# روش نوینی برای بهینه سازی در محیط های پویا مبتنی بر الگوریتم جهش قورباغه

سمیه رنجکش۱؛ محمدرضا میبدی۲؛ بهروز معصومی۳

## چکیده

**در بسیاری از مسائل بهینه‌سازی دنیای واقعی، تابع هدف، یا محدودیت‌ها می‌توانند در طول زمان تغییر یابند و بنابراین بهینهٔ این مسائل نیز می‌تواند تغییر یابد. اگر هر یک از این رویدادهای نامعین در فرآیند بهینه‌سازی مورد توجه قرار گیرند, این مسأله دینامیک یا پویا نامیده می‌شود. بسیاری از مسائل در دنیای واقعی پویا، غیرقطعی و پیچیده می باشند و حل آنها بصورت ایستا، به حل مسئله در دنیای واقعی کمکی نمی کند . در این گونه مسائل علاوه بر پیدا کردن بهینهٔ سراسری می بایست آن را در طول زمان دنبال نمود .در این مقاله یک روش نوین برای بهینه سازی در محیط پویا مبتنی بر الگوریتم جهش قورباغه پیشنهاد شده است .نتایج حاصل از رهیافت پیشنهادی بر روی معیار قله های متحرک که در حال حاضر شناخته شده ترین معیار برای ارزیابی در محیط های پویا می باشد ارزیابی شده و با نتایج حاصل ازچندین الگوریتم معتبر مورد مقایسه قرار گرفته است .نتایج بدست آمده نشان دهندهٔ کارایی بالای الگوریتم پیشنهادی در مقایسه با سایر الگوریتم ها می باشد.**

## کلمات کلیدی

**بهینه سازی، محیط های پویا، الگوریتم جهش قورباغه ، معیار حرکت قله ها.**

# A new Approach for Optimization in Dynamic Environments based on Frog Leaping Algorithm

Somayeh Ranjkesh; Mohammd Reza Meybodi; Behrouz Masomi

**ABSTRACT**

In many real-world optimization problems, the objective function, or constraints can be changed over time and thus Optimum these issues can also be changed. If any of these adverse events considered in the optimization process are given, the problem is called dynamic. Many real-world problems dynamically, statically indeterminate and are complex and solving them, the real world does not help solve the problem. In addition to these issues, it had to find the global optimum can be followed over time.

In this paper, a new approach for optimization in dynamic environments based on frog leaping algorithm is proposed. The results of the proposed approach on moving peaks benchmark the currently known criteria for evaluation in a dynamic environment is evaluated And the results were compared to validate multiple algorithms. Obtained results show high efficiency of the proposed algorithm is compared with other algorithms.

**KEYWORDS**

Optimization, Dynamic Environment,Frog Leaping Algorithm, Moving Peak Benchmark.

۱.دانشجوی کارشناسی ارشد هوش مصنوعی دانشگاه آزاد اسلامی واحد قزوین ،s.ranjkesh@yahoo.com
۲. عضو هیئت علمی دانشکده مهندسی کامپیوتر، فناوری اطلاعات، دانشگاه صنعتی امیرکبیر،mmeybodi@aut.ac.ir
۳. عضو هیئت علمی دانشکده برق-رایانه دانشگاه آزاد اسلامی واحد قزوین،masoumi_b@yahoo.com

## ۱. مقدمه

عدم قطعیت در بسیاری از مسائل در دنیای واقعی کاملا واضح و مشهود است .یکی از روشهای برخورد با عدم قطعیت استفاده از روشهای تکاملی و هوش جمعی می باشد .مسائل غیر- قطعی که تاکنون توسط روشهای تکاملی مورد بررسی قرار گرفته اند را بطور کلی میتوان به چهار دسته تقسیم نمود .وجود نویز در تابع ارزیاب، آشفتگی در متغیرهای طراحی، تقریبی بودن تابع ارزیاب و پویا بودن راهحل بهینه .در این مقاله عدم قطعیت از نوع پویا بودن راهحل بهینه که جزو عمومی ترین انواع عدم قطعیت می باشد، مورد توجه قرار گرفته است. تاکنون روشهای مختلفی برای حل مسائل پویا با استفاده از روشهای پردازش تکاملی[۱] و هوش جمعی[۲] [۳] ارائه شده است .

دو مشکل عمده روشهای پردازش تکاملی که باعث عدم توانایی استفاده مستقیم از این روش ها برای بهینه سازی در محیط های پویا شده است، حافظه غیرمعتبر و از دست رفتن تنوع می باشد .هنگامی که محیط تغییر می کند، راه حل های بدست آمده موجود در حافظه دیگر معتبر نمی باشند لذا می بایست آنها را بطور کامل فراموش نمود یا دوباره ارزیابی کرد. همچنین از آنجا که اکثر روش های پردازش تکاملی و هوش جمعی بدلیل ماهیتشان به یک نقطه همگرا میشوند لذا تنوع در جمعیتشان در محیط از بین میرود و در صورت تغییر در محیط، همگرا شدن به نقطه بهینه جدید در صورت امکان، بسیار زمانگیر خواهد بود . روشهای مختلفی برای ایجاد یا نگهداری تنوع دسته در محیط وجود دارد .

ساده ترین راه برای واکنش در برابر تغییر محیط، در نظر گرفتن هر تغییر به عنوان ورود یک مسأله بهینه- سازی جدید و حل آن از ابتدا می باشد .در صورت داشتن زمان کافی، این یک گزینه مناسب میباشد اما در بیشتر مواقع زمان کافی برای بهینهسازی مجدد وجود ندارد .یک روش مناسب برای تسریع فرآیند بهینه سازی پس از هر تغییر، استفاده از اطلاعات مربوط به جستجوی قبلی برای تسریع در امر جستجو پس از تغییر میباشد .معمولاً در اکثر مسائل تغییرات عمده نمی باشند و تغییر در محیط تنها باعث جابجایی کوچکی در بهینه جدید خواهد بود .در این مواقع روش اشاره شده بسیار مناسب می باشد .میزان اهمیت استفاده از اطلاعات گذشته بمنظور تسریه امر جستجو با میزان تغییر در محیط نسبت معکوس دارد . هر چه تغییر بیشتر باشد می بایست از اطلاعات گذشته کمتر استفاده شود تا فرایند بهینه سازی به مسیری گمراه کننده نرود.

