اولین همایش ملی هوش مصنوعی و سیستم های فازی
The first National Conference
Artifical intelligence and fuzzy Systems
برای کسب اطلاعات بیشتر و نحوه ی ثبت نام در همایش به سایت WWW.Artintconf.ir همایش مراجعه نمایید.
مهلت ارسال کامل مقالات: ۱ مهرماه ۱۳۹۵
محل و زمان برگزاری: همدان – ۸ مهرماه ۱۳۹۵

مدل جدید موازی‌سازی الگوریتم اجتماع پرندگان برای حل مسائل عددی

حمیدرضا آل‌رضا امیری

دانشگاه آزاد اسلامی، واحد بابل، باشگاه پژوهشگران جوان و نخبگان، بابل، ایران

چکیده:

الگوریتم‌های تکاملی روش‌های بسیار مناسبی برای حل مسائل پیچیده و مشکل می‌باشند. همواره تغییراتی در ساختار استاندارد این الگوریتم‌ها برای رسیدن به جواب‌هایی مطلوب‌تر اعمال شده و نسخه‌های جدیدتر و با کیفیت‌تری از این الگوریتم‌ها معرفی شده است. استفاده از تکنیک موازی‌سازی برای کاهش زمان اجرای این الگوریتم‌ها و رسیدن به جواب‌هایی با کیفیت‌تر موضوع مهمی می‌باشد. در این مقاله قصد داریم تا یک مدل جدید موازی‌سازی را، مبتنی بر معماری حافظه‌ی مشترک معرفی کنیم. در مدل پیشنهادی قصد داریم تا با شکستن فضای مسئله به چندین ناحیه‌ی برابر مستقل از هم و اجرای موازی و همزمان یک الگوریتم بر روی هر کدام از این ناحیه‌ها به جواب‌هایی با کیفیت‌تری دست یابیم. کوچک شدن این ناحیه‌ها باعث می‌شود تا الگوریتم اجرا کننده در آن ناحیه جواب بهینه را بسیار راحتر پیدا کند. این در حالی است که در مدل سریال الگوریتم باید در کل فضای جست‌وجو، جواب بهینه را می‌یافت. برای تست از پنج بنچمارک معرف مسائل بهینه‌سازی عددی استفاده کرده‌ایم و مدل پیشنهادی خود را با مدل ایسلند که یکی از معروف‌ترین مدل‌های موازی‌سازی الگوریتم‌های تکاملی می‌باشد، مقایسه کرده و به جواب‌های با کیفیت‌تری دست یافتیم.

واژه‌های کلیدی: الگوریتم اجتماع پرندگان، الگوریتم اجتماع پرندگان غیرهمزمان تصادفی، مدل ایسلند، بهینه‌سازی، موازی‌سازی

۱- مقدمه

روش‌های بهینه‌سازی به طور گسترده در علوم و صنایع مختلف به کار گرفته می‌شوند. معروف‌ترین روش‌های بهینه‌سازی عبارتند از: بهینه‌سازی خطی، غیر خطی، صحیح و فرا ابتکاری. از بین این روش‌ها الگوریتم‌های فرا ابتکاری به دلیل سادگی و توانایی زیاد در حل مسائل NP بسیار مورد توجه قرار گرفته‌اند. از معروف‌ترین الگوریتم‌های فراابتکاری می‌توان به الگوریتم ژنتیک (Davis,1991)، الگوریتم کلونی مورچه‌گان (Dorigo,2009)، الگوریتم کلونی زنبور عسل (Karaboga,2007) و الگوریتم اجتماع ذرات (Eberhart,1995) را نام برد. از آنجایی که در بسیاری از مسائل در حوزه صنعت، افزایش بازده حتی به مقدار ناچیز منجر به صرفه‌جویی و سود زیادی می‌شود، پس همواره تلاش شده تا روش‌های بهینه‌سازی کاراتر و قدرتمندتر شده و این بهبود عملکرد را ممکن کنند. در همین راستا در بسیاری از تحقیقات سعی شده تا با اعمال تغییراتی روی ساختار استاندارد الگوریتم‌های فراابتکاری نتایج مطلوب‌تری از این روش‌ها حاصل شود (Ebrahimnejad ,2016, 48-56). مثلا در مورد الگوریتم اجتماع ذرات بخشی از کارهایی که انجام گرفته پیرامون تغییر متغیرها، مثل ضریب شتاب، تعداد ذرات و توپولوژی ذرات بود (Lee, 2007, 1120-1131) و بخش دیگری از تحقیقات هم پیرامون تغییر در ساختار الگوریتم اجتماع ذرات و قادر کردن این الگوریتم بر حل مسائل گسسته بود (Ebrahimnejad, 2015, 203-222). عمده‌ی این تغییرات برای رسیدن به جواب‌های با کیفیت‌تر، زمان اجرای کوتاه‌تر و محاسبات کمتر بود. یکی از روش‌های کاهش زمان اجرای الگوریتم‌ها، موازی‌سازی آنها هست. موازی‌سازی را می‌توان از منظرهای مختلف بررسی کرد. از نظر معماری، ماشین‌ها به دو دسته‌ی ماشین‌های حافظه مشترک و ماشین‌های حافظه‌ی توزیع شده تقسیم می‌شوند. در ماشین‌های حافظه‌ی مشترک تمام پردازنده‌ها به حافظه اشتراکی دسترسی دارند و ارتباط بین آنها از طریق خواندن و نوشتن بر روی حافظه‌ی اشتراکی صورت می‌گیرد. در ماشین‌های حافظه‌ی توزیع شده، هر پردازنده حافظه‌ی محلی مخصوص به خودش را دارد و ارتباط بین پردازنده‌ها توسط تبادل پیام از طریق شبکه صورت می‌گیرد (Yazdani,, 2010, 678-687). دو نوع موازی‌سازی وجود دارد، موازی سازی داده و موازی سازی وظایف. در موازی‌سازی داده، یک مجموعه یکسان از دستورات بر روی مجموعه‌های مختلفی از داده‌ها اعمال می‌شوند. این نوع از موازی‌سازی می‌تواند به راحتی بر روی ماشین‌های حافظه‌ی اشتراکی اعمال شود. در موازی سازی وظایف، برنامه به چندین وظیفه‌ی مجزا تقسیم می‌شود. هر کدام از این وظایف کد مخصوص به خود را دارند، که در صورت حفظ انسجام برنامه می‌توانند همزمان اجرا شوند. خط لوله نمونه بارز موازی-سازی وظایف می‌باشد (Harman, 2012, 1-59). در بخش دوم کارهای انجام شده برای موازی‌سازی الگوریتم‌های تکاملی بخصوص الگوریتم اجتماع ذرات را مورد بررسی قرار می‌دهیم. در بخش سوم ساختار الگوریتم اجتماع ذرات و تغییراتی که بر روی نسخه‌های مختلف این الگوریتم برای بهبود سرعت انجام گرفته را بررسی می‌کنیم. در بخش چهارم مدل پیشنهادی خود را در معماری حافظه‌ی مشترک و درقالب موازی‌سازی داده‌ها شرح می‌دهیم. در بخش پنجم، شش بنچمارک استفاده شده برای تست الگوریتم را معرفی خواهیم کرد و نتایج بدست آمده را مورد تحلیل قرار می‌دهیم. در بخش ششم هم به نتیجه‌گیری و پیشنهاد کارهایی که در آینده می-توان در این حوزه انجام داد، می‌پردازیم.

