مجله محاسبات نرم، جلد ۹، شماره ۱ (بهار و تابستان ۱۳۹۹)، صفحه ۵۶ـ۹۱
https://dx.doi.org/10.22052/scj.2021.111452

مجله محاسبات نرم
SOFT COMPUTING JOURNAL
تارنمای مجله: scj.kashanu.ac.ir

دانشگاه کاشان
University of Kashan

ارائه یک نسخه بهبودیافته از الگوریتم ژنتیک مبتنی بر راهکار خودسازماندهی بحرانی و حافظه گوسی برای حل مسائل بهینه‌سازی پویا✧

مجید محمدپور[۱*]، کارشناسی ارشد، بهروز مینایی بیدگلی[۲]، دانشیار، حمید پروین[۳]، استادیار، کیوان رحیمی‌زاده[۴]، استادیار

[۱] باشگاه پژوهشگران و نخبگان، دانشگاه آزاد اسلامی واحد یاسوج، کهگیلویه و بویراحمد، ایران.

[۲] دانشکده مهندسی کامپیوتر، دانشگاه علم و صنعت، تهران، ایران.

[۳] دانشکده مهندسی کامپیوتر، دانشگاه آزاد اسلامی، نورآباد ممسنی، فارس، ایران.

[۴] دانشکده فنی و مهندسی، گروه مهندسی برق و کامپیوتر، دانشگاه یاسوج، یاسوج، ایران.

اطلاعات مقاله

تاریخچه مقاله:
دریافت ۱۸ اردیبهشت ماه ۱۳۹۹
پذیرش ۱۷ بهمن ماه ۱۳۹۹

کلمات کلیدی:
بهینه‌سازی
الگوریتم ژنتیک
محیط پویا
حافظه
خوشه‌بندی
تخمین تراکم
جفت‌گزینی
جهش
خودسازمانده

چکیده

از آن‌جایی‌که اجزای پویا، همراه با محدویت‌های غیرخطی و اهداف متعدد، یکی از خصوصیت‌هایی است که به‌طور مکرر در مسائل دنیای واقعی ظاهر می‌شود و از طرف دیگر زمان زیادی است که محاسبات تکاملی وارد حوزه کاربردهای صنعتی شده است (به خصوص به‌علت توانایی آن‌ها در مواجهه با محیط‌های چندهدفه و غیرخطی) انتظار می‌رود که توجه به این زمینه در جامعه علمی رشد پیدا کند. هدف این مقاله، امکان طراحی پروتکل‌های الهام گرفته از طبیعت در الگوریتم ژنتیک است که روی بهینه‌سازی در محیط‌های پویا موثر باشد، در حالی‌که پیچیدگی الگوریتم را حفظ کند و تغییرات در فضای مسئله به‌صورت دوره‌ای رخ دهد. در این مقاله، یک الگوریتم ژنتیک بهبودیافته (خودسازمانده بحرانی) مبتنی بر حافظه برای حل مسائل بهینه‌سازی پویا ارائه شده است. در الگوریتم ارائه‌شده، از یک عملگر جهش خودسازمانده استفاده شده است. این عملگر جهش می‌تواند نرخ‌های جهش خودتنظیم‌شونده را با یک توزیع ویژه بر مبنای مدل تپه شنی انجام دهد که این برای بهینه‌سازی پویا مناسب است. اگر تغییرات به‌صورت دوره‌ای رخ دهند، به‌طور معمول استفاده از اطلاعات گذشته اجازه می‌دهد الگوریتم به‌سرعت بعد از تغییر محیط به سازگاری در شرایط محیطی جدید برسد. ایده مورد نظر در این زمینه، استفاده از یک حافظه می‌باشد. یکی از چالش‌های اساسی در به‌کارگیری حافظه، تنوع می‌باشد. برای افزایش سطح تنوع از یک حافظه تخمین تراکم با خوشه‌بندی گاوسی استفاده شده است. همچنین از راهکاری برای جایگزینی و بازیابی در حافظه استفاده شده است. در طرح پیشنهادی ابتدا جهش خودسازمانده بحرانی جدید، با سایر الگوریتم‌های ژنتیک ارائه شده توسط سایر محققین ترکیب شده و نتایج حاصل شده نشان می‌دهد که این روش توانسته به‌کرات سایر الگوریتم‌های ژنتیک را برای محیط‌های پویا بهبود بخشد. در نهایت روش پیشنهادی این مقاله که ترکیب خودسازماندهی بحرانی جدید با حافظه تخمین تراکم گوسی است ارائه شده است. نتایج این روش بر روی مسائل محک مختلف با عنوان توابع تله پویا (Royal road، One Max و Deceptive)، آزمایش شده ونتایج حاصله حاکی از کارایی بیشتر روش پیشنهادی است.

نحوه ارجاع به مقاله: محمدپور، مجید، مینایی بیدگلی، بهروز، پروین، حمید، رحیمی‌زاده، کیوان، «ارائه یک نسخه بهبودیافته از الگوریتم ژنتیک مبتنی بر راهکار خودسازماندهی بحرانی و حافظه گوسی برای حل مسائل بهینه‌سازی پویا»، مجله محاسبات نرم، جلد ۹، شماره ۱، ص۵۶ـ۹۱، بهار و تابستان ۱۳۹۹.

۱. مقدمه

در سال‌های اخیر بهینه‌سازی پویا (یعنی، بهینه‌سازی توابع ناپایدار) یکی از موضوعات اصلی تحقیقات محاسبات تکاملی شده است [۱-۳]. از آن‌جایی‌که سیستم‌های پویا، همراه با قیود غیرخطی و اهداف گوناگون، یکی از ویژگی‌هایی است که به کرات در مسائل جهان واقعی پدیدار می‌شود و به‌دلیل این‌که مدت زمان زیادی است که بهینه‌سازی تکاملی وارد حوزه کاربردهای صنعتی شده است، انتظار می‌رود که هر چه زودتر توجه به این موضوع در جوامع علمی رشد پیدا کند. علاقه-مندی در این موضوع به هر حال موضوع جدیدی نیست و مطالعات و تحقیقات بسیاری روی سیستم‌های تکاملی برای اهداف بهینه‌سازی منتشر شده است [۱-۳].

برای حل مسائل بهینه‌سازی پویا نیاز به کاوش و ذهن قوی می‌باشد. در محیط‌های پویا که با زمان دچار تغییر می‌شوند چهار چالش اساسی و مهمی پیش روی یک الگوریتم بهینه-سازی وجود دارند. آن چهار دسته شامل، واکنش به تغییرات (فراجهش)، نگهداری تنوع (مهاجران تصادفی)، طرح‌های مبتنی بر حافظه و روش‌های چندجمعیتی است [۱]. برای حل مسائل بهینه‌سازی پویا روش‌های مختلفی ارائه شده است که هر کدام از این روش‌ها به نوعی یک یا چند چالش موجود برای محیط‌های پویا را برطرف می‌نمایند. امروزه تحقیقات در حال گسترش به سمت حفظ تنوع و طرح‌های مبتنی بر حافظه رهنمود شده است. این احتمالا به خاطر این است که بسیاری از طرح‌های مبتنی بر چندجمعیتی ممکن است در یکی از دسته‌های باقیمانده قرار گیرند (و در تعدادی، متمایز کردن آنها از طرح‌های مبتنی بر حافظه واقعا دشوار است)، همچنین طرح‌های تکاملی که به تغییرات واکنش نشان می‌دهند، فقط وقتی که تشخیص آن تغییرات آسان باشند (و تنها همان تغییرات در محیط موجود باشند)، می‌توانند استفاده شوند. علاوه‌بر این کارایی آن‌ها، به شدت تغییرات وابسته است. به اضافه این‌که طرح‌های چندجمعیتی محض، معمولا نیاز به روش‌های به‌روزرسانی و مهاجرت پیچیده‌ای دارند که باعث می‌شود تنظیم و پیاده‌سازی الگوریتم مشکل شود [۲].

هدف این مقاله و ادعای اصلی آن، امکان طراحی پروتکل‌های الهام‌گرفته از طبیعت برای استفاده در الگوریتم ژنتیک است که در مسائل بهینه‌سازی پویا موثر باشند و پیچیدگی الگوریتم را حفظ کند. در این مقاله فرض شده است که، تغییرات به صورت دوره‌ای رخ داده و الگوریتم به اطلاعات قبلی مساله برای استفاده در آینده نیاز داشته باشد. این تحقیق عمدتا روی تکنیک‌های حفظ تنوع تمرکز دارد، بنابراین چالش اساسی این است که الگوریتم باید با یک محیط در حال تغییر درگیر باشد و بنا به دوره‌ای رخ دادن تغییرات، بتواند از اطلاعات گذشته برای استفاده در آینده بهره ببرد. بدین منظور در این مقاله از نوعی حافظه صریح برای حفظ راه‌حل‌های گذشته استفاده شده است. در سال‌های اخیر، سیستم‌های خودسازمانده بحرانی، به‌عنوان مهم‌ترین نوع از سیستم‌های خودسازمانده، توجه ویژه‌ای را به‌خود جلب کرده‌اند. سیستم‌های خودسازمانده بحرانی، از تعدادی مولفه خودمختار تشکیل شده‌اند و راهکار کنترلی در میان این مولفه‌ها توزیع شده است. هر یک از مولفه‌های سیستم سعی می‌کنند با تکیه بر اطلاعات و سیاست‌های محلی و همچنین تعامل با دیگر مولفه‌ها، به اهداف محلی خود دست یابند. در فیزیک نقطه بحرانی نقطه‌ای است که در آن تغییر اساسی در رفتار یا ساختار یک سیستم به وجود می‌آید. تحقیقات گذشته محققین نشان می‌دهد که سیستم‌های خودسازمانده برای حل مسائل بهینه‌سازی پویا می‌توانند بسیار مناسب باشند [۱، ۳، ۲۲، ۲۵، ۳۵، ۴۰]. ولی در تحقیقات گذشته، تا به حال ترکیب یک الگوریتم مبتنی بر حافظه با یک روش خودسازمانده بحرانی استفاده نشده است. معمولا این دو روش (مدل حافظه صریح و مدل خودسازمانده) را محققین به صورت دو مدل کاملا متفاوت از هم می‌دانند. ولی همه آن‌ها بر این عقیده هستند که

✧ نوع مقاله: پژوهشی
* نویسنده مسئول
پست(های) الکترونیک: majid.mohammadpour2@gmail.com (محمدپور)
b_minaei@iust.ac.ir (مینایی بیدگلی)
parvin@iust.ac.ir (پروین)
rahimizadeh@yu.ac.ir (رحیمی‌زاده)

این دو مدل می‌توانند برای حل مسائل بهینه‌سازی پویا موثر واقع شوند [۱، ۳، ۲۲، ۲۵، ۳۵، ۴۰].