مسئله دیگر این است که در بیشتر الگوریتم های بهینه سازی، پس از اینکه الگوریتم به یک نقطه همگرا شد، تنوع خود را از دست می دهد که این امر باعث کاهش انطباق پذیری و سازگاری الگوریتم در مقابل تغییر در محیط خواهد بود ، بنابراین در کنار انتقال اطلاعات بین عامل های الگوریتم های بهینه سازی بین دو تغییر در محیط، می بایست به دنبال راهکارهایی برای افزایش تنوع و سازگاری الگوریتم پس از تغییر در محیط نیز بود.

تاکنون برای بهینه سازی در محیط های پویا از روش های تکاملی و هوش جمعی مختلفی از جمله الگوریتم بهینه سازی الگوریتم ژنتیک، سیستم ایمنی مصنوعی وبهینه سازی دسته ذرات (PSO) و غیره استفاده شده است .الگوریتم بهینه سازی دسته ذرات در سال۱۹۹۵ توسط Eberhart وkennedy معرفی شد [۴] . از آن سال تاکنون نسخه های مختلفی از این الگوریتم پیشنهاد شده و بهبودهای مناسبی نیز بر روی آن داده شده است .از این الگوریتم برای بهینه سازی در محیط های پویا بسیار استفاده شده است [۷-۵].

در این مقاله یک الگوریتم بهینه سازی مبتنی بر الگوریتم جهش قورباغه پیشنهاد می گردد که برای عمل در محیط های پویا طراحی شده است و تمام الزامات محیط های پویا را برآورده می کند. الگوریتم پیشنهادی از راهکارهای خودتطبیقی نوینی برای رسیدن به نتایج بهتر بهره برده است .الگوریتم پیشنهادی بر روی بنچمارک قله های متحرک (MPB) [۸] که از معروفترین بنچمارک های محیط های پویا است به کار رفته و کارایی آن با سه الگوریتم دیگر به نام های mQSO [۵]، CellularPSO [۹] و Adaptive mQSO [۸] در فرکانس های مختلف با تعداد قله های مختلف مقایسه شده است . معیار مقایسه، خطای برون خطی است که یکی از معیارهای اصلی مقایسه الگوریتم های طراحی شده برای محیط های پویا می باشد [۱۰] . نتایج آزمایشات نشان می دهد که الگوریتم پیشنهادی از کارایی قابل قبولی برخوردار است.

ادامه این مقاله بدین ترتیب سازماندهی شده است .در بخش دوم الگوریتم ترکیبی جهش قورباغه معرفی می شود و در بخش سوم به تشریح الگوریتم پیشنهادی می پردازیم . در بخش چهارم نتایج آزمایشات مورد بررسی قرار میگیرند و بخش آخر به بیان نتیجه گیری می پردازد.

## ۲. الگوریتم ترکیبی جهش قورباغه

این الگوریتم بر اساس قوانین احتمالی، جستجوی تصادفی و جمعیت کار می کند. روند اجرای این الگوریتم برگرفته شده از رفتار قورباغه ها در تالاب ها برای یافتن غذا می باشد. هر قورباغه نشان دهندهٔ یک راه حل شدنی در فضای مسئله می باشد. موقعیت قورباغهٔ iام در فضای D بُعدی برابر $x_i=( x_{i,1}, x_{i,2},..., x_{i,D})$ می باشد و مقدار غذای موجود در موقعیت این قورباغه که همان مقدار شایستگی آن می باشد برابر $f(x_i)$ می باشد.موقعیت هایی که دارای شایستگی بیشتری هستند از مقدار غذای بیشتری برخوردارند و هدف قورباغه ها یافتن غذای بیشتر است.

بدین ترتیب قورباغه ها با اجرای روند SFLA سعی می کنند به غذای بیشتری دست یابند. ابتدا اعضای جمعیت بر اساس مقدار شایستگی شان، مرتب می شوند (بهترین قورباغه در اندیس یک قرار می گیرد).قورباغه ها به m گروه که ممپلکس نام دارند تقسیم می شوند. هر ممپلکس شامل n قورباغه خواهد بود. نحوهٔ تقسیم بندی جمعیت به m ممپلکس بدین ترتیب است که اولین قورباغه به اولین ممپلکس، دومین قورباغه به دومین ممپلکس و m امین قورباغه به m امین

ممپلکس داده خواهد شد. سپس (m+۱) امین قورباغه به ۱امین ممپلکس داده می شود و به همین ترتیب فرآیند دسته بندی ادامه می یابد تا در هر ممپلکس n قورباغه قرار بگیرند. پس از دسته بندی قورباغه ها در m ممپلکس، هر ممپلکس روند زیر را itr بار تکرار می کند.

در هر ممپلکس موقعیت بهترین و بدترین قورباغه از نظر مقدار شایستگی پیدا می شود که به ترتیب برابر با $X_b$ و $X_w$ می باشد. سپس یک موقعیت کاندید با استفاده از روابط (۱) و (۲) بدست می آید:

$$D = Rand.(X_b - X_W) \qquad (۱)$$

$$X'_W = X_W + D\,, \qquad D_{\min} \le D \le D_{\max} \qquad (۲)$$

rand یک تابع تولید عدد تصادفی با توزیع یکنواخت در بازهٔ [۱و۰] می باشد و برابر با طول گام جابه‌جایی است که مقدار آن در هر یک از ابعاد مسئله باید در محدودهٔ $[D_{min},D_{max}]$ باشد.

پس از بدست آوردن موقعیت ، مقدار شایستگی آن سنجیده می شود و در صورتی که مقدار شایستگی آن بهتر از موقعیت قبلی باشد، $X_w$ به موقعیت جدید خود حرکت می کند در غیر این صورت روابط (۱) و (۲) مجدداً اجرا می‌شوند با این تفاوت که به جای موقعیت $X_b$، از موقعیت بهترین قورباغه در میان تمام ممپلکس ها یعنی $X_g$ استفاده می شود.

اگر پس از این مرحله نیز مقدار شایستگی موقعیت بدست آمده از رابطهٔ (۲) بهتر از موقعیت قبلی $X_w$ نبود، این قورباغه مقداردهی اولیه می شود یعنی موقعیت جدید آن در فضای مسئله به صورت تصادفی تعیین می شود.