۲- کارهای مرتبط

مقالاتی که برای موازی‌سازی الگوریتم‌های تکاملی مخصوصا الگوریتم اجتماع ذرات انجام شده است، در سه مدل کلی دسته‌بندی می-شوند (Laguna-Sánchez, 2009, 293-307) :

۱-Farming model or master-slave در این مدل یک پردازنده به نام ارباب روال کلی برنامه را در دست می‌گیرد و برای کاهش زمان اجرا، محاسبات را بین پردازنده‌های برده توزیع می‌کند و بعد نتایج را از آنها تحویل می‌گیرد. این مدل در صورتی مفید می‌باشد که تعداد پردازنده‌ها کم باشد یا زمان اجرای محاسبات، بسیار زیاد باشد. چون در غیر این صورت، هزینه‌ی سربار ارتباط بین پردازنده‌ها می‌تواند خیلی بیشتر از مزایای موازی‌سازی شود.

۲-Island model در این مدل، جمعیت ذرات به چند زیرجمعیت تقسیم می‌شوند و هر پردازنده الگوریتم اجتماع ذرات را بر روی زیر جمعیت خودش به طور مستقل اجرا می‌کند. بعد از تعداد مشخصی از تکرارها عمل مهاجرت، انجام می‌شود. به این صورت که پردازنده-ها موقعیت بهترین ذره‌ی خود را برای بقیه‌ی ایسلندها ارسال می‌کنند تا آنها هم از بهترین نتایج بدست آمده تا حالا استفاده کنند. در این مدل سیاست مهاجرت بسیار تعیین کننده هست و در صورتی که یک ایسلند در بهینه‌ی محلی گرفتار شود ممکن است با دریافت موقعیت بهترین ذره از همسایه‌های خود از بهینه‌ی محلی فرار کند. این مدل بسیار محبوب بوده و در اکثر مقالات مورد استفاده قرارگرفته است.

۳-Diffusion model این مدل بر گرفته شده از مدل ایسلند می‌باشد با این تفاوت که جمعیت هر ایسلند فقط دارای یک ذره می-باشد. یعنی تعداد کل ذرات با تعداد ایسلندها برابر می‌باشد. در این مدل عمل بروزرساتی موقعیت ذرات با سیاست‌های خاصی انجام می‌شود.