در این مقاله، الگوریتمی مبتنی بر حافظه و مدل خودسازمانده بحرانی ارائه خواهد شد که این الگوریتم از مزایای هر دو روش برای حل مسائل بهینه‌سازی پویا بهره می‌برد. طرح پیشنهادی روی طیف گسترده‌ای از مسائل بهینه‌سازی پویا آزمایش شده است و قادر است به کرات دیگر الگوریتم‌های ژنتیک را بهبود ببخشد. اثبات شده است که روش پیشنهادی این مقاله در حل مسائل بهینه‌سازی پویا موثر است، زیرا با روش‌های مشابه دیگر در پارامترهایی از محیط‌های پویا مقایسه شده که این روش‌ها در این پارامترها، بهترین کارایی را دارا هستند. نوآوری اساسی روش ارائه شده در این مقاله، ترکیب حافظه تخمین تراکم گوسی و جهش خودسازمانده بحرانی با الگوریتم ژنتیک برای حل مسائل مختلف بهینه‌سازی پویا می‌باشد. در نسخه قبلی ارائه شده در مرجع [۴۳] فقط از نرخ جهش خودسازمانده در الگوریتم ژنتیک استفاده شده بود که دارای ایرادتی است. در این مقاله سعی شده تا با ترکیب حافظه تخمین تراکم گوسی در کنار روش [۴۳] این معایب از بین برده شوند. در زمینه مسائل بهینه‌سازی پویا کارهای زیادی انجام گرفته است که هر کدام به نوعی چالش‌های پیش روی این مسائل را مورد بررسی قرار می‌دهند. برخی از معروف‌ترین روش‌های ارائه شده برای مسائل بهینه‌سازی در محیط‌های پویا در ادامه در بخش ۲ آورده شده است.

این مقاله بدین صورت سازماندهی شده است: در بخش اول مقاله، مقدمه و کلیات موضوع تشریح شده است. در بخش دوم، ادبیات تحقیق و کارهای گذشته مشابه با روش پیشنهادی مفصلا توضیح داده شده است. بخش سوم این مقاله، روش پیشنهادی را تشریح نموده و در فصل چهارم آزمایش‌های مختلفی بر روی روش پیشنهادی صورت گرفته و نتایج تجربی آن‌ها آورده شده است. در نهایت فصل پنجم، به نتیجه‌گیری کلی و پیشنهادات آتی می‌پردازد.

۲. ادبیات تحقیق و مرور منابع

در این بخش ابتدا برجسته‌ترین تکنیک‌های تکاملی استفاده شده در محیط‌های پویا همراه با کاربرد آن‌ها شرح داده می‌شود. به‌علاوه در قسمت مرور منابع از همین بخش، الگوریتم‌هایی جهت بهینه‌سازی پویا شرح داده می‌شوند و عملکرد آن‌ها بررسی می‌شود. همچنین در این بخش، دسته‌بندی محیط‌های نامطمئن (نوعی محیط پویا هستند) و الگوریتم‌های الهام گرفته‌شده زیستی برای بهینه‌سازی پویا دنبال شده است. در ادامه مولد مسائل پویا که در [۱] ارائه شده و در سرتاسر این تحقیق از آن استفاده شده است، توصیف می‌شود.

۲،۱. الگوریتم‌های تکاملی

الگوریتم‌های تکاملی به‌طور موفق در محدوده وسیعی از برنامه‌های کاربردی استفاده شده‌اند و برای حل مسائل بهینه‌سازی مناسب می‌باشند. اکثر برنامه‌های دنیای واقعی پویا هستند و الگوریتم‌های مورد استفاده برای حل آنها باید بتوانند با شرایط جدید سازگار شوند. برای این نوع بهینه‌سازی، یک الگوریتم تکاملی موثر باید بتواند تغییرات را شناسایی کند و به سرعت تغییرات را دنبال نماید.

برخی از الگوریتم‌ها برای این گونه مسائل مناسب نیستند، زیرا تمایل دارند پیش از موعد به یک راه‌حل همگرا شوند و هنگامی که شرایط محیط تغییر می‌کند، جمعیت همه افراد معمولا در یک نقطه خاص از فضای جستجو متمرکز می‌شوند. بنابراین، زمان زیادی برای تطبیق جمعیت و دستیابی به راه‌حل جدید صرف می‌شود. همچنین برخی از الگوریتم‌های دیگر دارای جستجوی سراسری ضعیف می‌باشند. برای مقابله با این محدودیت‌ها می‌بایست بهبودهایی بر روی الگوریتم‌های تکاملی انجام پذیرد که شامل حفظ تنوع، استفاده از چندین استراتژی مانند فوق‌جهش، مهاجران تصادفی، عملگرهای ژنتیکی، استفاده از طرح‌های حافظه، استفاده از چند جمعیت و پیش‌بینی تغییرات در محیط باشد.

الگوریتم‌های تکاملی شامل تکنیک‌های بهینه‌سازی مختلفی می‌باشند که از انتخاب طبیعی و تکامل زیست‌شناسی الهام گرفته شده‌اند. معمولا، الگوریتم‌های تکاملی جستجو را بر پایه جمعیت با استفاده از عملگرهایی از جمله تزویج و جهش انجام می‌دهند. جستجو بر اساس جمعیت با حافظه بسیار مناسب است، به‌طوری که عناصر متعددی از حافظه می‌توانند درون فرآیند جستجو به‌کار روند. شبه‌کد الگوریتم ژنتیک استاندارد در شکل (۱) آورده شده است.

Algorithm 1: Standard Genetic Algorithm (SGA)

```
init(POP) //initialize the population
eval(POP)
WHILE (termination criteria not fulfilled)
  POP = POP ∪ M
  S_POP = select(POP)
  S'_POP = crossover(S_POP)
  S''_pop = mutation(S'_POP)
  POP = S''_POP   //update population
  eval(POP)   //evaluate individuals in the population
```

شکل (۱): شبه‌کد الگوریتم ژنتیک استاندارد (SGA)

الگوریتم‌های تکاملی بر اساس یک مدل ساده از نظریه تکامل با انتخاب طبیعی شکل گرفته‌اند. برای حل یک مساله خاص، مجموعه‌ای از راه‌حل‌های کاندید مربوط به مساله به‌صورت تصادفی ایجاد می‌شوند، افراد انتخاب شده با استفاده از عملگرهای ژنتیکی تکثیر پیدا می‌کنند، به‌عنوان مثال، عملگر ترکیب مجدد و جهش، یک جمعیت جدید فرزند را تولید می‌کنند، در نهایت نسل بعدی جمعیت با ترکیب جمعیت والدین و فرزندان شکل می‌گیرد، این فرآیند تا زمانی که شرایط توقف خاصی، به‌عنوان مثال، حاصل شدن تعداد نسل خاص، تکرار می‌شود. الگوریتم‌های تکاملی به‌عنوان روش‌های مبتنی بر جستجوی تصادفی، به‌طور موفق برای حل مسائل سخت و پیچیده مورد استفاده قرار گرفته‌اند، یعنی مسائلی که دارای هیچ راه‌حل تحلیلی نیستند و یا ابعاد فضای جستجو بسیار وسیع است و باعث می‌شود مساله با استفاده از تکنیک‌های رایج و متعارف قابل حل نباشد [۱-۳]. در الگوریتم‌های تکاملی مواردی می‌بایست در نظر گرفته شوند که عبارتند از:

جمعیت[1]: شامل مجموعه‌ای از افراد است که معمولا کروموزوم نامیده می‌شوند و اندازه ثابتی دارند، هر فرد در جمعیت یک راه‌حل ممکن برای مسئله است و شامل یک توالی از اجزای کوچکتر است که ژن‌ها[2] نامیده می‌شوند و هر ژن ممکن است شامل مقادیر مختلفی باشد. در ابتدای فرآیند تکاملی، جمعیت اولیه به‌صورت تصادفی یا با استفاده از یک روش مشخص تولید می‌شود. این جمعیت تکامل یافته و در انتهای فرآیند جستجو، جمعیت به یک منطقه از فضای جستجو همگرا می‌شود که شامل یک راه‌حل بهینه یا شبه بهینه است.

نمایش: به ساختار افراد بر اساس نوع مساله‌ای که باید حل شود نمایش[3] گفته می‌شود، نمایش تعریف می‌کند که چگونه افراد جمعیت کدگذاری شوند. بر اساس نوع مساله و نوع الگوریتم تکاملی، انواع مختلفی از نمایش به‌عنوان مثال باینری، جایگذاری عددی، درختی، نموداری و غیره می‌تواند مورد استفاده قرار گیرد.

تابع برازش[4]: برای اندازه‌گیری کیفیت افراد جمعیت استفاده می‌شود. برازش یک مقدار واقعی است که با استفاده از تابع برازش به‌دست می‌آید، بر اساس نوع مساله، بهترین برازش می‌تواند بیشترین (مساله بیشینه‌سازی) یا کمترین (مساله کمینه‌سازی) مقدار باشد.