پس از انجام این مرحله توسط تمام ممپلکس ها (به تعداد itr بار)، شرط پایان الگوریتم سنجیده می شود و در صورتی که این شرط برقرار نشود، مجدداً تمام قورباغه ها بدون در نظر گرفتن اینکه عضو کدام ممپلکس بوده اند در یک دسته قرار می گیرند (shuffling) و این مراحل را اجرا می کنند.

## ۳. الگوریتم پیشنهادی در محیط پویا

در این بخش یک الگوریتم برای بهینه سازی در محیط های پویا پیشنهاد می گردد که در آن از SFLA به عنوان الگوریتم پایه استفاده شده است. در الگوریتم پیشنهادی مکانیزم های تطبیقی به کار رفته که چالش های محیط های پویا را رفع می کنند. در الگوریتم پیشنهادی از مکانیزم چنددستگی به صورت تطبیقی استفاده شده است به همین دلیل این الگوریتم را SFLA چنددسته ای تطبیقی یا amSFLA (Adaptive multi-swarm Frog Leaping Algorithm) می نامیم. محیط های پویا از جمله پر چالش ترین محیط ها برای بهینه سازی محسوب می شوند.

### ۱,۳. رفع چالش کشف تغییر در محیط

اولین چالشی که یک الگوریتم طراحی شده برای بهینه سازی در محیط های پویا باید با آن روبرو شود کشف تغییر محیط است. در واقع الگوریتم باید بتواند بدون داشتن دانش قبلی از زمان تغییر محیط، آنرا کشف کند. مکانیزم طراحی شده برای کشف تغییر محیط وابستگی بسیاری به محدودهٔ تغییرات دارد. در amSFLA از یک نقطهٔ تصادفی به نام test_point استفاده می شود. این نقطه در ابتدای اجراای الگوریتم به صورت تصادفی و با توزیع یکنواخت در فضای مسئله تعیین می شود و مقدار شایستگی آن محاسبه و ذخیره می‌گردد. در هر تکرار از اجرای amSFLA ، شایستگی test_point ارزیابی می گردد و مقدار بدست آمده از آن با مقدار ذخیره شده از تکرار قبلی مقایسه می گردد. در صورتی که مقدار بدست آمده با مقدار ذخیره شده مغایرت داشته باشد یعنی محیط تغییر کرده است. پس از کشف تغییر محیط، مقدار شایستگی ذخیره شده برای test_point بروز می شود. بدین ترتیب amSFLA قادر به کشف تغییر در محیط در هر تکرار از اجرای خود خواهد بود.

## ۲,۳. رفع چالش از بین رفتن تنوع

در amSFLA ، الگوریتم پایه SFLA است و مطابق با ساختار و ماهیت این الگوریتم ، diversity loss برای SFLA پس از همگرایی رخ می دهد. در چنین شرایطی همگی قورباغه ها در فاصلهٔ بسیار نزدیکی از یکدیگر قرار دارند و حول هدف جمع شده اند. در واقع این شرایط توسط روند الگوریتم با هدف افزایش توانایی جستجوی محلی در اطراف هدف به وجود می آید که با جابه جا شدن موقعیت هدف باعث بوجود آمدن مشکل می شود. در amSFLA ابتدا اجازه داده می شود diversity loss اتفاق بیافتد تا الگوریتم بتواند پیش از تغییر محیط، دقت نتیجه را تا جای ممکن افزایش دهد. پس از کشف تغییر محیط برای افزایش سرعت همگرایی به سمت موقعیت جدید هدف، تنوع در میان قورباغه ها ایجاد خواهد شد.

برای افزایش تنوع در دستهٔ قورباغه ها موقعیت بهترین قورباغه دسته در پیش از تغییر محیط حفظ می شود و قورباغه های دیگر در فضایی به شعاع $r_{div}$ به صورت تصادفی و با توزیع یکنواخت با استفاده از رابطهٔ (۳) پخش می شوند.

$$X_{i,j} = X_{Gbest,j} + (rand(-1,1) \times r_{div}) \qquad (۳)$$

که در این رابطهٔ مؤلفهٔ jام از قورباغه iام دسته به صورت تصادفی و بر اساس مؤلفهٔ jام موقعیت بهترین قورباغه دسته ($X_{Gbest}$) و پارامتر $r_{div}$ بدست می آید. تابع rand یک عدد تصادفی با توزیع یکنواخت در بازهٔ [۱و۱-] تولید می کند. مقدار $r_{div}$ بر اساس shift severity محیط تعیین می شود.

## ۳,۳. رفع چالش حافظهٔ نامعتبر

پس از کشف تغییر محیط، بهترین قورباغه دسته در موقعیت خود باقی می‌ماند و موقعیت قورباغه های دیگر در اطراف آن به صورت تصادفی تعیین می شود سپس مقدار شایستگی تمام قورباغه ها ارزیابی می شود و مقدار شایستگی آنها در محیط جدید در حافظه ذخیره می شود. بدین ترتیب مقدار شایستگی ذخیره شده در حافظه معتبر خواهد بود.

## ۴,۳. رفع چالش های وجود بهینه های بالقوه و مجهول بودن تعداد قله ها

در amSFLA از مکانیزم چنددستگی برای پوشش قله ها استفاده می شود. بدین ترتیب چندین دستهٔ قورباغه در فضای مسئله وجود خواهد داشت که به طور موازی و مستقل از یکدیگر فرآیند بهینه سازی را در محیط انجام می دهند. هر یک از دسته های قورباغه ها روند الگوریتم SFLA را تحت نظارت مرکزی amSFLA انجام می دهند[۱۱] [۱۲].

در amSFLA هر یک از دسته ها وظیفه دارد یکی از قله‌های موجود در فضای مسئله را تحت نظر داشته باشد. مکانیزم استفاده شده در این الگوریتم برای کنترل دسته‌ها به صورت تطبیقی می باشد. در amSFLA ابتدا یک دسته در فضای مسئله مقداردهی می شود و شروع به انجام فرآیند بهینه سازی می کند. پس از اینکه این دسته همگرا شد، یک دستهٔ دیگر در فضای مسئله ایجاد می شود. در واقع یک دستهٔ جدید هنگامی ایجاد می شود که تمام دسته های موجود در فضای مسئله همگرا شده باشند.

هنگامی یک دسته یک قله را یافته و به آن همگرا شده که فاصلهٔ اقلیدسی موقعیت بهترین قورباغه آن در تکرارm ام با موقعیت آن در تکرار m+n ام کمتر از یک حد آستانه به نام $r_{conv}$ باشد. یک دسته پس از همگرایی به یک قله، وظیفهٔ پوشش دادن این قله و تعقیب آن پس از تغییر محیط را بر عهده دارد. بدین ترتیب دسته های جدید در محیط به جستجوی قله هایی می پردازند که قبلاً کشف نشده باشند.

