	۰/۱۰	۰/۳۲	۲	۱	۱/۳۰	۰/۴۸
F23	۲	۳۲	۲۶	۲۹/۱۰	۱/۹۱	۳۲	۲۵	۲۹/۱۰	۲/۲۳
	۳	۸۱	۶۸	۷۴/۶۰	۴/۴۸	۸۲	۷۰	۷۶/۸۰	۴/۱۸
	۴	۱۰۲	۸۰	۹۳/۲۰	۶/۰۵	۱۰۶	۹۱	۹۸/۳۰	۴/۶۹



(ب) (الف)

شکل ۱: جمعیت نهایی پیداشده برای تابع محک F18 با تعداد ابعاد ۲ توسط الگوریتم‌های: (الف) GLIPS و (ب) LIPS



(ب) (الف)

شکل ۲: جمعیت نهایی پیداشده برای تابع محک F20 با تعداد ابعاد ۲ توسط الگوریتم‌های: (الف) GLIPS و (ب) LIPS

مراجع

[1] J. Nocedal and S. Wright, *Numerical optimization*, Springer Science & Business Media, 2006.

[2] J.J. Liang, B.Y. Qu, P.N. Suganthan and Q. Chen, "*Problem definitions and evaluation criteria for the CEC 2015 competition on learning-based real-parameter single objective optimization*." Technical Report201411A, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore, 2014.

[3] B.Y. Qu, J.J. Liang, Z.Y. Wang, Q. Chen and P.N. Suganthan, "*Novel benchmark functions for continuous multimodal optimization with comparative results*," Swarm and Evolutionary Computation, vol. 26, pp. 23-34, 2016.

[4] E.G. Talbi, *Metaheuristics: from design to implementation*, John Wiley & Sons, 2009.

[5] K. Miettinen, P. Neittanmaki, M.M. Makela and J. Periaux, *Evolutionary algorithms in engineering and computer science*, John Wiley and Sons, Ltd, New York, 1999.

[6] J. Kennedy, R.C. Eberhart and Y. Shi, *Swarm intelligence*. Morgan Kaufmann, 2001.

[7] R.C. Eberhart and J. Kennedy, "*A new optimizer using particle swarm theory*," In Proceedings of the sixth international symposium on micro machine and human science, 1995.

[۸] عباسیان و نظام آبادی پور، «الگوریتم جستجوی گرانشی چندهدفه مبتنی بر مرتب‌سازی جبهه‌های مغلوب‌نشده»، مجله مهندسی برق، دوره ۴۱، شماره ۱، صفحات ۸۰-۶۸، دانشگاه تبریز، ۱۳۹۰.

[۹] شکرانی‌پور و افتخاری‌مقدم، «ACPSO: یک الگوریتم جدید بهینه‌سازی گروه ذرات تعاونی با قابلیت به‌روزرسانی تطبیقی پارامترها»، مجله مهندسی برق، دوره ۴۰، شماره ۲، صفحات ۳۶-۲۱، دانشگاه تبریز، ۱۳۸۹.

[10] M. Dorigo, *Optimization, learning and natural algorithms*, Ph.D. Thesis, Politecnico di Milano, Italy, 1992.

[11] B.Y. Qu, P.N. Suganthan and S. Das, "*A distance-based locally informed particle swarm model for multimodal optimization*," IEEE Transactions on Evolutionary Computation, vol. 17(3), pp. 387-402, 2013.

[12] R. Brits, A.P. Engelbrecht and F. Van den Bergh, "*A niching particle swarm optimizer*," In Proceedings of the 4th Asia-Pacific conference on simulated evolution and learning. Singapore: Orchid Country Club, 2002.

[13] E. Özcan and M. Yılmaz, "*Particle swarms for multimodal optimization*," In International Conference on Adaptive and Natural Computing Algorithms (pp. 366-375). Springer Berlin Heidelberg, 2007.

[14] A. Passaro and A. Starita, "*Particle swarm optimization for multimodal functions: a clustering approach*," Journal of Artificial Evolution and Applications, vol. 2008, 2008.

[15] X. Li, "*Niching without niching parameters: particle swarm optimization using a ring topology*," IEEE Transactions on Evolutionary Computation, vol. 14(1), pp. 150-169, 2010.

[16] X. Li, "*Adaptively choosing neighborhood bests using species in a particle swarm optimizer for multimodal function optimization*," in Proc. Genet. Evol. Computat. Conf., vol. 3102, pp. 105–116, 2004.

[17] X. Li, "*A multimodal particle swarm optimizer based on fitness Euclidean-distance ration*," in Proc. Genet. Evol. Computat. Conf., pp. 78–85, 2007.

[18] X. Li, "*Efficient differential evolution using speciation for multimodal function optimization*," in Proc. Conf. Genet. Evol. Computat., pp. 873–880, 2005.

[19] R. Thomsen, "*Multimodal optimization using crowding-based differential evolution*," in Proc. IEEE Congr. Evol. Computat, pp. 1382–1389, 2004.

زیرنویس‌ها

[1] Multimodal Optimization problems

[2] Population-based

[3] Single-solution based

[4] Evolutionary Algorithms (EAs)

[5] Stochastic

[6] Particle

[7] Particle Swarm Optimization (PSO)

[8] Gravitational Search Algorithm (GSA)

[9] Ant Colony Optimization (ACO)

[10] Locally Informed Particle Swarm (LIPS)

[11] Gravitational Locally Informed Particle Swarm (GLIPS)

[12] Niche

[13] Oscillation

[14] Success rate