موازی‌سازی هر کدام از این مدل‌ها توسط معماری حافظه‌ی توزیع شده، محاسبات را در گیر دو مشکل نا خواسته می‌کند: ۱. ناهمگونی پردازنده‌ها: همیشه پردازنده‌های با توان پردازشی پایین‌تر، بقیه‌ی پردازنده‌ها را در انتظار خود نگه می‌دارند و به عنوان گلوگاه سرعت عمل می‌کنند. ۲. تاخیر شبکه: چون تبادل اطلاعات بین پردازنده‌ها از طریق شبکه می‌باشد و امکان تاخیرهای پیش‌بینی نشده بسیار زیاد است. اولین مورد موازی‌سازی الگوریتم اجتماع ذرات در سال ۲۰۰۴ انجام گرفت (Schutte, 2004). موازی کردن این الگوریتم همچنان موضوع جالب و مهیجی هست و از آن برای حل مسائل پیچیده‌ی بهینه‌سازی استفاده می‌شود (Mussi, 2011, 1555-1562). یکی از مقالاتی که معماری حافظه‌ی توزیع شده را با پردازنده‌های ناهمگن در نظر گرفت، مقاله (Gelado, 2010, 347-358) بوده است. در این مقاله مدل ایسلند با سیاست‌های مهاجرتی مختلف و ساختارهای همسایگی متفاوت بررسی شده است. در (Bingyu, 2006) برای حل مسائل پیچیده‌ی بهینه‌سازی از الگوریتم موازی اجتماع ذرات در معماری حافظه توزیع شده بهره برده شده است. در این مقاله از مدل ایسلند و سیاست مهاجرت استفاده شده و ارتباط پردازنده‌ها از طریق تبادل پیام صورت گرفته است. در مقاله‌ای مشابه الگوریتم موازی اجتماع ذرات در مدل ایسلند پیاده‌سازی شده و ضمن بهبود کیفیت جواب‌ها، از همگرایی‌های زودرس هم جلوگیری شده است (Li, 2015, 1310-1317). در مقاله (Chu, 2003) برای بهبود مدل ایسلند سه استراتژی مهاجرت مطرح شده تا از سربار ارتباط بین پردازنده‌ها بکاهد و بین مهاجرت ذرات و سربار ارتباط پردازنده‌ها تعادل برقرار کند. در این مقاله (Moraes, 2013) نویسندگان برای بالا بردن توانایی الگوریتم ازدحام پرندگان در حل مسائل پیچیده حالت موازی شده و غیرهمزمان این الگوریتم را معرفی کردند. این الگوریتم که AIU-PPSO نامیده شد بر اساس معماری توزیع شده و مدل مستر اسلیو پیاده‌سازی شد. برقراری ارتباط در این الگوریتم بر اساس روش MPI بوده است. در مقاله (Tian, 2014, 309-318) نویسندگان یکی از نسخه‌های بهبود یافته‌ی الگوریتم ازدحام پرندگان را که QPSO نامیده می‌شود را بر اساس معماری حافظه‌ی توزیع شده, هم در مدل ایسلند و هم در مدل مستر اسلیو در دو حالت همزمان و غیرهمزمان برای حل مشکل شناسایی شکل معکوس[1] استفاده کرده‌اند. در این مقاله QPSO همزمان موازی به جواب‌های با کیفیت‌تری دست یافت در حالی که QPSO غیرهمزمان موازی از نظر سرعت اجرا سریعتر به کارش خاتمه داد.

[1] Inverse shape identification problem

۳- ساختار الگوریتم اجتماع ذرات

برای اولین بار ایبرهات و کندی الگوریتم اجتماع ذرات را به عنوان یک روش جدید بهینه‌سازی معرفی نمودند (Eberhart,1995). در این روش محاسباتی که بر پایه مشاهده رفتار جمعی بعضی از گروه‌های حیوانات مخصوصا دسته پرندگان شکل گرفته است هر فرد در اجتماع با توجه به اطلاعات مشترک در میان اعضا و اطلاعات فردی خود حرکت بعدی خود را شکل می‌دهد. از این جهت الگوریتم اجتماع ذرات شامل یک جمعیت متشکل از ذرات فرضی است که در یک فضای جست‌وجوی با سرعت‌های تنظیم شده به طور پویا در حال حرکت فرضی می‌باشند. سرعت ذرات برای به‌روز کردن موقعیت‌های مربوطه به کار می‌رود. هر ذره یک حافظه محدود دارد که بهترین موقعیت تجربه پرواز خودش را ذخیره می‌کند.

یکی از جنبه‌های کلیدی و جذاب الگوریتم اجتماع ذرات سادگی آن است، به گونه‌ای که فقط شامل دو معادله سرعت و مکان می‌باشد. موقعیت هر ذره نمایان‌گر یک جواب مسئله است. بردارهای موقعیت و سرعت دو بردار مرتبط با هر ذره در فضای جستجو N بعدی هستند که به ترتیب به صورت $x_i = [\, x_{i1}, x_{i2}, \dots, x_{in}]$ و $v_i = [\, v_{i1}, v_{i2}, \dots, v_{in}]$ نشان داده می‌شوند. موقعیت سرعت هر ذره بر اساس بهترین جستجوی ذره، بهترین تجربه کلی پرواز گروهی و بردار سرعت ذره، مطابق روابط زیر به‌روز می‌شود.

$$v_i^{k+1} = v_i^k + c_1 r_1 \left(pbest_i^k - x_i^k\right) + c_2 r_2 \left(gbest^k - x_i^k\right) \quad (1)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \quad (2)$$

که در آن c_1 و c_2 دو ثابت مثبت هستند. r_1 و r_2 دو عدد تصادفی با توزیع یکنواخت در محدوده‌ی [۰-۱] است. W وزن لختی یا اینرشی است که می‌تواند یک مقدار ثابت باشد و تعیین کننده‌ی مقدار اثرگذاری سرعت قبلی ذره می‌باشد.

$pbest_i^k$ بهترین موقعیت ذره i است که بر اساس تجربه این ذره به دست می‌آید.

$gbest^k$ بهترین موقعیت ذره بر اساس تجربه کلی گروهی است. و همچنین k شاخص تکرار است. ساختار شبکه، بیان کننده‌ی نحوه‌ی همسایه‌گی ذرات می‌باشد. در مقاله (Engelbrecht, 2007, 1-13) چندین توپولوژی الگوریتم اجتماع ذرات معرفی شده‌اند.

۳-۱- الگوریتم اجتماع ذرات همزمان

ذرات در الگوریتم اجتماع ذرات استاندارد، اطلاعات کاملی از همسایه‌گان خود دارند. یعنی همه‌ی ذرات موقعیت بهترین ذره در فضای مسئله را می‌دانند و با توجه به این دانش به سمت بهترین موقعیت پیدا شده حرکت می‌کنند. این الگوریتم که، الگوریتم استاندارد اجتماع پرندگان می‌باشد Synchronous PSO (SPSO) نیز نامیده می‌شود. شبه کد این الگوریتم را در شکل (۱) مشاهده می‌کنید.

```
while not stopping condition do
    foreach Particle p in swarm do
     p.update();
    end
    foreach Particle p in swarm do
     p.evaluate();
     p.communicate()
    end
end
```

شکل (۱) شبه کد الگوریتم اجتماع پرندگان همزمان

در حلقه‌ی اول هر ذره براساس معادلات ۱و۲ بردار سرعت و موقعیت خودش را بروز رسانی می‌کند. پس از پایان حلقه‌ی اول، در حلقه-ی دوم هر ذره در متد evaluate میزان مطلوب بودن موقعیتش را می‌سنجد و بعد در متد communicate اگر موقعیت فعلیش از موقعیت قبلی خودش یا بهترین موقعیت همسایگانش بهتر بود، مقدار آنها را با مقدار جدید عوض می‌کند.