بدین ترتیب تعداد دسته های موجود در محیط منطبق بر تعداد قله ای است که الگوریتم آنها را کشف کرده باشد و انتظار می رود پس از مدتی تمام قله ها تحت پوشش دسته ها قرار بگیرند. در نتیجه با ایجاد دسته های جدید و کشف قله ها توسط آنها، چالش مجهول بودن تعداد قله ها رفع می شود.

همچنین با ساکن شدن دسته ها در قله ها، الگوریتم بر قله ها نظارت دارد و قله ای که پس از تغییر محیط تبدیل به بهینهٔ سراسری شود را به سرعت می یابد بنابراین چالش بهینه های بالقوه نیز در amSFLA رفع شده است.

### ۵،۳. رفع چالش همگرایی دو الگوریتم پایه به یک قله

در amSFLA این امکان وجود دارد که یک دستهٔ جدید به قله ای همگرا شود که قبلاً دستهٔ دیگری در آن ساکن باشد. در چنین حالتی مقدار شایستگی بهترین قورباغهٔ این دو دسته با یکدیگر مقایسه می شو و دسته ای که بهترین قورباغهٔ آن بهتر باشد در قله باقی می ماند و دستهٔ دیگر از بین می رود. در چنین حالتی اگر تمام دسته های موجود درفضای مسئله همگرا شده باشند، دستهٔ جدیدی در مسئله ایجاد می شود. هنگامی می گوییم دو دسته به یک قله همگرا شده اند که فاصلهٔ اقلیدسی موقعیت بهترین قورباغه آنها با یکدیگر کمتر از مقدار مشخصی به نام $r_{excl}$ باشد. مقدار $r_{excl}$ در الگوریتم پیشنهادی براساس [۱۳] تعیین می شود. بنابراین در هر تکرار از اجرای الگوریتم، فاصلهٔ اقلیدسی موقعیت بهترین قورباغه دستهٔ همگرا نشده (تازه ایجاد شده) با تمام دسته های قدیمی موجود در فضای مسئله محاسبه می شود و در صورتی که مقدار این فاصله با هر یک از دسته ها کمتر از $r_{excl}$ باشد، دستهٔ بدتر، از بین می رود. مکانیزم به کار رفته در این قسمت برای جلوگیری از همگرایی دو دسته به یک قله به نام انحصار شناخته می شود[۱۰].

### ۶،۳. رفع چالش سرعت همگرایی

در الگوریتم پیشنهادی برای افزایش کارایی و دادن فرصت بیشتر به دسته‌ای که نزدیک بلندترین قله است، از یک مکانیزم خواب-بیداری برای دسته‌ها استفاده می‌شود. در این مکانیزم، دسته ها می توانند در دو وضعیت متفاوت خواب و بیدار قرار داشته باشند. دستهٔ بیدار، دسته ای است که ذرات آن در فضای مسئله وجود دارند و برای انجام فرآیند جستجو، ارزیابی شایستگی انجام می دهد و قورباغه های آن در هر تکرار از اجرای الگوریتم حرکت می کنند. دستهٔ خواب، دسته ای است که قورباغه های آن در فضای مسئله وجود دارند اما حرکت نمی کنند و ارزیابی شایستگی انجام نمی دهند.

در این مکانیزم، پس از هر تغییر محیط، تمام دسته‌ها بیدار شده و فرآیند بهینه‌سازی را انجام می دهند. همانطور که پیش از این بحث شد، دسته‌هایی که در قله-های غیر بهینه ساکن هستند، باید پس از هر تغییر محیط، جستجوی محلی را انجام دهند تا فاصله‌شان از قلهٔ تحت پوشش آنها زیاد نشود. اما پس از نزدیک شدن به قلهٔ مورد نظر، ادامهٔ جستجوی محلی برای افزایش دقت کار مفیدی نیست و تأثیری در نتیجهٔ الگوریتم ندارد. بر همین اساس پس از اینکه یک دسته که در یک قلهٔ غیر بهینه ساکن است به نزدیکی هدفش رسید، به خواب می رود.

در واقع با انجام این مکانیزم، پس از اینکه دسته هایی که در محیط جاری، در قله های غیربهینه ساکن هستند به هدفشان نزدیک شدند، به خواب می روند و انجام ارزیابی شایستگی را تا تغییر محیط بعدی قطع می کنند. بدین ترتیب از انجام بیهودهٔ تعداد قابل توجهی ارزیابی شایستگی جلوگیری می شود و از آنها برای دادن فرصت بیشتر به دسته ای که در قلهٔ بهینه سراسری ساکن است استفاده می شود. این امر باعث می شود که جستجوی محلی در اطراف موقعیت بهینه سراسری بیشتر انجام شود و در نتیجه کارایی و دقت نتیجهٔ بدست آمده از کل الگوریتم بهبود یابد.

در الگوریتم پیشنهادی برای تشخیص اینکه آیا دسته‌ای به هدف خود نزدیک شده یا خیر از فاصلهٔ قورباغه های موجود در آن استفاده می‌شود. در واقع هنگامی که یک دسته به سمت موقعیت هدف همگرا می‌شود به نزدیکی آن می‌رسدفاصلهٔ بین قورباغه های آن کاهش می یابد.

در الگوریتم پیشنهادی اگر فاصلهٔ اقلیدسی تمام اعضای یک دسته با یکدیگر کمتر از پارامتر L باشد یعنی این دسته به نزدیکی هدف خود رسیده است و باید به خواب برود. شایان ذکر است که در هر محیط، دسته ای که دارای بهترین مقدار شایستگی Gbest در میان دسته ها است به خواب نمی رود. همچنین تمام دسته های خواب پس از کشف تغییر محیط بیدار می شوند. روند اجرای الگوریتم پیشنهادی در شکل (۱) نشان داده شده است.