روش انتخاب[5]: برای انتخاب یک مجموعه از والدین بر اساس برازش آنها می‌باشد، به‌طوری که راه‌حل‌هایی با مقدار برازش بالاتر احتمال بیشتری برای انتخاب دارند، به این ترتیب، یک جمعیت مناسب‌تر ایجاد می‌شود و الگوریتم تکاملی به سمت راه‌حل بهینه تکامل می‌یابد، روش‌های مختلفی برای انتخاب وجود دارند که می‌توانند مورد استفاده قرار گیرند، مانند چرخ رولت[6]، انتخاب مبتنی بر مسابقات[7] یا

[1] Population
[2] Jene
[3] Presentaion
[4] Fitness
[5] Selection
[6] Roulette Wheel
[7] Tournament-based

رتبه‌بندی[1]. روش چرخش رولت افراد را بر اساس برازش نسبی آنها انتخاب می‌کند، افراد با برازش بالاتر احتمال بیشتری جهت انتخاب دارند. در انتخاب بر اساس مسابقات، تعداد مشخصی (اندازه تورنمنت) از افراد به‌صورت تصادفی انتخاب شده و بهترین افراد مورد استفاده قرار می‌گیرند. در انتخاب بر اساس رتبه‌بندی، افراد با توجه به برازش خود، مرتب می‌شوند و هر فرد احتمال معینی جهت انتخاب دارد که بسته به موقعیت آن در رتبه‌بندی است.

عملگرهای ژنتیک: نقش عملگرهای ژنتیکی[2]، ایجاد تغییرپذیری در میان افراد جمعیت است. عملگرهای ژنتیکی را می‌توان به دو دسته اصلی ترکیب مجدد (یا تزویج) و جهش تقسیم کرد. ترکیب مجدد با استفاده از دو (یا بیشتر) والد انتخاب شده صورت می‌گیرد و محتوای ژنتیکی آنها ترکیب می‌شوند. همچنین در جهش مقدار ژن‌ها تغییری کوچکی می‌کنند.

جمعیت جدید[3]: شامل افراد جدیدی است که از جمعیت قبل یا والدین حاصل شده است. در حقیقت، دو روش اصلی برای تولید جمعیت نسل بعدی وجود دارد. یک روش این است که همه والدین را حذف نماییم و تمام فرزندان را نگه داریم که این کار، رویکرد نسلی[4] نامیده می‌شود، روش دیگری به این صورت است که دو جمعیت را ادغام کرده و بهترین‌ها را برای انتقال به جمعیت بعدی انتخاب کنیم، در یک روش دیگر یک کسر کوچک از بهترین والدین را نگه می‌داریم و بقیه را از جمعیت بهترین فرزندان انتخاب می‌کنیم، که این رویکرد، نخبه‌گرایی[5] نامیده می‌شود.

۲،۲. بهینه‌سازی پویا

بهینه‌سازی پویا شامل یک سطح پیچیده‌تری از مواردی است که در بهینه‌سازی ایستا معرفی شده است. یک مساله زمانی پویا است که یک تغییر در تابع کارایی، محدودیت‌ها یا نمونه مساله وجود داشته باشد، بنابراین بایستی تغییرات بهینه به خوبی ساخته شوند. در هر دوره بهینه‌سازی تابع کارایی مقطعی است، اما تا زمانی‌که تغییراتی رخ می‌دهد، راه‌حل‌هایی که قبلا پیدا شده بودند ممکن است خیلی متغیر نباشند و این باعث می‌شود که یک جستجوی جدید حاصل شود. هنگام استفاده از الگوریتم‌های تکاملی و دیگر الگوریتم‌های الهام گرفته زیستی جهت مقابله با این نوع از مسائل، نیاز به یک تعادل خاص بین شناسایی و استخراج در محیط‌های ایستا است [۶]. اگر الگوریتم به‌طور کامل همگرا شود، مشکل است که دوباره خودش را با تغییرات در آن محیط وفق دهد و عکس‌العملی نشان دهد. جستجوی سراسری می‌بایست برخی اوقات حتی در سرعت همگرایی افزایشی باشد، البته [۵] اظهار داشتند:

ساده‌ترین راه برای واکنش به تغییرات در محیط این است که هر تغییر به عنوان یک ورودی مسئله بهینه‌سازی جدید در نظر گرفته شود که بایستی مجددا حل شود. با توجه به زمان کافی این کار مطمئنا یک جایگزین مناسب است، با این حال در برخی از حالت‌ها زمان کافی وجود ندارد، در حالی که مشکلات دیگری در زمانی که تغییرات در محیط به‌سادگی قابل تشخیص نیستند به‌وجود می‌آید و واکنش به تغییرات یکسان سخت است. به‌علاوه، حتی اگر آن تغییرات قابل تشخیص باشند، بعد از یک تغییر در محیط، تصمیم‌گیری سخت است که آیا آن جمعیت دوباره شروع مجدد شود یا جستجو با همان جمعیت ادامه پیدا کند. این تصمیم ممکن است به شدت تغییرات وابسته باشد یا حتی پویایی آن‌ها توسط سرعت شدت (سختی)، دوره تناوب و دیگر صفات تعیین شود [۳].

دشواری بهینه‌سازی پویا می‌تواند بر اساس یافتن توازنی مناسب بین دو ویژگی متضاد از روال جستجو بیان شود، مانند موازنه‌ای که بین کاوش و بهره‌برداری در الگوریتم‌های تکاملی وجود دارد. در طول چند سال گذشته تعدادی از نویسندگان زمانی که مسائل پویا از طریق الگوریتم‌های تکاملی و الهام گرفته‌شده طبیعی حل می‌شده‌اند، مسائل همگرایی و بعد از آن

[1] Ranked-based
[2] Genetic Operators
[3] New Population
[4] Generational
[5] Elitism

عدم تطبیق‌پذیری را عنوان کرده‌اند، این روش‌ها که توسط برانک[1] [۳] معین شده است در ۴ طبقه، دسته‌بندی می‌شوند:

الف. واکنش به تغییرات[2]

فراجهش‌ها [۷] روش‌هایی هستند که در این طبقه‌بندی قرار دارند. فراجهش با محیط‌های متغیر زمانی که یک تغییر کشف شود از طریق افزایش چشم‌گیر احتمال جهش سروکار دارد. مقادیر احتمال از ۰/۰۰۱ تا ۰/۵ متغیر هستند. افزایش احتمال جهش به‌وسیله یک دوره از تنزل دنبال می‌شود که نرخ آن به مقادیر پایه‌اش تنزل پیدا می‌کند. اخیرا نویسندگان مرجع [۷] نشان داده‌اند که مقادیر فراجهش بزرگ، زمانی که تغییرات محیطی به‌صورت مکرر اتفاق می‌افتد بهینه بهتری را دنبال می‌کنند در حالی‌که فراجهش‌های کوچک‌تر سطوح عملکردشان در زمانی‌که تغییرات کمتر هستند بهتر است. ندانستن بهترین مقدار تابع کارایی ممکن است کارآیی و حتی زیست‌پذیری را کاهش دهد. ایده‌های مشابهی در [۸–۱۰] پیشنهاد شده است. به طورکلی این روش‌های ساده در مقایسه با استراتژی‌های دیگر ناکارآمد هستند برای این که حل یک مساله بدون استفاده مجدد از اطلاعات گذشته ممکن است وقت‌گیر باشد و یک تغییر ممکن نیست به‌طور مستقیم شناسایی شود، یا این‌که ممکن است راه‌حل مساله جدید نسبت به راه‌حل‌های مساله قدیمی خیلی تفاوت نداشته باشند. بنابراین بعضی اوقات بهتر است الگوریتمی داشته باشیم که قادر باشد به‌طور پیوسته راه‌حل را با یک محیط در حال تغییر تطبیق دهد و دوباره از اطلاعات به‌دست آمده در گذشته استفاده نماید.

ب. طرح‌های حافظه

نوع دیگری از این روش‌ها الگوریتم‌های با حافظه[3] هستند که راه‌حل‌ها را ذخیره می‌کنند برای این که بعدا مجدد استفاده شوند. این کار در مواردی که محیط به‌صورت دوره‌ای تغییر می‌کند و وضعیت‌های تکراری رخ دهد، می‌تواند مفید باشد. همچنین از سویی دیگر اگر تغییرات محیط جدید باشند، می‌تواند زیان‌بخش باشد. حال حافظه می‌تواند به دو روش کلی ارائه شود: به‌طور ضمنی[4]، که از نمایش‌های افزونه استفاده می‌کند یا به‌طور صریح[5]، که شامل یک حافظه اضافی و همچنین استراتژی‌هایی برای ذخیره و بازیابی راه‌حل‌ها است [۳]. روش اصلی برای حافظه‌های ضمنی و نمایشی افزونه روش‌های موسوم به روش‌های دیپلویدی[6] می‌باشند. که برای اولین‌بار در بهینه‌سازی پویا به‌وسیله نویسندگان [۱۱] استفاده شده است، مثال‌های دیگری در [۱۲–۱۵] ارائه شده است. الگوریتم ژنتیک دیپلویدی شامل مجموعه‌ای از دو کروموزوم است (به‌جای تنها یک کروموزوم)، و بیشتر از یک ژن جهت رفتار فنوتایپ در آن افراد یکسان رقابت می‌کنند. یک نقشه غالب برای برچسب برخی از ژن‌ها به‌عنوان غالب و مغلوب استفاده می‌شود و اگر یک ژن غالب با یک ژن مغلوب جفت شوند تنها آن قبلی در فنوتایپ بیان می‌شود و ژن مغلوب را ترک می‌کند [۱۶]. در نتیجه ژن غالب ممکن است کمتر از ژن‌های مغلوب حفظ شود. بعد از تغییرات تابع، ژن‌هایی که خیلی مناسب نیستند ممکن است اگر محیط مساعد باشد به‌داخل بیان ژن آورده شوند. برای این ساختار، الگوریتم ژنتیک دیپلوید[7] به نقشه غالب نیاز دارد. یکی از بیشترین نقشه‌هایی که به آن ارجاع شده به‌وسیله مرجع [۱۷] پیشنهاد شده است، اما برخی نویسندگان، روش‌های جایگزین دیگری را ارائه دادند [۱۲–۱۵]. در [۱۸] یک پایگاه دانش برای ذخیره افراد موفق در حافظه پایدار استفاده شده است. با فرض این‌که سیستم می‌تواند وضعیت‌های محیطی را اندازه‌گیری کند. این طرح‌ها، حافظه مبتنی بر مورد[8] نامیده می‌شوند و به‌عنوان یک نوع از نخبه‌گرایی طولانی مدت استفاده می‌شود. بهترین راه‌حل‌هایی که در هر نسل ایجاد می‌شوند در یک منبع خارجی ذخیره می‌شوند و می‌توانند بعدا به‌وسیله افرادی که مجددا از حافظه به جمعیت معرفی می‌شوند کشف و استخراج شوند یا به‌وسیله معرفی مجدد، زمانی‌که تابع کارایی بالایی دارند استفاده شوند. این الگوریتم‌ها، اگرچه ممکن است در برخی از