۳-۲- الگوریتم اجتماع ذرات غیر همزمان

این الگوریتم که Asynchronous PSO (APSO) نامیده می‌شود، تفاوت اصلیش با حالت همزمان در اطلاعات موجود از ذرات همسایه می‌باشد. در به‌روزرسانی‌های غیرهمزمان ذرات اطلاعات ناقصی از موقعیت بهترین ذرات همسایه دارند. در شکل (۲) شبه کد این الگوریتم توضیح داده شده است.

```
while not stopping condition do
    foreach Particle p in swarm do
     p.update();
     p.evaluate();
     p.communicate()
    end
end
```

شکل (۲) شبه کد الگوریتم اجتماع ذرات غیرهمزمان

در این الگوریتم هر ذره به محض انتخاب شدن ابتدا با اطلاعاتی که تا این لحظه از ذرات همسایه‌ش دارد، بردار سرعت و موقعیت خودش را به‌روزرسانی می‌کند و بعد میزان مطلوبیت موقعیتش را می‌سنجد و سپس در صورت بهتر بودن موقعیت فعلیش نسبت به موقعیت pbest و gbest مقدار آنها را تغییر می‌دهد. این الگوریتم نسبت به SPSO درحالاتی که شرط خاتمه‌ی الگوریتم تعداد ثابتی از تکرارها باشد زمان اجرایی کمتری دارد و در مسائلی که زمان اجرا خیلی مهم است می‌تواند جایگزین خوبی برای SPSO شود.

۳-۳- الگوریتم اجتماع ذرات غیر همزمان تصادفی

این الگوریتم که Random Asynchronous PSO (RAPSO) نامیده می‌شود، در هر تکرار ذرات را به طور تصادفی برای پردازش انتخاب می‌کند (Rada-Vilela, 2011, 220-225). انتخاب تصادفی ذرات هم با توزیع یکنواخت و مستقل از انتخاب‌های قبلی صورت می‌گیرد. یعنی در هر تکرار، به تعداد ذرات جمعیت هر بار یک ذره به صورت تصادفی انتخاب می‌شود تا مورد بروزرسانی، ارزیابی و برقراریی ارتباط قرار بگیرد. در نتیجه ممکن است در یک تکرار بعضی از ذرات بیش از یک بار انتخاب شوند و بعضی از ذرات هم اصلا انتخاب نشوند. در شکل (۳) شبه کد این الگوریتم را مشاهده می‌کنید.

```
while not stopping condition do
    for i=1:size(swarm) do
     Particle p=randomParticle(swarm);
     p.update();
     p.evaluate();
     p.communicate()
    end
end
```

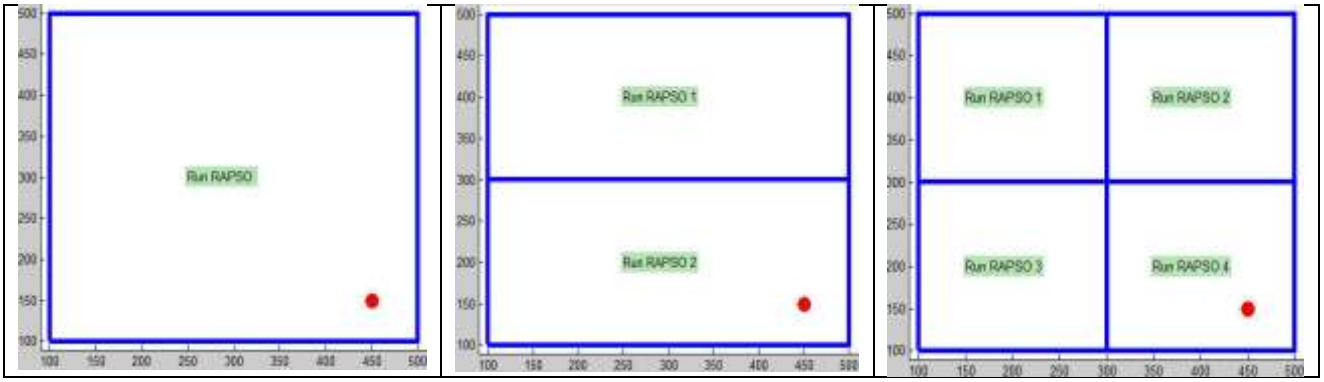
شکل (۳) شبه کد الگوریتم اجتماع ذرات غیر همزمان تصادفی

۴- مدل پیشنهادی

همان طور که ذکر شد معماری توزیع شده دارای دو مشکل تاخیرهای ناخواسته‌ی شبکه و ناهمگونی پردازنده‌های اجرا کننده می‌باشد. از این رو مدل پیشنهادی را مبتنی بر معماری حافظه‌ی مشترک و با هدف استفاده‌ی بهینه از توانایی پردازنده‌های چند هسته‌ای معرفی می‌کنیم.