## ۴. نتایج الگوریتم پیشنهادی

برای ارزیابی کارایی، از تابع محک قله های متحرک استفاده می کنیم. آزمایشات در دو قسمت اصلی انجام شده‌اند. قسمت اول به بررسی تأثیر پارامترهای مختلف در amSFLA می‌پردازد. در قسمت دوم به مقایسهٔ این الگوریتم با الگوریتم mQSO [۱۴] با پیکربندی $(5^9+5)10$، که به معنی استفاده از ۱۰ دسته و ۵ ذره خنثی و ۵ ذره کوآنتوم می باشد ، الگوریتم FMSO [۱۷]، الگوریتم CellularPSO [۱۵] و الگوریتم MQSO تطبیقی (Adaptive mQSO) [۱۶] در حالات مختلف MPB می‌پردازیم.

```
amSFLA
//Initializing first swarm
Foreach Frog j in first_swarm
    initialize X1,j randomly
endfor
initialize Test_Point randomly
repeat
foreach swarm i
     Execute an iteration of SFLA  procedure for  swarm i
Endfor
//Sleep
foreach awake swarm i
        if distance between all Forgs <rsleep then
              swarm i goes to sleep mode
        end if
 endfor
wake up best swarm
//Exclusion
foreach swarm i
        if distance(Gbesti and Gbestnew_swarm) <rexclthen
            if f(Gbesti) ≤ f(Gbestnew_swarm) then
                reinitialize Swarm i
            else
                reinitialize new_swarm
        end if
 endfor
//Convergence & activation
if All swarm are converged then
    Create new_swarm and initialize it
end if
//Test for Change
Evaluate Test_Point
if new value is different from last iteration then
      foreach swarm i
          if swarmi is converged
              keep best_Frogi and randomize others around it based on
Shift_length
          end if
        endfor
end if
Until stopping criterion is met
```

**شکل(۱) شبه کد الگوریتم پیشنهادی برای محیط پویا(amSFLA) .**

برای معیار مقایسه کارایی میان الگوریتم‌ها از خطای برون خطی استفاده می‌شود. خطای برون خطی برابر با متوسط شایستگی بهترین موقعیت یافت شده توسط الگوریتم در تمامی ارزیابی شایستگی‌ها می‌باشد. در واقع خطای برون خطی برابر با متوسط مقدار خطای استاندارد در طول اجرای الگوریتم است. در صورتی که در تمام ارزیابی شایستگی ها مقدار خطای استاندارد برابر صفر باشد، مقدار خطای برون خطی برابر صفر خواهد شد. خطای برون خطی با استفاده از رابطهٔ (۴) محاسبه می‌شود.

$$offline\_error = \frac{1}{Max\_FE}\sum_{t=1}^{Max\_FE}\left(f\left(Swarm_{best}(t)\right)\right) \qquad (۴)$$

که در آن Max_FE برابر با تعداد نهایی ارزیابی شایستگی است و Swarmbest(t) برابر بهترین موقعیت گروه در زمان ارزیابی شایستگی tام است. آزمایشات هر دو بخش ۳۰ بار اجرا شده است و هر بار تا ۱۰۰ بار تغییر محیط ادامه داشته است. آزمایشات هر بار با Random Seedهای مختلف انجام شده است.

از تابع قله های متحرک برای تست کارایی الگوریتم پیشنهادی استفاده شده است این تابع این قابلیت را دارد که ابعاد محیط،تعداد قله ها،شکل قله ها،میزان شدت تغییرات و فرکانس تغییرات تنظیم شود.تنظیمات پیش فرض برای آزمایشات درجدول(۱) آمده است.

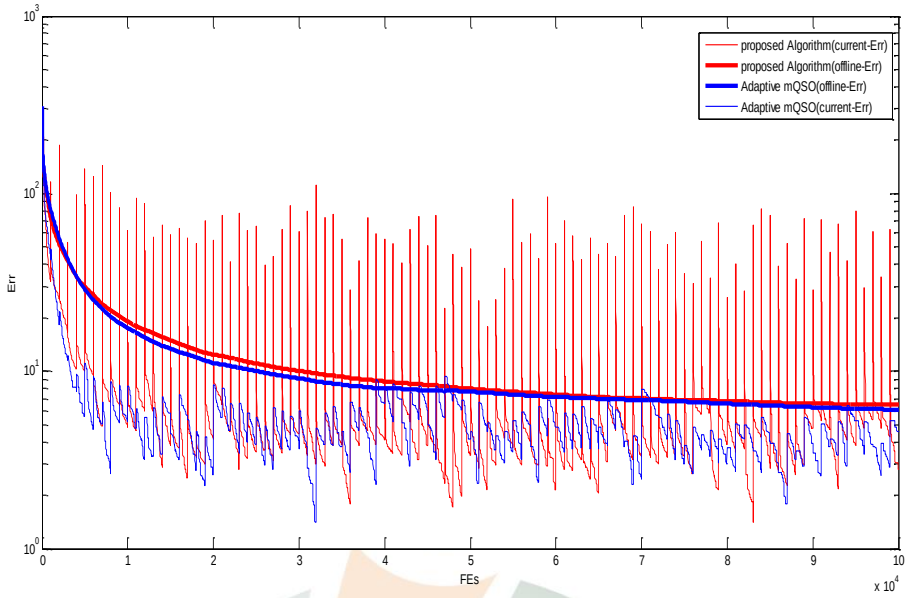
**جدول (۱) تنظیمات استاندارد برای تابع قله های متحرک**

| پارامتر | مقدار |
|---|---|
| تعداد قله ها=m | ۱۰ |
| فرکانس تغییرات | ۵۰۰۰ |
| شدت تغییرات ارتفاع قله ها | ۷ |
| شدت تغییرات عرض قله ها | ۱ |
| شکل قله ها | cone |
| میزان حرکت مکان قله ها | ۱ |
| ابعاد فضای جستجو | ۵ |
| محدوده ارتفاع قله ها | [۳۰و۷۰] |
| محدوده عرض قله ها | [۱و۱۲] |
| ارتفاع استاندارد قله ها | ۵۰ |
| محدوده ی فضای جستجو | [۰و۱۰۰] |

در جداول (۲) تا (۵) و همچنین در اشکال (۲) تا (۵) کارایی الگوریتم پیشنهادی در فرکانس‌ها و تعداد قله‌های مختلف با دیگر الگوریتم‌های معرفی شده در این بخش مقایسه شده است.