[1] Branke
[2] Reaction to changes
[3] Memory schemes

[4] Implicitly
[5] Explicitly
[6] Diploidy
[7] Diplodity
[8] Case-based memory

انواع مسائل پویا کارا باشند، اما آن‌ها وقتی مسائل پویای کلی‌تر در نظر گرفته می‌شوند، با مشکلاتی مواجه می‌شوند. به‌عنوان مثال اگر تغییرات قابل تشخیص نباشند تصمیم‌گیری سخت خواهد بود و همچنین زمانی‌که افراد دوباره به جمعیت معرفی می‌شوند بایستی توسط تابع کارایی راه‌حل‌ها، مجدد به‌روز شوند و این‌کار نیاز به ارزیابی زیادی خواهد داشت. حال اگر تغییرات قابل تشخیص باشند حافظه می‌بایست بعد از هر تغییر دوباره ارزیابی شود. اگر فرکانس تغییرات بالا باشد محاسبات کاملا مرتبط خواهد بود و می‌بایست تعدادی تصمیم‌گیری انجام شود و استراتژی‌هایی برای بازیابی آن راه‌حل‌ها انتخاب شود و این عملکرد الگوریتم‌ها را تحت تاثیر قرار می‌دهد [۱۸]. مثال‌های مختلفی از شرح فوق به‌وسیله [۱۹] ارائه شده است که در آن هر فرد با حافظه اضافی برای تعدادی از اجدادش و یا به‌وسیله یک تغییر از نخبه‌گرایی تکاملی در الگوریتم‌های ژنتیکی گسترش می‌یابد، در هر نسل بهترین فرد در حافظه ذخیره می‌شود و فرد دیگری بر اساس سن از حافظه حذف می‌شود. برانک [۳] تعدادی از استراتژی‌های جایگزینی برای درج افراد جدید داخل یک حافظه را مقایسه کرد و تاکید کرد که برای روش‌های مبتنی بر حافظه نیز تنوع اهمیت فراوانی دارد، به عنوان مثال برانک پی برد که یک جایگزینی ساده که بر اساس شبیه‌ترین فرد انجام گیرد تقریبا معادل با یک استراتژی است که دو عدد از بدترین افراد را در حافظه نزدیک به یکدیگر جایگزین می‌کند. دیگر محققان، حافظه را با استراتژی‌های دیگری که برای بهینه‌سازی پویا طراحی شده نیز ترکیب کردند. در مقاله [۲۰] دو راه‌حل برای ترکیب با الگوریتم SSPCO ارائه شده است که عبارتند از، روش چندجمعیتی و حافظه با تخمین تراکم گوسی. راهکار چندجمعیتی، می‌تواند جمعیت را در فضای مساله به چندین زیرجمعیت تقسیم نماید. چند جمعیت به‌جای یک جمعیت تنها، می‌تواند تنوع را در فضای مسئله افزایش داده و در نهایت همگرایی ذرات جمعیت به بهینه (یا بهینه‌ها) سریع‌تر صورت می‌گیرد. محمدپور و پروین در مرجع [۲۱] یک الگوریتم ژنتیک آشوب‌گونه مبتنی بر خوشه‌بندی و حافظه برای حل مسائل پویا ارائه نموده‌اند. آن‌ها در این مقاله اثبات نموده‌اند یک سیستم آشوب‌گونه پیش‌بینی دقیق‌تری از آینده نسبت به یک سیستم تصادفی دارد و میزان هم‌گرایی را در الگوریتم افزایش می‌دهد. آنها در این مقاله روشی ارائه نموده‌اند که خوشه‌بندی را هم در حافظه و هم در جمعیت اصلی انجام می‌دهد. این کار باعث شده تا تنوع در حین اجرای الگوریتم با تبادل اطلاعات میان خوشه‌های متناظر (خوشه‌ها با برچسب شبیه به‌هم) در حافظه و جمعیت اصلی حفظ شود. آن‌ها در این مقاله توانسته‌اند با به کارگیری حافظه، نتایج قابل قبولی به‌دست آورند [۲۱]. نویسندگان مقاله [۲۱] در سه مقاله دیگر، که در مراجع [۲۲-۲۴] آورده شده‌اند، توانسته‌اند با استفاده از حافظه نتایج قابل قبول و مناسبی برای مواجهه با مسائل بهینه-سازی پویا به‌دست آورند. به‌طور کلی، حافظه زمانی مفید است که پویایی به‌صورت چرخشی تغییر می‌کند و شکل چشم‌انداز تابع کارایی لحظه به لحظه تکرار می‌شود. اگر تغییرات بدون تناوب صورت گیرد، طرح‌های حافظه برخی از اثرات‌شان را از دست می‌دهند. البته باید این موضوع را بیان نمود که، همان-طور که در [۲۱-۲۴] به اثبات رسیده است، اگر طرح‌های مبتنی بر حافظه را بتوان با روش‌های مناسبی همانند خوشه-بندی، آشوب و روش‌های چندجمعیتی ترکیب نمود، می‌توان راه‌حل‌های بسیار خوبی برای حل مسائل بهینه‌سازی پویا به دست آورد. به این دلایل، طرح‌های حافظه، مانند الگوریتم-هایی که برای غلبه بر محیط‌های پویا به تغییرات واکنش نشان می‌دهند راهکارهای موثری برای این نوع مسائل که هدف این مقاله است، می‌باشند. در طیف گسترده‌ای سناریوهای مربوط به پویایی محیط، تغییرات به‌آسانی قابل کشف نیستند و هدف، طراحی الگوریتم‌های قوی است که پیچیدگی را در فضای پارامترهایش افزایش ندهد.

ج. روش‌های چندجمعیتی[1]

این روش‌ها جمعیت را به چندین زیرجمعیت تقسیم می‌کنند که انتظار می‌رود چندین قله را در چشم‌انداز تابع کارایی دنبال کنند؛ یک بخش از جمعیت بهترین راه‌حل‌ها را دنبال

[1] Multi-population approaches

می‌کند درصورتی‌که بخش‌های دیگر جمعیت راه‌حل‌های زیربهینه را جستجو می‌کنند. در واقع، الگوریتم‌های چندجمعیتی برای بهینه‌سازی پویا یک نوع از روش‌هایی هستند که می‌توانند در افزایش تنوع در یک محیط پویا موثر باشند [۳]. برانک و همکاران [۳] برای غلبه بر مساله بهینه-سازی در محیط‌های پویا روش پیش‌آهنگ‌های خودسازمانده[1] را پیشنهاد دادند. ایده اصلی الگوریتم این است که وقتی یک قله پیدا شد (به‌عنوان مثال جمعیت به یک ناحیه کارایی بالا همگرا شد)، جمعیت می‌بایست تقسیم شود این در حالی است که جمعیت فرزندان می‌بایست جستجو را برای قله‌های جدید گسترش دهند. یک روش مشابه، الگوریتم ژنتیک چندملیتی[2] در مرجع [۲۵] ارائه شده است که از چند مدل بهینه‌سازی برای دنبال کردن بهینه متحرک استفاده می‌کند. یک ملت به عنوان یک جمعیت و دولت (بهترین راه‌حل در آن جمعیت) و سیاست (قله‌ای که جمعیت نزدیک آن شده) معین می‌شود. برای این‌که جمعیت مختلف در میان قله‌های مختلف چشم-انداز تابع کارایی گسترش یابد، یک روش تشخیص تپه-دره[3] و یک سیاست مهاجرت استفاده شده است. علاوه‌براین، مقادیر احتمالی جهش‌های مختلف بر اساس فاصله افراد با قله‌ای که ملت به آن تعلق دارد به‌کار برده می‌شود. همچنین انتخاب نسبت به الگوریتم‌های تکاملی سنتی متفاوت‌تر است. اگرچه آزمایش‌ها در [۲۵] نشان داد که الگوریتم ژنتیک چندملیتی کارآیی خوبی را از خود نشان داده است. اما، الگوریتم ژنتیک چندملیتی خیلی پیچیده است و نیاز به تنظیم بالایی دارد. اخیرا نویسندگان مراجع [۲۶-۲۸]، الگوریتم ژنتیک دوجمعیتی[4] را پیشنهاد داده‌اند. این الگوریتم از یک جمعیت اضافی استفاده می‌کند که جمعیت رزرو نامیده می‌شود، که تنوع اضافی را فراهم می‌کند. نتایج اولیه نشان داد که کارایی الگوریتم‌ها خیلی حساس به مسافت بین آن دو جمعیت می‌باشد و خیلی سخت است که بهترین مسافت بدون دانش قبلی در رابطه با آن مساله تعیین شود. برای غلبه بر این نوع مشکلات، یک جمعیت رزرو دوم به آن طرح اولیه اضافه می‌شود [۲۸-۳۱].