در الگوریتم اجتماع ذرات یک نکته از نظر تجربی و شهودی قابل درک است و آن هم این موضوع است که برای رسیدن به جواب بهینه‌ی مسئله، اندازه‌ی فضای جست‌وجو با تعداد تکرارها و تعداد ذرات الگوریتم رابطه‌ی مستقیم دارد، یعنی در حالت کلی هر چه فضای جست‌وجو بزرگتر باشد برای رسیدن به جواب بهینه، الگوریتم باید با تعداد ذرات و تعداد تکرارهای بیشتری اجرا شود. که این موضوع باعث افزایش زمان اجرا نیز می‌شود. از این رو در مدل پیشنهادی قصد داریم تا فضای مسئله‌ی جست‌وجو را به چند ناحیه‌ی مجزای برابر تکه تکه کنیم و همزمان به طور موازی بر روی هر یک از تکه‌ها یک الگوریتم اجتماع ذرات مجزا را اجرا کنیم. در نهایت

بعد از خاتمه‌ی همه‌ی الگوریتم‌های موازی اجرا شده، جواب حاصل از همه‌ی الگوریتم‌ها را با هم مقایسه می‌کنیم و بهترین جواب بدست آمده از آنها را به عنوان جواب کلی مسئله در نظر می‌گیریم. چون نقطه‌ی بهینه‌ی مسئله حداقل در یکی از این ناحیه‌ها قرار دارد، الگوریتمی که در آن ناحیه اجرا می‌شود، بسیار راحتر از حالت اجرای سریال می‌تواند جواب بهینه را پیدا کند، زیرا ناحیه‌ی مورد جست‌وجو بسیار کوچکتر شده است. به دلیل اینکه ناحیه‌ی مورد جست‌وجو برای هر الگوریتم بسیار کوچکشده است و شانس رسیدن به جواب بهینه افزایش پیدا کرده است، پس می‌توانیم تعداد تکرارهای الگوریتم‌ها را نیز در این حالت مقداری کاهش دهیم تا باعث کاهش زمان اجرا نیز بشویم. این تکنیک هم باعث بهبود کیفیت جواب می‌شود و هم زمان اجرا را کاهش می‌دهد. می‌توانیم اجرای هر کدام از این الگوریتم‌ها را به یک هسته واگذار کنیم. در شکل (۴) نمونه‌ای از شکستن فضای جست‌وجو و تخصیص الگوریتم به هر کدام از این ناحیه‌ها را مشاهده می‌کنید. فرض شده دایره‌ی قرمز رنگ داخل شکل نقطه‌ی بهینه‌ی مسئله را نشان دهد. در شکل (۴-الف) کل فضای جست‌وجو را مشاهده می‌کنید. در اجرای سریال یک الگوریتم بر روی کل این فضا به منظور یافتن جواب بهینه اجرا می‌شود. در شکل (۴-ب) فضای مسئله به دو ناحیه‌ی مجزای برابر شکسته شده و دو الگوریتم به طور موازی هم، هر کدام در یک ناحیه اجرا می‌شوند و پس از خاتمه‌ی اجرای آنها و مقایسه‌ی جواب حاصل از آن دو، جواب کلی مسئله بدست می‌آید. واضح است که چون نقطه‌ی بهینه در ناحیه‌ی مورد کاوش الگوریتم ۲ قرار دارد، پس جواب حاصل از الگوریتم ۲ باید بهینه‌تر باشد و به عنوان جواب کلی مسئله انتخاب شود. در شکل (۴-ج) فضای مسئله به چهار ناحیه تقسیم می‌شود. در حالت سریال اگر مثلا الگوریتم در کل فضا با ۵۰۰ تکرار اجرا می‌شود، در حالتی مانند شکل(۴- ج) هر الگوریتم در یک چهارم کل فضا جست‌وجو می‌کند و می‌توان تعداد تکرارهای آن را مثلا به ۴۰۰ کاهش داد. که در این صورت هم زمان اجرا و هم کیفیت جواب‌ها بهبود چشمگیر پیدا می‌کند. در شکل (۴) فرض شده است که الگوریتم RAPSO بر روی هر ناحیه اجرا می‌شود.



شکل (۴- الف) شکل (۴-ب) شکل (۴-ج)

تفاوت مدل پیشنهادی با اجرای سریال در این است که در حالت سریال هم از توانایی‌های سخت‌افزار در اجرای موازی استفاده نمی‌شود و هم اینکه، یک الگوریتم مجبور بود کل فضای مسئله را جست‌وجو کند، در حالی که در مدل پیشنهادی با اجرای همزمان چند الگوریتم مجزا در ناحیه‌های مجزای کوچک شانس پیدا کردن جواب بهینه را بسیار بیشتر کرده‌ایم و دیگر به تکرارهای زیاد نیاز نداریم. فرق مدل پیشنهادی با مدل ایسلند هم در این است که، در مدل ایسلند فضای مسئله و تعداد تکرارها (مانند حالت سریال) برای همه‌ی الگوریتم‌ها یکسان است و فقط جمعیت بین الگوریتم‌ها تقسیم می‌شود. در حالی که در مدل پیشنهادی فضای مسئله بین الگوریتم‌ها تقسیم می‌شود و الگوریتمی که مسئول اجرا در ناحیه‌ی است که جواب بهینه در آن قرار دارد راحت‌تر می‌تواند آنرا بیابد. روش ایسلند از

سیاست مهاجرت استفاده می‌کند، که در یافتن جواب بهینه به آن مدل کمک می‌کند ولی از طرفی تبادل پیام در دفعات زیاد بین ایسلندها موجب کندی و افزایش زمان اجرا می‌شود. در این مدل جمعیت اولیه بین همه‌ی ایسلندها توزیع می‌شود، افزایش تعداد ایسلندها باعث می‌شود تا ایسلندهایی با جمعیت کم داشته باشیم، کم بودن تعداد ذرات در الگوریتم باعث ایجاد ضعف در جست‌وجو می‌شود. در مدل پیشنهادی چون الگوریتم‌ها از هم مستقل اجرا می‌شوند نیازی به تبادل پیام در تکرارهای مختلف و مهاجرت نداریم و فقط در انتهای کار الگوریتم‌ها بهترین جواب حاصل شده‌ی خود را برای تعیین جواب کلی مسئله به هم ارسال می‌کنند. مدل پیشنهادی به نوعی موازی‌سازی بر روی داده است.