**جدول (۲) مقایسه کارایی الگوریتم پیشنهادی با روش های دیگر در فرکانس ۱۰۰۰.**

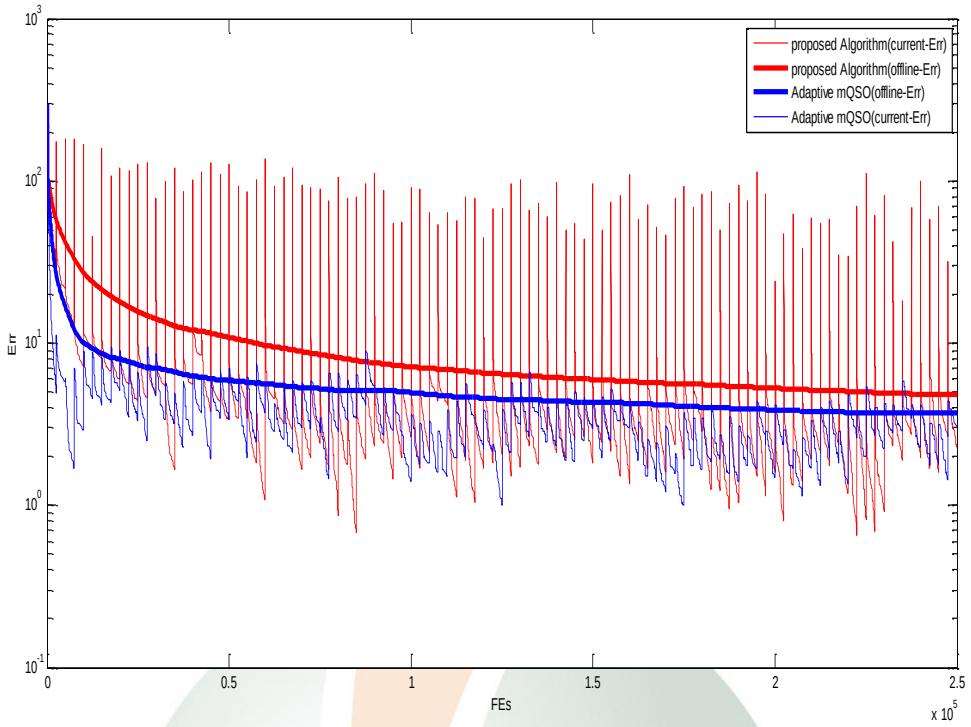
| m | MQSO(5,5$^{q}$) | CPSO | Adaptive mQSO | amSFLA |
|---|---|---|---|---|
| ۱ | ۱۷,۲۳±۱,۵۱ | ۸,۷۴±۰,۴۰ | ۲,۵۵±۰,۱۳ | ۳,۲۹±۰,۲۷ |
| ۵ | ۶,۵۹±۰,۴۱ | ۶,۵۶±۰,۳۸ | ۳,۴۷±۰,۰۹ | ۴,۳۲±۰,۵۴ |
| ۱۰ | ۵,۲۶±۰,۲۳ | ۵,۷۱±۰,۲۲ | ۴,۵۶±۰,۴۰ | ۴,۸۰±۰,۴۷ |
| ۲۰ | ۵,۷۹±۰,۱۶ | ۵,۹۵±۰,۱۱ | ۵,۳۶±۰,۴۷ | ۵,۶۹±۰,۷۵ |
| ۵۰ | ۶,۰۲±۰,۱۴ | ۷,۱۹±۰,۱۲ | ۶,۰۶±۰,۱۴ | ۶,۴۷±۰,۱۷ |
| ۱۰۰ | ۶,۲۲±۰,۱۶ | ۷,۲۸±۰,۱۱ | ۵,۲۶±۰,۱۲ | ۷,۹۵±۰,۹۷ |
| ۲۰۰ | ۶,۱۹±۰,۱۴ | ۷,۲۶±۰,۱۰ | ۵,۷۵±۰,۲۶ | ۷,۷۹±۰,۱۴ |




**شکل (۲) مقایسه نتایج الگوریتم پیشنهادی و الگوریتم Adaptive mQSO در فرکانس ۱۰۰۰ با ۵۰ قله.**

**جدول (۳) مقایسه کارایی الگوریتم پیشنهادی با روش های دیگر در فرکانس ۲۵۰۰.**

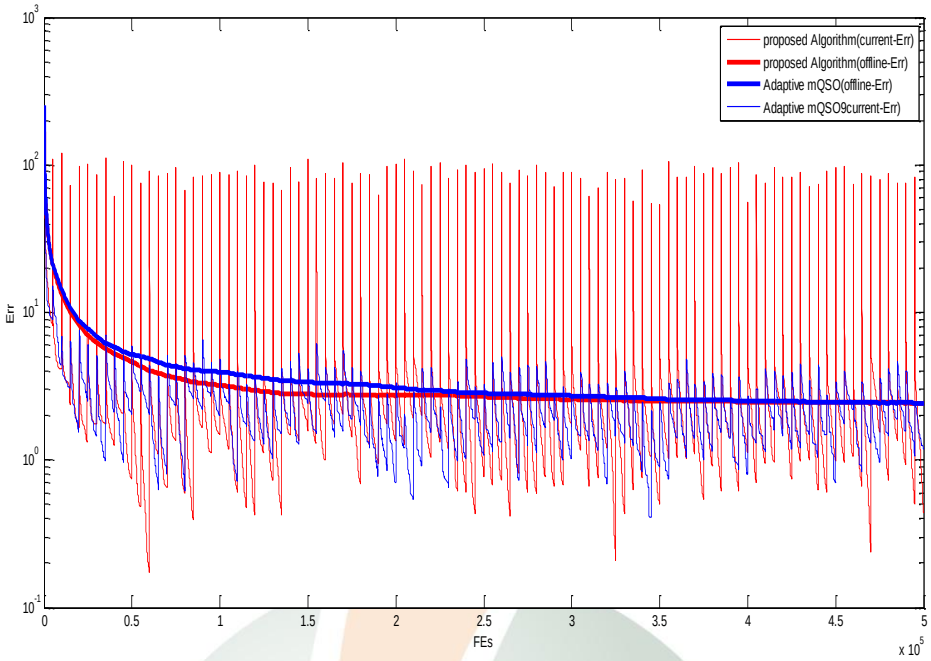
| m | MQSO(5,5[9]) | CPSO | Adaptive mQSO | amSFLA |
|---|---|---|---|---|
| ۱ | ۷٫۶۱±۰٫۶۶ | ۴٫۱۵±۰٫۲۵ | ۱٫۰۴±۰٫۰۵ | ۲٫۰۲±۰٫۲۳ |
| ۵ | ۳٫۱۸±۰٫۲۰ | ۲٫۵۸±۰٫۲۴ | ۲٫۱۶±۰٫۱۹ | ۱٫۱۰±۰٫۱۳ |
| ۱۰ | ۳٫۰۵±۰٫۱۴ | ۲٫۸۲±۰٫۱۴ | ۲٫۴۹±۰٫۱۰ | ۲٫۱۰±۰٫۱۳ |
| ۲۰ | ۳٫۷۳±۰٫۱۱ | ۳٫۴۱±۰٫۱۴ | ۳٫۱۱±۰٫۱۱ | ۳٫۱۳±۰٫۴۲ |
| ۵۰ | ۴٫۱۷±۰٫۱۲ | ۳٫۸۶±۰٫۱۰ | ۳٫۶۸±۰٫۱۵ | ۳٫۷۶±۰٫۱۸ |
| ۱۰۰ | ۴٫۴۲±۰٫۱۵ | ۴٫۱۰±۰٫۱۱ | ۳٫۵۳±۰٫۱۴ | ۴٫۴۸±۰٫۲۹ |
| ۲۰۰ | ۴٫۵۰±۰٫۱۳ | ۳٫۹۷±۰٫۱۰ | ۳٫۲۷±۰٫۰۶ | ۴٫۲۰±۰٫۰۴ |