د. حفظ تنوع[5]

یک راه کلی برای پیگیری راه‌حل‌های متغیر نیز حفظ تنوع در سرتاسر اجراست. طرح‌هایی که به‌طور مداوم تنوع را معرفی می‌کنند یا به‌نحوی کمبودشان را به تاخیر می‌اندازند ممکن است سرعت همگرایی را کاهش دهند، اما برای بهینه‌سازی پویا بعضی اوقات مهم‌تر این است که فقط نزدیک بودن به بهینه را حفظ کنند به‌جای این‌که به‌طور کامل به آن همگرا شوند. در یک محیط پویا یک الگوریتم باید از همگرایی کامل تمام وقت اجتناب کرده و امید است که یک جمعیت گسترده می‌تواند به‌راحتی با تغییرات محیطی وفق پیدا کند. این الگوریتم‌ها عمومی‌تر هستند و ممکن است برای یک طیف گسترده از راه‌حل‌ها به کار برده شوند، بدون این‌که وابستگی داشته باشند. یک نمونه شناخته‌شده از تکنیک‌های حفظ تنوع که قدمتش به سال ۱۹۹۲ برمی‌گردد: الگوریتم ژنتیک مهاجران تصادفی (RIGA) [۱] است. الگوریتم RIGA از طریق معرفی راه‌حل‌های تصادفی r_p در جمعیت هر نسل تنوع را حفظ می‌کند و تضمین می‌کند که ماده ژنتیک جدیدی در هر مرحله زمانی وارد جمعیت می‌شود، بنابراین از همگرایی کل جمعیت برای یک ناحیه کوچکی از فضای جستجو جلوگیری می‌کند.

ه. روش‌های مبتنی بر یادگیری تقویتی[6]

گونه دیگری از الگوریتم‌ها وجود دارند که مبتنی بر یادگیری تقویتی هستند (مانند اتوماتای یادگیر) [۳۲-۳۴]. یادگیری تقویتی یکی از گرایش‌های یادگیری ماشینی است که از روانشناسی رفتارگرایی الهام گرفته شده است. این روش بر رفتارهایی تمرکز دارد که ماشین باید برای بیشینه کردن پاداش انجام دهد. نحوه آموزش اتوماتای یادگیر نیز از طریق پاداش و جریمه‌ای است که از افزایش یا کاهش تابع کارایی به دست می‌آید [۳۲-۳۴]. پروین و همکاران در مقاله [۵۲] نمونه جدیدی از الگوریتم‌های تکاملی برای محیط‌های پویا ارائه

[1] Self-Organizing Scouts
[2] Multinacional
[3] Hill-valley
[4] Dual-Population
[5] Diversity maintenance
[6] Reinforcement Learning

نموده‌اند. این الگوریتم، ترکیبی از چندین الگوریتم پایه است. الگوریتم تکاملی جدید شامل الگوریتم بهینه‌سازی جمعی ذرات، مدل مبتنی بر الگوریتم ژنتیک، مدل یادگیری تقویتی و الگوریتم شبیه‌سازی حرارتی است. در الگوریتم پیشنهادی به ازای هر ذره در بهینه‌سازی جمعی ذرات، یک مدل یادگیر وجود دارد که نحوه رفتار مناسب آن ذره را نشان می‌دهد. وضعیت‌های هر مدل یادگیر، تنظیم‌کننده یک ذره شامل یک سه‌تایی آتاماتی یادگیر است. مولفه اول پراکندگی بهینه‌های محلی را بیان می‌کند، مولفه دوم فاصله نرمال بهینه محلی ذره از بهینه سراسری را بیان می‌دارد و مولفه سوم فاصله نرمال ذره از بهینه محلی ذره را بیان می‌کند.

در [۵۳] یک روش بهینه‌سازی جمعی ذرات مبتنی بر اتوماتای سلولی[1] به‌نام CellularPSO پیشنهاد شده است. ایده اصلی در این رویکرد، بهره‌گیری از تعاملات محلی در آتوماتای سلولی[2] (CA) و تقسیم کردن جمعیت ذرات در داخل سلول‌های اتوماتای سلولی می‌باشد. هر گروه برای یافتن بهینه محلی تلاش می‌کند که این کار باعث پیدا کردن بهینه سراسری می‌شود. در این روش یک اتوماتای سلولی با CD سلول برابر در محیط D-بعدی بکار رفته است، بنابراین یک ذره در فضای جستجو می‌تواند به یک سلول از اتوماتای سلولی منتسب شود. این مفهوم برای حفظ تنوع در فضای جستجو بکار رفته است. همچنین مفهومی با نام چگالی ذرات برای هر سلول استفاده شده است، به این صورت که یک حد آستانه برای بیشینه تعداد ذرات که می‌توانند در یک سلول قرار گیرند تعیین شده است. این کار سبب می‌شود که همه ذرات بر روی یک سلول همگرا نشوند، بنابراین فقط یک نسبتی از ذرات می‌توانند عمل جستجو را در یک ناحیه از فضای جستجو انجام دهند و بقیه ذرات عمل جستجو را در نواحی دیگری از فضای جستجو انجام می‌دهند. علاوه بر آن وقتی که چگالی ذرات در سلول از آستانه تعیین شده تجاوز نماید، تعدادی از ذرات در سلول به طور تصادفی انتخاب و مجددا در اتوماتای سلولی مقداردهی می‌شوند. به‌طوری که ابتدا حالت ذره انتخاب شده به حالت غیرفعال تغییر پیدا می‌کند، پس از آن یک سلول به‌طور تصادفی به‌عنوان مقصد برای آن ذره انتخاب می‌گردد و سپس فاصله بین سلول جاری و سلول مقصد محاسبه و در یک متغیر به نام شمارنده مسیر قرار می‌گیرد، پس از آن در مرحله بعدی همسایه‌های سلول مربوطه اطلاعات این ذره غیرفعال را اضافه می‌کنند، این کار تا هنگامی که سلول مقصد، این ذره را دریافت کند ادامه می‌یابد، در این لحظه ذره فعال شده و جستجوی خود را در همسایگی جدید آغاز می‌نماید. علاوه بر آن در این روش یک تعامل محلی بین سلول‌های اتوماتای سلولی وجود دارد تا بتواند تنوع بهتری را در میان سلول‌ها حفظ نماید.

و. بررسی جدیدترین مقالات در زمینه بهینه‌سازی پویا

در این بخش برخی از جدیدترین مقالاتی که برای حل مسائل بهینه‌سازی پویا ارائه شده‌اند، مورد بحث و بررسی قرار می‌گیرند.

در مقاله [۵۴]، مساله مشکل کوله‌پشتی محدودکننده شانس (DCCKP) با محدودیت تغییرات پویا در نظر گرفته شده است. همچنین فرض شده که هر یک از کالاهای موجود در کوله-پشتی وزن مشخصی ندارند و این در حالی است که سود قطعی است. برای مولفه پویا، تنظیمات تعریف شده در [۵۵] دنبال شده است: ظرفیت کوله پشتی[3] در طول زمان در هر τ تکرار به اندازه از پیش تعریف‌شده تغییر می‌کند. علاوه بر این، برای مولفه تصادفی، از رویکرد و تنظیمات پیشنهادی در [۵۶] پیروی شده است. بنابراین، هدف در این مطالعه محاسبه مجدد یک راه‌حل سود حداکثر پس از تغییرات پویا در محدودیت ظرفیت است، در حالی که وزن نامشخص کل می‌تواند با احتمال کمی از ظرفیت فراتر رود. برای بهره‌مندی از بهینه-سازی دو هدفه، نمی‌توان به طور مستقیم هدف دوم مورد استفاده در مطالعات قبلی را به کار برد زیرا آنها فقط جنبه پویا یا تصادفی مساله بهینه‌سازی را به‌صورت جداگانه برای تابع هدف دوم در نظر می‌گیرند. بنابراین، یک تابع هدف معرفی

[1] Collective Optimization of Particles Based on the Cellular Automata
[2] Cellular Automata
[3] Knapsack

شده که با عدم قطعیت‌ها سروکار دارد و جنبه‌های پویایی مسائل را برآورده می‌کند. این هدف کوچکترین ظرفیت کوله پشتی که برای آن یک محدودیت شانس را نقض نمی‌کند ارزیابی می‌کند. این هدف همچنین می‌تواند مجموعه‌ای از راه-حل‌های غیرمسلط را که برای ردیابی بهینه متحرک استفاده می‌شود، حفظ کند. این هدف از نابرابری چبیشف[1] و مرزهای چرنوف[2] استفاده می‌کند. از مزایای این روش می‌توان به این موارد اشاره نمود: با افزایش پیچیدگی فضای مساله، الگوریتم خطای برون‌خطی روش پیشنهادی در این مقاله افزایش چندانی نداشته است، همچنین در این پژوهش نشان داده شده که الگوریتم‌هایی که از چرنوف استفاده می‌کنند از الگوریتم-های دیگر که از نابرابری چبیشف استفاده می‌کنند بهتر عمل می‌کنند. برخی از معایب این روش نیز بدین صورت هستند: این رویکرد محدود به مساله کوله‌پشتی محدود شانس بوده و فرمول‌بندی بایستی با طیف گسترده‌ای از مسائل دیگر سازگار شود. در این روش وقتی تغییرات سریع‌تر(کوچک‌تر) اتفاق می‌افتد، زمان کمتری برای تکامل جمعیت خود دارد. بنابراین فقط یک فرد را که به طور تصادفی در جمعیت خود انتخاب کرده، جهش می‌دهد و این باعث می‌شود شانس کمتری در انتخاب بهترین فرد برای جهش در جمعیت خود داشته باشد.