۵- پیاده‌سازی و تحلیل نتایج

برای تست الگوریتم پیشنهادی از پنج بنچمارک معروف مسائل بهینه‌سازی استفاده کردیم که در جدول (۱) آنها و جزییاتشان را مشاهده می‌کنید (Jamil, 2013, 150-194). تمام این بنچمارک‌ها، توابع مینیمم‌ای هستند و نقطه‌ی صفر جواب مینیمم مطلق‌شان است. مدل پیشنهادی را بر روی یک کامپیوتر با سخت افزارهای CPU Intel core i7 1.6 GHz و RAM 4 GB و سیستم عامل ویندوز ۷ و نرم افزار متلب R2010a پیاده سازی شده است.

جدول (۱) لیست بنچمارک‌ها.

نام تابع	فرمول	محدوده
Hyper Ellipsoid	$\sum_{i=1}^{n} i x_i^2$	$-5.12 < x_i < 5.12$
Spherical	$\sum_{i=1}^{n} x_i^2$	$-5.12 < x_i < 5.12$
Rastrigin	$10n + \sum_{i=1}^{n} x_i^2 - 10\cos(2\pi x_i)$	$-5.12 < x_i < 5.12$
Ackley	$20 + e - 20\exp[-.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\,] - \exp[\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)]$	$-32.76 < x_i < 32.76$
Griewank	$1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}})$	$-600 < x_i < 600$

در اولین پیاده‌سازی عملکرد سه نسخه‌ی سریال معرفی شده‌ی الگوریتم اجتماع پرندگان را مورد بررسی قرار خواهیم دادیم. در جدول (۲) مقدار پارامترهای مورد استفاده در شبیه‌سازی را مشاهده می‌کنید. در ارزیابی نسخه‌های سریال هر الگوریتم ۲۰ بار اجرا شد، میانگین بهترین جواب، انحراف معیار و میانگین زمان اجرایش را به ازای هر بنچمارک در جدول (۳) مشاهده می‌کنید. در جدول (۳)

بهترین جواب‌های حاصله بر روی هر بنچمارک به صورت بولد شده نشان داده شده است. نتایج نشان می‌دهد که الگوریتم RAPSO کیفیت جواب‌های بهتری داشته است. هر چند تفاوت عملکرد در سه نسخه‌ی الگوریتم چندان چشمگیر نبود ولی بهترین عملکرد را در کیفیت جواب الگوریتم RAPSO و سپس APSO و در نهایت SPSO یا همان نسخه‌ی استاندارد الگوریتم داشته است. در معیار زمان اجرا هم تفاوت قابل ملاحظه‌ای بین سه الگوریتم وجود نداشته است.

جدول (۲) پارامترهای شبیه‌سازی

پارامترها	مقادیر
ابعاد بنچمارک‌ها	۳۰
ضریب شتاب	$c_1=c_2=2$
اینرسی	$w=1$
ساختار ارتباطی	ستاره‌ای
حداکثر سرعت	محدوده حرکت × ۰,۲
تعداد ذرات	۶۰
تعداد تکرارها	۵۰۰
ابعاد بنچمارک‌ها	۳۰

جدول (۳) عملکرد نسخه‌های سریال الگوریتم اجتماع پرندگان

	اجرای سریال								
	الگوریتم اجتماع ذرات همزمان			الگوریتم اجتماع ذرات غیر همزمان			الگوریتم اجتماع ذرات غیر همزمان تصادفی		
بنچمارک‌ها	میانگین	انحراف معیار	زمان اجرا	میانگین	انحراف معیار	زمان اجرا	میانگین	انحراف معیار	زمان اجرا
Hyper ellipsoid	2.1e-4	1.1e-3	1.7	3.7e-7	7.8e-7	1.65	***4.5e-8***	5.5e-12	1.85
Spherical	1.1e-4	5.2e-3	4.22	3.2e-7	1.2e-6	4.50	***1.2e-7***	4.5e-7	4.35
Rastrigin	165.9	31.73	4.99	***119***	30.60	4.61	130	38.16	4.92
Ackley	2.56	0.62	4.04	1.80	0.882	4.64	***1.78***	0.50	4.61
Griewank	13.77	32.96	9.42	***4.54***	20.11	9.71	4.55	20.14	9.07

از آنجایی که در آزمایش نسخه‌های سریال، الگوریتم RAPSO بهترین کیفیت جواب را داشته، پس مدل پیشنهادی را با این الگوریم اجرا می‌کنیم و آن را با مدل ایسلند مقایسه می‌کنیم. نتایج حاصل از این مقایسات را در جدول (۴) مشاهده می‌کنید. در مدل ایسلند جمعیت کل ذرات که ۶۰ ذره بوده بین ایسلندها تقسیم می‌شود. افزایش تعداد ایسلندها باعث کاهش تعداد ذرات در هر ایسلند می‌شود مثلا در ایسلند ۴ تایی تعداد ذرات در هر ایسلند ۱۵ ذره می‌باشد. کم شدن تعداد ذرات یعنی در مسائل پیچیده فضا به خوبی جستوجو نمی‌شود و شانس رسیدن به جواب بهینه کاهش پیدا می‌کند. از این رو مشاهده می‌کنید که در سه بنچمارک مالتی‌مدل که بنچمارک-های پیچیده‌ای (سه بنچمارک انتهای جدول) هستند در ایسلند ۲ تایی و ۴ تایی جواب‌های با کیفیت‌تری نسبت به ایسلند ۸ تایی حاصل شده است.