**شکل (۳) مقایسه نتایج الگوریتم پیشنهادی و الگوریتم Adaptive mQSO در فرکانس ۲۵۰۰ با ۵۰ قله.**

**جدول (۴) مقایسه کارایی الگوریتم پیشنهادی با روش های دیگر در فرکانس ۵۰۰۰.**

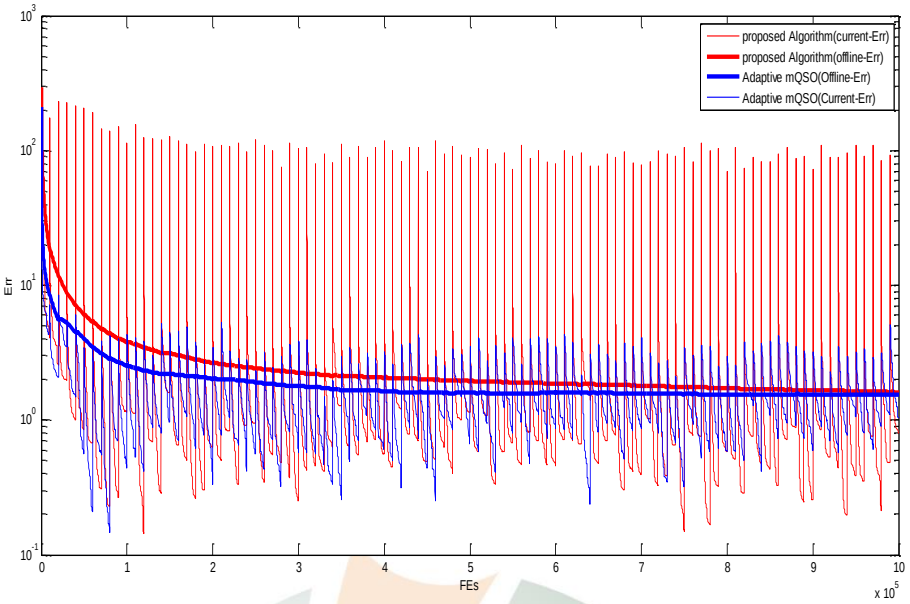
| m | MQSO(5,5$^{q}$) | CPSO | Adaptive mQSO | amSFLA |
|---|---|---|---|---|
| ۱ | ۳,۸۱±۰,۳۴ | ۲,۵۵±۰,۱۲ | ۰,۵۱±۰,۰۴ | ۲,۵۳±۰,۶۲ |
| ۵ | ۱,۹۰±۰,۱۴ | ۱,۶۸±۰,۱۱ | ۱,۱۶±۰,۰۹ | ۱,۰۲±۰,۳۱ |
| ۱۰ | ۱,۸۰±۰,۱۰ | ۱,۷۸±۰,۰۵ | ۱,۵۱±۰,۱۰ | ۱,۱۰±۰,۳۶ |
| ۲۰ | ۲,۹۶±۰,۱۲ | ۲,۶۱±۰,۰۷ | ۲,۱۸±۰,۱۵ | ۱,۷۶±۰,۱۸ |
| ۵۰ | ۳,۳۶±۰,۱۲ | ۲,۹۳±۰,۰۸ | ۲,۴۳±۰,۱۳ | ۲,۴۲±۰,۱۱ |
| ۱۰۰ | ۳,۷۰±۰,۱۴ | ۳,۱۴±۰,۰۸ | ۲,۶۸±۰,۱۲ | ۲,۹۰±۰,۱۱ |
| ۲۰۰ | ۳,۷۶±۰,۱۴ | ۳,۲۶±۰,۰۸ | ۲,۶۲±۰,۱۰ | ۳,۲۰±۰,۱۵ |




**شکل (۴) مقایسه نتایج الگوریتم پیشنهادی و الگوریتم Adaptive mQSO در فرکانس ۵۰۰۰ با ۵۰ قله.**

**جدول (۵) مقایسه کارایی الگوریتم پیشنهادی با روش های دیگر در فرکانس ۱۰۰۰۰.**

| anSFLA | Adaptive mQSO | CPSO | MQSO(5,5$^{q}$) | m |
|---|---|---|---|---|
| ۰,۲۵±۰,۰۴ | ۰,۱۹±۰,۰۲ | ۱,۵۳۰±۰,۱۲ | ۱,۹۰±۰,۱۸ | ۱ |
| ۰,۳۸±۰,۰۵ | ۰,۷۴±۰,۱۱ | ۰,۹۲±۰,۱۰ | ۱,۰۳±۰,۰۶ | ۵ |
| ۰,۶۶±۰,۰۶ | ۰,۷۶±۰,۰۶ | ۱,۱۹±۰,۰۷ | ۱,۱۰±۰,۰۷ | ۱۰ |
| ۰,۹۳±۰,۰۳ | ۱,۲۸±۰,۱۲ | ۲,۲۰±۰,۱۰ | ۱,۸۴±۰,۰۸ | ۲۰ |
| ۱,۶۱±۰,۰۷ | ۱,۵۵±۰,۰۸ | ۲,۶۰±۰,۱۳ | ۲,۰۰±۰,۰۹ | ۵۰ |
| ۱,۵۶±۰,۰۵ | ۲,۶۸±۰,۱۲ | ۲,۷۳±۰,۱۱ | ۱,۹۹±۰,۰۷ | ۱۰۰ |
| ۱,۹۷±۰,۰۳ | ۲,۵۲±۰,۱۰ | ۲,۸۴±۰,۱۲ | ۱,۹۹±۰,۷ | ۲۰۰ |