الگوریتم بهینه‌سازی چندهدفه تکاملی مبتنی بر پیش‌بینی یکی از محبوب‌ترین الگوریتم‌های بهینه‌سازی برای حل مساله بهینه‌سازی چندهدفه پویا است [۵۷ و ۵۸]. از مدل‌های سری زمانی برای پیش‌بینی مجموعه آینده پارتو بر اساس راه‌حل‌های گذشته استفاده می‌کند. با این‌حال، ابعاد متغیرهای تصمیم‌گیری ممکن است برای پیش‌بینی خیلی زیاد باشد. علاوه‌براین، یک واریانس نسبتاً کوچک در متغیرهای تصمیم‌گیری ممکن است منجر به اختلاف زیادی در فضای هدف شود. برای حل این مشکلات، در مقاله [۵۷] یک روش جدید پیش‌بینی همکارانه، که نه‌تنها راه‌حل پارتو، بلکه یک راه‌حل مناسب به‌عنوان تقریب پیش‌بینی در فضای هدف ارائه می‌کند. از این راه‌حل برای هدایت فرآیند جستجو و تسریع در همگرایی استفاده می‌شود. از مزیت‌های این روش این است که، روش [۵۷] نه‌تنها راه‌حل پارتو را پیش‌بینی می‌کند، بلکه پارتو را توسط یک راه‌حل قوی تخمین می‌زند. فاصله بین فرد و بهینه پارتو به-عنوان عملگر انتخاب، برای جایگزینی عملگر انتخاب مرتب-سازی غیرغلبه در مرحله اولیه بهینه‌سازی استفاده می‌شود. از راه‌حل قوی برای هدایت فرآیند جستجو و تسریع در همگرایی استفاده شده است. از معایب روش [۵۷] این است که با افزایش پیچیدگی فضای مساله کارآیی این روش به تدریج کاهش می‌یابد.

۲،۳. طبقه‌بندی مسائل بهینه‌سازی پویا

با توجه به تعداد زیاد طرح‌های پیشنهادی برای حل مسائل بهینه‌سازی پویا و طیف گسترده‌ای از پویایی که یک مساله خاص را پوشش می‌دهد، طراحی یک مجموعه آزمایش‌های ساده و تجدیدپذیر به‌منظور مقایسه الگوریتم‌های مختلف و نتیجه‌گیری روی زمینه‌ای از کاربردهای هر پیشنهاد بسیار حیاتی است. برای این منظور تعدادی از نویسندگان تلاش کردند موضوعات دسته‌بندی‌شده‌ای از مسائل پویایی را نشان دهند. علاوه‌براین یک مقدار قابل توجهی از مسائل محک و مسائل بهینه‌سازی پویا در دو دهه اخیر مطرح شده است. در ادامه، کارهای مربوط در این زمینه شرح داده می‌شود.

یکی از مهم‌ترین موضوعاتی که در این‌گونه مسائل وجود دارد، ارزیابی کارآیی این الگوریتم‌ها برای بهینه‌سازی پویا و به‌دست آوردن کارآیی برخط (آنلاین) هر الگوریتم است. در مرجع [۳] تعدادی معیار پیشنهاد شده، که بر اساس آن محیط-های پویا، دسته‌بندی و آزمایش می‌شوند. معیارهای پیشنهاد شده به‌صورت زیر می‌باشند [۳].

الف) فرکانس تغییرات

یک تابع کارایی متغیر با زمان ممکن است در یک تعداد نامحدودی از راه‌حل‌ها تغییر کند. به‌طوری‌که در واحد زمان، آن تابع ممکن است به‌سرعت یا به کندی تغییر کند (به‌عنوان مثال، تغییرات هر چند ثانیه ظاهر می‌شوند). معیار فرکانس

[1] Chebyshev
[2] Chernoff

تغییرات[1] مشخص می‌کند چند وقت یک‌بار محیط تغییر می‌کند (از تغییرات خیلی کم شروع می‌شود و به بالا ادامه پیدا می‌کند).

ب) شدت تغییر[2]

این معیار معین می‌کند سیستم با چه شدتی در حال تغییر است که بر اساس فاصله بین بهینه قبلی و بعدی مشخص می‌شود. این فاصله می‌تواند به روش‌های مختلف اندازه‌گیری شود.

ج) طول چرخه

طول چرخه[3] این معیار مشخص می‌کند که، چه وقتی آن بهینه به موقعیت قبلی‌اش برمی‌گردد یا حداقل به آن نزدیک می‌شود. بدیهی است، که فرض بر این است که پویایی یک چرخه از تغییر توزیع تابع کارایی به نحوی به شکل و مکان قبلی‌اش برمی‌گردد یا این که به آن نزدیک می‌شود.

د) پیش‌بینی تغییرات

پیش‌بینی تغییرات[4] یک مفهومی را نشان می‌دهد که اگر یک الگو یا روند در تغییرات وجود داشته باشد به آن مساله وابسته است و به‌نحوی پیش‌بینی زمان یا شدت تغییرات بعدی را با توجه به تغییراتی که تاکنون با آن روبه‌رو شده است ممکن می‌سازد. اندازه‌گیری در برخی از وضعیت‌ها مشکل است و مقایسه میان مسائل بهینه‌سازی مختلف ممکن است دور از دسترس باشد، اما ویژگی‌های بالا می‌تواند روی یک مساله متفاوت باشد [۳]. بهتر است این موضوع به‌وسیله مساله محک شناخته شده درک شود. مساله محک کوله پشتی صفر و یک پویا، یکی از معروف‌ترین مسائل محک است که در ادامه تشریح می‌شود [۲۹-۳۱].

۲,۴. مولدهای مسائل بهینه‌سازی پویا

همان‌طور که در [۲۹] گفته شده است، مولدهای مسائل پویا می‌بایست برخی الزامات اساسی را برخوردار باشند یا برخی از خواص مشترک به‌منظور مقایسه کارایی الگوریتم‌های ژنتیک مختلف در محیط‌های پویا را داشته باشند. نویسندگان در [۲۹] ویژگی‌های مورد نیاز را شرح داده‌اند:

۱- می‌بایست به‌گونه‌ای باشد که پارامترهای محیطی مرتبط با جنبه‌های مختلف مساله را تغییر دهد.

۲- می‌بایست برای درک پویایی مختف ساده باشد، مانند فرکانس تغییرات، شدت تغییرات، سیکل (چرخه)

۳- می‌بایست به‌راحتی با پیچیدگی و سختی مسائل پویا وفق داده شود.

۴- می‌بایست محاسبات کارآمدی جهت محیط‌های پویا را داشته باشد.

۵- می‌بایست به‌سادگی جهت تجزیه و تحلیل به صورت فرمال انجام‌پذیر باشد.

اخیرا مرجع [۲۹] مولدی را برای مسائل بهینه‌سازی پویا گسترش داده است که چرخه محیط‌ها را ایجاد کند. این مساله به‌طور ویژه جهت آزمایش طرح‌های مبتنی بر حافظه مفید است، چون همان‌طور که قبلا گفته شد، این نوع روش زمانی بهتر کار می‌کند که تغییرات چرخه‌ای (دوره‌ای) باشد. از آن‌جایی‌که روش پیشنهادی این مقاله یک طرح مبتنی بر حافظه است، از همین مساله محک برای شبیه‌سازی مسائل بهینه‌سازی پویا در این مقاله استفاده شده است.

۲,۵. سیستم‌های خودسازمانده بحرانی

پیشرفت‌های روزافزون در فن‌آوری، محیط محاسباتی را پویا و پیچیده ساخته است. در چنین شرایطی، سیستم‌ها نیازمند درجه‌ی بالاتری از خودمختاری هستند تا امکان ادامه‌ی عملکرد، بدون دخالت انسان را کسب کنند. در نتیجه‌ی بالا رفتن خودمختاری در سیستم‌های امروز، پیچیدگی آن‌ها نیز افزایش می‌یابد. وفق‌پذیری و همکاری طبیعی سیستم‌های زیستی ایده‌ی رویکردهای جدیدی است که برای مدیریت این افزایش پیچیدگی و ایجاد متدولوژی‌های قوی در طراحی سیستم‌ها و حل مسائل محاسباتی استفاده شده است. سیستم‌های بهره گرفته از این متدولوژی‌ها نوعاً با عنوان

[1] Frequency of change
[2] Severity of change
[3] Cycle length
[4] Predictability of change

سیستم‌های خودسازمانده بحرانی[1] شناخته می‌شوند. در سال‌های اخیر، سیستم‌های خودسازمانده بحرانی، به‌عنوان مهم‌ترین نوع از سیستم‌های خودسازمانده، توجه ویژه‌ای را به خود جلب کرده‌اند. سیستم‌های خودسازمانده بحرانی، از تعدادی مولفه خودمختار تشکیل شده‌اند. راهکار کنترلی در میان این مولفه‌ها توزیع شده است. هر یک از مولفه‌های سیستم سعی می‌کنند با تکیه بر اطلاعات و سیاست‌های محلی و همچنین تعامل با دیگر مولفه‌ها، به اهداف محلی خود دست یابند. در فیزیک نقطه بحرانی نقطه‌ای است که در آن تغییر اساسی در رفتار یا ساختار یک سیستم به‌وجود می‌آید، به‌عنوان مثال تغییر حالت از جامد به مایع. در حالت کلی یک پارامتر کنترل وجود دارد که یک عامل خارجی مثل فرد آزمایش‌کننده می‌تواند رفتار سیستم را در حالت بحرانی کنترل کند، به‌عنوان مثال در حالت ذوب این پارامتر حرارت است [۳۹]. حالت بحرانی خودسازمان‌داده‌شده روندی برخلاف این موضوع دارد؛ بدین معنی که به‌صورت مستقل از هر پارامتر کنترلی عمل می‌نماید و سیستم با استفاده از برهم‌کنش‌های درونی، حالت بحرانی خود را کنترل می‌کند. در این حالت، انتشار برهم‌کنش‌ها بین اجزای سیستم نسبت معکوس با فراوانی رخداد آن‌ها دارد. این نسبت را می‌توان با استفاده از رابطه قانون–قدرت نشان داد که این رابطه در ادامه همین بخش آورده شده است [۳۹]. پویایی و پیچیدگی سیستم‌های خودسازمانده بحرانی چالش‌های بسیاری برای طراحان این نوع از سیستم‌ها ایجاد کرده است. از این میان تضمین درستی رفتار سیستم در حین و بعد از خودسازماندهی دارای اهمیت ویژه‌ای است.