در دو بنچمارک بالای جدول که توابع یونی‌مدل هستند ایسلندهای با تعداد ذره‌ی کم هم موفق به یافتن جواب‌های مناسب شده‌اند از طرف دیگر کاهش تعداد ذرات در ایسلندها باعث کاهش زمان اجرای الگوریتم‌ها نیز می‌شود. زمان اجرا در مدل ۴ ایسلندی به علت کمتر بودن تعداد ذرات، از مدل ۲ ایسلندی کمتر بوده است. در مدل ۸ ایسلندی با اینکه کمترین تعداد ذرات را در اجراهایش داشته ولی به علت مهاجرت‌های زیاد بین ایسلندها زمان اجرا افزایش پیدا کرده است، در واقع سربار موازی‌سازی باعث کندی اجرا شده است. سیاست مهاجرت مدل ایسلند در پیاده‌سازی به صورت حلقه در نظر گرفته شده است. مدل پیشنهادی را در سه حالت شبیه‌سازی کردیم. در حالتی که فضای جست‌وجوی به ۸ قسمت شکسته می‌شد و ۸ الگوریتم همزمان بر روی هر کدام از ناحیه‌ها اجرا می‌شدند، بهترین جواب‌ها حاصل شده است. دلیل این موفقیت هم کوچک‌تر شدن ناحیه‌ی مورد جست‌وجو بوده است. کنترل سرعت ذرات و محدود کردن آن در مدل پیشنهادی بسیار مهم می‌باشد، مخصوصا وقتی که فضای جست‌وجو بسیار کوچک می‌شود. چون در این شرایط در بروزرسانی ذرات الگوریتم شاهد بیرون زدگی‌های پی‌درپی از ناحیه خواهیم بود و مجبوریم که به جای ذره‌ی بیرون زده یک موقعیت تصادفی تولید کنیم. افزایش بیرون زدگی‌ها باعث می‌شود تا سیر حرکت ذره به سمت نقطه‌ی بهینه با مشکل جدی رو به رو شود و ذره حرکات تصادفی داشته باشد. تعداد ذرات در تمام اجراهای مدل پیشنهادی ۶۰ ذره و تعداد تکرارها در حالت دو ناحیه ۴۰۰ تکرار، در حالت ۴ ناحیه ۳۵۰ تکرار و در حالت ۸ ناحیه‌ای برابر ۳۰۰ تکرار بوده است. علت کاهش تعداد تکرارها هم به خاطر کاهش زمان اجرا بوده است. چون فضای مورد جست‌وجو هر بار کوچکتر شده پس تعداد تکرارها را هم کمتر کرده‌ایم. مدل پیشنهادی در تمام بنچمارک‌ها جواب‌های بهتری نسبت به مدل ایسلند داشته است و در سه بنچمارک موفق به یافتن مینیمم مطلق یا همان صفر شده است. در جدول (۴) از شش حالت شبیه‌سازی شده به ازای هر بنچمارک، آن حالتی که بهترین جواب را در کمترین زمان یافته است به صورت ضخیم شده و مورب نشان داده شده است.

جدول (۴) نتایج مقایسه‌ی مدل پیشنهادی و مدل ایسلند

	مدل پیشنهادی									مدل ایسلند								
	۸ الگوریتمی			٤ الگوریتمی			۲ الگوریتمی			۸ ایسلند			٤ ایسلند			۲ ایسلند		
	زمان	انحراف معیار	میانگین	زمان	انحراف معیار	میانگین	زمان	انحراف معیار	میانگین	زمان	انحراف معیار	میانگین	زمان	انحراف معیار	میانگین	زمان	انحراف معیار	میانگین
Hyp	1.7	0	0	***1.6***	***0***	***0***	1.1	5e-7	3e-7	0.67	1e-4	1e-4	0.7	3.e-6	2 e-6	1.1	3e-9	1e-7
Sph	2.4	0	***0***	1.9	0	0	***1.7***	***0***	***0***	0.7	٢.e-4	1.e-4	0.73	3.e-6	2.e-6	1.12	3.e-9	1.e-9
Ras	***2.5***	***11.8***	***35.1***	1.8	6.53	49.1	1.5	54.6	94.4	0.56	34.8	150.9	0.73	37.5	122.4	0.94	34.0	109.3
Ack	2.2	1.83	0.69	***1.6***	***2.35***	***0.6***	1.6	4.94	1.87	0.57	0.77	3.73	0.67	0.78	1.56	1.02	3.7	1.67
Gri	8.7	0	0	***6.6***	***0***	***0***	6.8	0	0	2.48	0.19	0.47	3.08	0.019	***0.022***	4.57	0.028	0.01

۶- نتیجه‌گیری

در این مقاله ابتدا چند نسخه‌ی بهبود یافته‌ی الگوریتم اجتماع ذرات را بررسی کرده و سپس یک مدل جدید را برای موازی‌سازی الگوریتم‌های بهینه‌سازی مطرح کردیم. در مدل پیشنهادی خود تلاش کردیم تا فضای مسئله را به چندین ناحیه‌ی مستقل شکسته، و