**شکل (۵) مقایسه نتایج الگوریتم پیشنهادی و الگوریتم Adaptive mQSO در فرکانس ۱۰۰۰۰ با ۵۰ قله.**

همانطور که مشاهده میشود نتایج نشان میدهد روش پیشنهادی نسبت به دو روش **Cellular PSO** و **MQSO**۱۰(۵+۵q) بهتر عمل می نماید و با الگوریتم **Adaptive mQSO** قابل مقایسه می باشد .هر چه تعداد قله ها بالاتر و فرکانس ارزیابی بالاتر باشد روش پیشنهادی نسبت به **Adaptive mQSO** جواب بهتری را ارائه میدهد.

## ۵. نتیجه گیری

در این مقاله یک روشی نوین برای بهینه سازی در محیط های پویا پیشنهاد شد و نتایج بر روی تابع معیار حرکت قلهها با چندین روش شناخته شده دیگر مورد مقایسه قرار گرفت .نتایج آزمایشات نشان داد که الگوریتم پیشنهادی از کارایی قابل قبولی مخصوصاً در مقابل روشهای **Cellular PSO** و **mQSO** که دارای جمعیت ثابتی بودند برخوردار بود اما نتایج آن در مواردی پایین تر از روش **Adaptive mQSO** بود .دلیل این امر، نیز برتری **Adaptive mQSO** در تطبیقی بودن تعداد دسته ها متناسب با قله ها می باشد . برای بهبود الگوریتم پیشنهادی میتوان تعداد دسته ها را در آن به صورت تطبیقی در نظر گرفت به طوری که به ازای هر قلهٔ یافت شده در فضای مسئله یک دسته ایجاد شود.

## ۶. مراجع


[۱] Y. Jin , J. Branke; “Evolutionary Optimization in uncertain environments –A Survey”, in IEEE Transaction on Evolutionary Computation, vol. ۹ , No. ۳ , pp. ۳۰۳-۳۱۷،۲۰۰۵.

[۲] C. Li , S. Yang; “Fast Multi-Swarm Optimization for Dynamic Optimization Problems”, In ٤th International Conference on Natural Computation, Jinan, Shandong, China, vol. ۷, pp.٦٢٤–٦٢٨, ۲۰۰۸.

[۳] S. Yang , C. Li; “A Clustering Particle Swarm Optimizer for Locating and Tracking Multiple Optima in Dynamic Environments”, in IEEE Transaction on Evolutionary Computation, pp. ۱-۱٦, ۲۰۱۰.

[٤] J. Kennedy , R. Eberhart; “Particle Swarm Optimization”, in IEEE International Conference on Neural Networks, Vol. ٤, pp. ۱۹٤۲-۱۹٤۸, Perth, November ۱۹۹٥.

[٥] T. Blackwell , J. Branke; “Multiswarm, Exclusion, and Anti-Convergence in Dynamic Environment”, in IEEE Transaction on Evolutionary Computation, Vol. ۱۰, No. ٤, pp. ٤٥۹-٤۷۲, ۲۰۰٦.

[٦] T. Blackwell , J. Branke; “Particle Swarms for Dynamic Optimization Problems”, in Swarm Intelligence, pp.۱۹۳–۲۱۷, ۲۰۰۸.

[۷] W. Du and B. Li, “Multi-Strategy Ensemble Particle Swarm Optimization for Dynamic Optimization”, in Information Sciences: an International Journal Vol.۱۷۸,pp.۳۰۹٦–۳۱۰۹, ۲۰۰۸.

[۸] B. Hashemi , M. R. Meybodi; “Cellular PSO: A PSO for Dynamic Environments”, in Advances in Computation and Intelligence, Lecture Notes in Computer Science, vol.٥۸۲۱, pp. ٤۲۲-٤۳۳, ۲۰۰۹.

[۹] T. Blackwell , J. Branke; “Particle Swarms for Dynamic Optimization Problems”, in Swarm Intelligence, pp.۱۹۳–۲۱۷, ۲۰۰۸.
[۱۰] http://www.aifb.unikarlsruhe.de/~jbr/MovPeaks/
[۱۱] Y. Shi , R. Eberhart; “A Modified Particle Swarm Optimization”, In IEEE International Conference on Evolutionary Computation Proceedings, pp. ٦۹-۷۳,Anchorage, ۱۹۹۸.
[۱۲] T. Blackwell , J. Branke; "Multiswarms, exclusion, and anti-convergence in dynamic environments," IEEE Transactions on Evolutionary Computation, vol. ۱۰, pp. ٤٥۹-٤۷۲, ۲۰۰٦.
[۱۳] T. Blackwell, J. Branke , X. Li; "Particle swarms for dynamic optimization problems," Swarm Intelligence, pp. ۱۹۳-۲۱۷, ۲۰۰۸.
[۱٤] T. Blackwell , J. Branke; “MultiswarmT Exclusion, and Anti-Convergence in Dynamic Environment”, in IEEE Transaction on Evolutionary Computation, Vol. ۱۰, No. ٤, pp. ٤٥۹-٤۷۲, ۲۰۰٦.
[۱٥] B. Hashemi , M. R. Meybodi; “Cellular PSO: A PSO for Dynamic Environments”, in Advances in Computation and Intelligence, Lecture Notes in Computer Science, vol. ٥۸۲۱, pp. ٤۲۲-٤۳۳, ۲۰۰۹.
[۱٦] M. A. Potter , K. A. de Jong; “A Cooperative Coevolutionary Approach to Function Optimization”, in The Third Parallel Problem Solving From Nature. Berlin, Germany: Springer-Verlag, pp. ۲٤۹–۲٥۷, ۱۹۹٤.
[۱۷] C. Li ,S. Yang; “Fast Multi-Swarm Optimization for Dynamic Optimization Problems,” in Fourth International Conference on Natural Computation, Jinan, Shandong, China, pp. ٦۲٤-٦۲۸, ۲۰۰۸.