یک سیستم خودسازمانده بیان می‌کند که [۳۹]:

- آیا سیستم‌های بسط‌دهنده[2] باز به صورت محلی با درجه‌ای از آزادی با یکدیگر تعامل دارند؟
- آیا اجزای سازنده خود را با قوانین ساده مدیریت می‌کنند؟
- آیا آستانه‌هایی در داخل سیستم وجود دارد؟
- آیا باید در یک سرعت بی‌نهایت هدایت شوند؟
- فشار در سیستم ایجاد می‌شود تا زمانی که از حد آستانه عبور کند.
- مشخصه‌های واریانس مکانی/زمانی نقطه بحرانی انتقال فاز را بدون نیاز به تنظیم دقیق مقادیر پارامترهای کنترل نمایش می‌دهد.
- پیشرفت به‌سوی یک وضعیت بحرانی بین نظم و آشوب صورت می‌گیرد.

در یک سیستم خودسازمانده بحرانی، اختلالات کوچک می‌تواند منجر به سقوط‌های عظیم[3] شود. علاوه‌براین، اختلال مشابه ممکن است منجر به سقوط‌های کوچک یا بزرگ شود، که در نهایت یک نسبت "قدرت– قانون" بین اندازه رویدادها و فراوانی آن‌ها آشکار می‌شود [۳۹]:

$$Ns = Cs^{-\tau} \qquad (۱)$$

که s اندازه رویدادها است، C ثابت است و $-\tau$ شیب قدرت–قانون است، که می‌تواند در یک مقیاس لگاریتمی نشان داده شود [۳۹]:

$$\log Ns = \log C - \tau \log s \qquad (۲)$$

این بدین معنی است که رویدادهای بزرگ فاجعه‌بار، ممکن است سیستم را از زمانی به زمان دیگر تغییر دهند و دوباره آن را پیکربندی کنند. این روابط قدرت–قانون بین اندازه رویدادها و فراوانی آن‌ها در طبیعت گسترده هستند. برای مثال، توزیع زلزله، قانون گوتنبرگ–ریشتر را دنبال می‌کند، که نسبت قدرت–قانون بین مقیاس زلزله‌هایی است که در یک دوره زمانی مشخص در یک منطقه خاص اتفاق می‌افتند. می‌توان بین سه نوع قدرت– قانون از سیستم‌های فیزیکی تمایز قائل شویم. به‌عنوان مثال، توزیع تراکم طیفی قدرت (مانند نویز صورتی) توسط رابطه ذیل توصیف می‌شود [۳۹]:

$$P(f) \propto \frac{1}{f^{\alpha}} \qquad (۳)$$

که f فرکانس است، $P(f)$ قدرت فرکانس است و α یک عدد واقعی بین 0 و 2.0 است، اما معمولا نزدیک به 1.0 است. اگر $\alpha = 0$ باشد آنگاه $P(f)$ نویز سفید نامیده می‌شود؛ اگر $\alpha = 2.0$ باشد نویز قرمز یا نویز برونیان نامیده می‌شود و

[1] Self-Organized Criticality (SOC)
[2] Dissipative
[3] Avalanches

هنگامی که $\alpha = 1.0$ است تابع $P(f)$ نویز صورتی را توصیف می‌کند. به‌طور کلی، این تابع توصیف می‌کند کدام فرکانس در رفتار زمانی سیستم مورد توجه قرار گرفته است، در تراکم طیفی قدرت، فقط مربع تبدیل فوریه سیگنال مورد نظر است. قدرت- قانون دیگری نیز در توزیع‌های اندازه مطرح می‌شود (مانند قوانین گتنبرگ-ریشتر) [۳۹]:

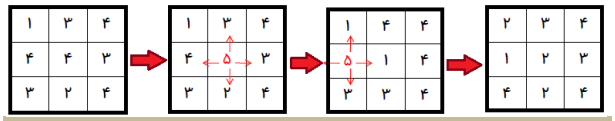
$$N(s) \propto \frac{1}{s^{\tau}} \qquad (۴)$$

که در آن s اندازه یک رویداد (یا اندازه) و $N(s)$ نشان‌دهنده یک توزیع از فرکانس چنین رویدادی است. نوع سوم قدرت- قانون در توزیع زمانی رویدادها شناسایی می‌شود، جایی که τ یا مدت زمان رویداد است یا زمان بین رویدادها که توسط رابطه (۵) بیان شده است [۳۹]:

$$N(\tau) \propto \frac{1}{\tau^{\gamma}} \qquad (۵)$$

سیستم خودسازمانده ممکن است خط مشترک بین طیف گسترده‌ای از پدیده‌های طبیعی فعال، در ناحیه بین نظم و آشوب باشد که این روابط قدرت- قانون را نشان می‌دهد. نوعی از سیستم خودسازمانده شناسایی شده، یک آتوماتای سلولی به‌نام تپه شن[1] است. در ادامه به بیان مدل تپه شنی باک-تانگ-ویزنفلد[2] می‌پردازیم.

در سال ۱۹۸۷، باک و همکاران [۳۹] پدیده خودسازماندهی مرتبط با سیستم‌های پویا را شناسایی کردند. اولین سیستمی که در خودسازماندهی بحرانی مشاهده شد، مدل **تپه** شن نام‌گذاری شده و شامل یک آتوماتای سلولی است که در آن در هر سلول شبکه، یک مقدار وجود دارد که مربوط به شیب **تپه** است. شکل (۲) مدل تپه شنی را نشان می‌دهد. در این شکل همان‌گونه که مشخص می‌باشد، هر سلول که بیش از چهار دانه داشته باشد سرریز نموده و به هر یک از سلول‌های مجاور خود یک دانه می‌دهد. سرریز زمانی متوقف می‌شود که تمامی سلول‌ها بیش از ۴ دانه نداشته باشند، که در این حالت اصطلاحاً سیستم به تعادل رسیده است.



شکل(۲): مدل تپه شنی

در ساده‌ترین حالت، **تپه** شن یک شبکه خطی از L سایت $(x_1, x_2, \dots, x_L)$ است که در آن شن به‌طور تصادفی به یک دانه کاهش یافته است. تعداد دانه‌های موجود در محل x_j توسط تابع $h(x_j)$ نشان داده می‌شود که ممکن است به‌عنوان ارتفاع **تپه** در سایت x_j ذکر شود. این دانه‌ها در شبکه انباشته می‌شوند تا زمانی‌که تفاوت ارتفاع بین سایت‌های مجاور از مقدار آستانه تجاوز نکند. اگر چنین اتفاقی بیفتد، دانه از سایت j به‌سمت بالا به سایت مجاور $j+1$ می‌رود و اگر تفاوت ارتفاع بین سایت $j+1$ و $j+2$ نیز بیش از آستانه باشد سپس دانه دوباره سرریز می‌کند. سرریز تنها زمانی توقف می‌شود که دانه به سایتی برسد که در آن شیب از مقدار تعریف شده به‌عنوان آستانه تجاوز نمی‌کند. با توجه به سیستم کلی، می‌توان گفت که سرریز زمانی متوقف می‌شود که **تپه** به‌حالت تعادل می‌رسد. اگر به‌جای ارتفاع، تمرکز بر شیب قرار گیرد، فرآیند ممکن است با توجه به روابط زیر توصیف شود [۳۹]:

$$Z_j \rightarrow Z_{j+1} \qquad (۶)$$

$$Z_{j+1} \rightarrow Z_{j+1} - 1 \qquad (۷)$$

$$\text{For } Z_j > Z_C\text{: } Z_j \rightarrow Z_j - 2v \qquad (۸)$$

$$Z_{j\pm1} \rightarrow Z_{j\pm1} + v \qquad (۹)$$

که $Z_j = h(x_j) - h(x_j + 1)$ شیب در سایت j است، Z_C مقدار بحرانی (یا آستانه) برای شیب است و v تعداد دانه‌هایی است که سرازیر می‌شوند. یک شرط مرزی وجود دارد که بیان می‌کند، اگر دانه سرازیر کند تا زمانی‌که به سایت x_L می‌رسد، سرزیر بعدی **تپه** را معلق می‌کند. روابط (۶) و (۷) کاهش دانه را توصیف می‌کنند: شیب در سایت j افزایش می‌یابد در حالی که شیب در سایت $j+1$ کاهش می‌یابد. سپس، اگر شیب Z_j بیش از شیب بحرانی Z_C باشد، v دانه سرزیر می‌کنند و شیب‌های مجاور افزایش می‌یابد (مطابق روابط (۸) و (۹)). بسته به وضعیت شبکه و موقعیتی که در آن دانه جدید رها شده