سپس الگوریتم‌های مجزایی را به طور موازی در هر ناحیه اجرا کنیم. پس از خاتمه‌ی اجرای تمام الگوریتم‌ها، جواب حاصل از آنها با هم مقایسه می‌شوند و بهترین جواب از بین آنها به عنوان جواب مسئله در نظر گرفته می‌شود. اجرای هر کدام از الگوریتم‌ها را می‌توان به یک هسته واگذار کرد. کوچک‌تر شدن ناحیه‌ها نسبت به کل فضای جست‌وجوی اولیه باعث می‌شود تا عمل جست‌وجو بهتر انجام شود و جواب‌های با کیفیت‌تری حاصل شود. برای بررسی عملکرد، مدل پیشنهادی را بر روی پنج بنچمارک معروف مسائل عددی اجرا کردیم و مدل پیشنهادی خود را با مدل ایسلند که یکی از معروف‌ترین مدل‌های موازی‌سازی هست مقایسه‌کرده‌ایم و مشاهده کردیم که مدل پیشنهادی به جواب‌های با کیفیت‌تری دست پیدا کرده است. تفاوت‌ها و مزایای مدل پیشنهادی با مدل ایسلند به طور کامل شرح داده شد. از دیگر کارهایی که در این حوزه قصد داریم انجام دهیم ترکیب مدل پیشنهادی با مدل ایسلند و مدل ارباب-برده می‌باشد. همین طور قصد بررسی بیشتر مدل پیشنهادی با بنچمارک‌های پیچیده‌تر را خواهیم داشت.

۷- منابع

Bingyu, Z. Y., & Chuansheng, Z. (2005)؛ «A Parallel Particle Swarm Optimization Algorithm Based on Multigroup for Solving Complex Functions Optimization», *Computer Engineering and Applications*, *16*, 018.

Chu, S. C., Roddick, J. F., & Pan, J. S. (2003)؛ «Parallel particle swarm optimization algorithm with communication strategies», submitted *to IEEE Transactions on Evolutionary Computation*.

Davis, L (1991)؛ «The handbook of genetic algorithms Van Nostrand Reingold», *New York*.

Dorigo, M, and Thomas S (2009)؛ «Ant colony optimization: overview and recent advances», *Techreport, IRIDIA, Universite Libre de Bruxelles*.

Eberhart, R. C., & Kennedy, J. (1995)؛ «A new optimizer using particle swarm theory», In *Proceedings of the sixth international symposium on micro machine and human science* ,Vol. 1, pp. 39-43.

Ebrahimnejad, A., Karimnejad, Z., & Alrezaamiri, H. (2015)؛ «Particle swarm optimisation algorithm for solving shortest path problems with mixed fuzzy arc weights», *International Journal of Applied Decision Sciences*, *8*(2), 203-222.

Ebrahimnejad, A., Tavana, M., & Alrezaamiri, H. (2016)؛ «A novel artificial bee colony algorithm for shortest path problems with fuzzy arc weights», *Measurement*, *93*, 48-56.

Engelbrecht, A. P. (2007)؛ «Introduction to computational intelligence», *Computational Intelligence: An Introduction, Second Edition*, 1-13.

Gelado, I., Stone, J. E., Cabezas, J., Patel, S., Navarro, N., & Hwu, W. M. W. (2010)؛ «An asymmetric distributed shared memory model for heterogeneous parallel systems»., In *ACM SIGARCH Computer Architecture News* ,Vol. 38, No. 1, pp. 347-358.

Harman, M., McMinn, P., De Souza, J. T., & Yoo, S. (2012)؛ «Search based software engineering: Techniques, taxonomy, tutorial», In *Empirical software engineering and verification*, pp. 1-59.

Jamil, M., & Yang, X. S. (2013): «A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*», *4*(2), 150-194.

Karaboga, D., & Basturk, B. (2007): «A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm», *Journal of global optimization*, *39*(3), 459-471.

Laguna-Sánchez, G. A., Olguín-Carbajal, M., Cruz-Cortés, N., Barrón-Fernández, R., & Álvarez-Cedillo, J. A. (2009): «Comparative study of parallel variants for a particle swarm optimization algorithm implemented on a multithreading GPU», *Journal of applied research and technology*, *7*(3), 292-307.

Lee, T. Y., & Chen, C. L. (2007): «Iteration particle swarm optimization for contract capacities selection of time-of-use rates industrial customers», *Energy conversion and management*, *48*(4), 1120-1131.

Li, J. Z., Chen, W. N., Zhang, J., & Zhan, Z. H. (2015): «A Parallel Implementation of Multiobjective Particle Swarm Optimization Algorithm Based on Decomposition», In *Computational Intelligence, 2015 IEEE Symposium Series on* (pp. 1310-1317).

Moraes, A. D. O. S., da Cunha Lage, P. L., & Secchi, A. R. (2013): «ASYNCHRONOUS PARALLELIZATION OF THE PARTICLE SWARM OPTIMIZATION ALGORITHM AND ITS APPLICATION TO PARAMETER ESTIMATION IN A LARGE DIMENSIONAL SPACE», COBEM, Brazil, 3490-3501.

Mussi, L., Nashed, Y. S., & Cagnoni, S. (2011): «GPU-based asynchronous particle swarm optimization», In *Proceedings of the 13th annual conference on Genetic and evolutionary computation* , 1555-1562.

Rada-Vilela, J., Zhang, M., & Seah, W. (2011): «Random asynchronous PSO», In *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on* (pp. 220-225).

Schutte, J. F., Reinbolt, J. A., Fregly, B. J., Haftka, R. T., & George, A. D. (2004): «Parallel global optimization with the particle swarm algorithm», *International journal for numerical methods in engineering*, *61*(13), 2296.

Tian, N., & Lai, C. H. (2014): «Parallel quantum-behaved particle swarm optimization», *International Journal of Machine Learning and Cybernetics*», *5*(2), 309-318.

Yazdani, M., Amiri, M., & Zandieh, M. (2010): «Flexible job-shop scheduling with parallel variable neighborhood search algorithm», *Expert Systems with Applications*, *37*(1), 678-687.