[1] Sand pile
[2] Bak-Tang-Wisenfeld Sandpile Model

است، واکنش‌های مختلفی ممکن است رخ دهد. رها کردن یک دانه ممکن است تغییراتی در سیستم ایجاد نکند اگر در سلولی با مقدار کمتر از آستانه سقوط کند (به غیر از افزایش ارتفاع آن سلول) اما همچنین ممکن است سقوط‌های بزرگی تولید کند که به شدت منجر به تغییر شکل **تپه** گردد. اگر $v = 2$ را در نظر بگیریم و تپه شن را با رها کردن یک دانه در یک سایت تصادفی اجرا کنیم، به‌روزرسانی تپه تا زمانی‌که سرریز به پایان رسد ادامه می‌یابد (درصورتی‌که سرریز شروع شده باشد). تکرار روند برای تعداد زیادی از تکرارها و سپس توزیع رویدادهای اندازه (یعنی نسبت احتمال (یا فراوانی) سقوط‌ها و اندازه آن‌ها) یک رفتار قدرت–قانون را به‌صورت رابطه (۷) نشان می‌دهد (یعنی فراوانی به‌عنوان یک تابع از اندازه s) و در یک مقیاس لگاریتمی منجر به ایجاد یک قانون خطی با شیب τ می‌شود. باید توجه داشته باشید که با تغییر اندازه L از شبکه، شیب قدرت–قانون نیز تغییر می‌کند و هنگامی که L افزایش می‌یابد، پایین‌تر می‌رود. تپه شن اکنون می‌تواند به دو بعد تعمیم یابد. در واقع، این موضوع واقعا برای این بخش اهمیت دارد، زیرا جهش تپه شن پیشنهادی روی یک شبکه دو بعدی با اندازه $n \times L$ تکامل می‌یابد (که n اندازه جمعیت است و L طول کروموزوم است). باز هم روابط برای شیب $Z(x,y)$ به شکل زیر تعریف می‌شوند [۳۹]:

$$Z(x-1), y \rightarrow Z(x-1, y) - 1 \quad (۱۰)$$

$$Z(x-1, y) \rightarrow Z(x, y-1) - 1 \quad (۱۱)$$

$$Z(x, y) \rightarrow Z(x, y) + 2 \quad (۱۲)$$

$$\text{if } Z(x, y) > Z_C: Z(x, y) \rightarrow Z(x, y) - 4 \quad (۱۳)$$

$$Z(x, y \pm 1) \rightarrow Z(x, y \pm 1) + 1 \quad (۱۴)$$

$$Z(x \pm 1, y) \rightarrow Z(x \pm 1, y) + 1 \quad (۱۵)$$

روابط (۱۰)، (۱۱) و (۱۲) روش اضافه کردن دانه به شبکه در سایت (x,y) را توصیف می‌کنند. دانه‌های شن به‌طور تصادفی بر روی شبکه تپه رها می‌شوند و ارزش سلول‌ها افزایش می‌یابد. سپس اگر شیب در سایت (x,y) بزرگتر از مقدار بحرانی Z باشد، دانه‌ها توسط سایت‌های همسایه آن توزیع می‌شوند – روابط (۱۳)، (۱۴) و (۱۵). اگر یکی از آن سایت‌ها نیز بیش از مقدار آستانه باشد، سقوط ادامه می‌یابد. اگر شبکه قبلا به یک مقدار بحرانی رانده شده باشد، ممکن است شاهد سقوط‌هایی با اندازه‌های متغیر از یک واحد به رویدادهایی که تقریبا تمام تپه را دوباره تنظیم می‌کنند، باشیم. باز هم احتمال یک سقوط در نسبت قدرت–قانون به اندازه سقوط است و سقوط‌ها در تمام مقیاس‌ها رخ می‌دهند. سقوط‌های بزرگ بسیار نادر هستند درحالی‌که سقوط‌های کوچک اغلب آشکار می‌شوند. بدون اصلاح دقیق پارامترها، سیستم به یک حالت بحرانی عدم تعادل می‌رسد. پس از شناسایی در تپه شن، سیستم خودسازمانده سپس در چندین سیستم پیچیده مشاهده خواهد شد. به‌عنوان مثال، در سال ۱۹۹۰ باک و همکاران [۳۹] مدل آتش جنگل را معرفی کردند، سیستم دیگری که خودسازماندهی را نمایش می‌دهد. بعدها، باک و اسنپن (۱۹۹۳) [۴۰]، مدل باک/اسنپن[1] را برای درک حوادث انقراض و تعادل متناوب ارائه دادند. این، مدلی الهام بخش برای GA است که بیشتر مربوط به روش پیشنهاد شده در این بخش می‌باشد و همچنین مدلی الهام‌بخش برای یک پارادایم محاسباتی جدید به‌نام بهینه‌سازی خارجی است.

۳. روش پیشنهادی

در این مقاله یک الگوریتمی ارائه می‌شود که بر مبنای الگوریتم ژنتیک استاندارد بوده ولی چندین روش برای بهبود کارآیی این الگوریتم در حل مسائل بهینه‌سازی پویا به آن اضافه می‌شوند. روش پیشنهادی، ترکیبی از موارد زیر است که این مراحل در ادامه تشریح می‌شوند:

۱- در مرحله انتخاب از الگوریتم ژنتیک، راهکاری ارائه‌شده است که روی جفت‌گزینی و جلوگیری از ادغام میان افراد مشابه از طریق یک راهکار مبتنی بر حافظه عمل می‌نماید که از این طریق، تنوع الگوریتم ژنتیک را حفظ می‌کند.

۲- دومی یک عملگر جهش خودسازمانده بحرانی [۴۳] است که می‌تواند نرخ‌های جهش خودتنظیم‌شونده را با یک

[1] The Bak-Sneppen Model

توزیع ویژه بر مبنای الگوریتم تپه شن تعمیم‌یافته، انجام دهد که برای بهینه‌سازی پویا مناسب است.

۳- برای ذخیره راه‌حل‌های مفید گذشته از یک نوع حافظه خاص، به‌نام حافظه تخمین تراکم گوسی استفاده شده است.

در ادامه مراحل روش پیشنهادی به‌طور مفصل تشریح می‌شود.

مرحله اول: تولید جمعیت اولیه به‌صورت تصادفی

مرحله دوم: ارزیابی مقدار برازش (شایستگی) برای هر کروموزوم از جمعیت

مرحله سوم: ذخیره مناسب‌ترین راه‌حل‌ها در حافظه تخمین تراکم گوسی

مرحله چهارم: تشخیص تغییر در محیط و استفاده از مناسب‌ترین راه‌حل‌های حافظه در جمعیت اصلی

مرحله پنجم: عملگرانتخاب

مرحله ششم: عملگرجفت‌گزینی دو نقطه‌ای

مرحله هفتم: به‌کارگیری راهکار جهش خودسازمانده پیشنهادی

مرحله هشتم: در صورت رسیدن به معیار خاتمه، به پایان الگوریتم برو.

۳،۱. حافظه تخمین تراکم با خوشه‌بندی گوسی

وقتی با یک جهان در حال تغییر مواجه می‌شوید، انسان‌ها نه تنها به آینده بلکه به گذشته هم توجه می‌کنند. توجه کردن به راه‌حل‌های مشابه به ما در تصمیم‌گیری برای آینده کمک می‌کند. زمانی‌که با وضعیتی روبه‌رو می‌شویم که قبلا آن را تجربه کرده باشیم، بهتر می‌توانیم با آن مواجه شویم. اگر در حل مسائل پویا در هنگام جستجو از اطلاعات گذشته داخل بهینه‌سازی و یادگیری استفاده شود، می‌تواند به فرآیند جستجو کمک شایانی نماید. اکثر مسائل در دنیای واقعی ماهیتی پویا دارند که در این مسائل محیط در حال تغییر است. در اکثر مواقع این محیط دارای تغییرات مشخص شده است که با ذخیره کردن راه‌حل‌های گذشته در حافظه می‌توان از آن‌ها در محیط‌های جدید استفاده نمود. محمدپور و همکارانش در مراجع [۲۱-۲۴] اثبات نموده‌اند که به‌کارگیری حافظه در حل مسائل بهینه‌سازی پویا بسیار موثر می‌باشد. محققین برای بهبود حافظه در حل مسائل بهینه‌سازی پویا راهکارهای مختلفی ارائه نموده‌اند که هر کدام به‌نوعی توانسته نقاط ضعف حافظه استاندارد را برطرف نماید.

حافظه به بهینه‌سازی پویا از طرق مختلف کمک می‌کند: نگهداری راه‌حل‌های خوب گذشته، سرعت بخشیدن به جستجو برای راه‌حل‌های مشابه بعد از تغییرات محیطی، مدخل‌های حافظه کمک زیادی به جستجو بعد از تغییر می‌کنند، به ایجاد تنوع در فرآیند جستجو کمک می‌کند و در نهایت، حافظه به ایجاد یک مدل از مسائل پویا در هر زمان کمک می‌کند. انواع متعددی از حافظه وجود دارد که یک سیستم حافظه استاندارد برای بهینه‌سازی پویا پدید آمده است. در این سیستم حافظه استاندارد تعداد محدودی راه‌حل‌ها ذخیره شده که در جستجو به‌کار می‌رود تا در صورت تغییر محیط نیز فرآیند جستجو را به‌سمت راه‌حل‌های خوب هدایت کنند. اندازه حافظه محدود و کوچک است و ذخیره راه‌حل‌های کافی و تصحیح کل حافظه سخت است.

برای درک بهتر این موضوع، ابتدا نقاط ضعف حافظه‌های استاندارد، مختصرا تشریح شده است:

الف) مدل محدود شده از فضای جستجوی پویا

حافظه‌های استاندارد یک مدل ساده از بهترین نواحی پویا را با توجه به مساله موجود در طول زمان ارائه می‌دهند. در این صورت مدل ارائه شده بسیار محدود می‌باشد. از آن‌جائی که اندازه حافظه باید در مقایسه با جمعیت کوچک‌تر باشد یک چشم‌انداز از چندین مدل که غالبا در مسائل پویا ارائه می‌گردند می‌تواند تنها شامل یک مدخل ورودی به حافظه باشند.

اندازه محدود حافظه: از زمانی‌که حافظه‌های استاندارد با حافظه‌های بسیار کوچک ترکیب شدند، محدودیت‌هایی در فضاهای تحت پوشش حافظه به‌وجود آمد. وقتی تعداد قله‌ها در مسائل پویا از اندازه حافظه بیشتر شود نواحی خوب ممکن نیست توسط حافظه پوشش داده شوند. این مسئله باعث

فرارتر شدن حافظه‌ها شود و تعداد راه‌حل‌های مناسب افزایش یابد. تا آن‌جایی‌که راه‌حل‌های ذخیره‌سازی شده در حافظه به احتمال کمتری مشابه راه‌حل‌های موجود هستند.

ب) مشکل اصلاح مدخل‌های حافظه

با توجه به اینکه حافظه‌ها معمولا جستجو را به مکان‌های از پیش تعیین شده‌ای (امید بخش) راهبری می‌کنند، زمان‌هایی رخ می‌دهد که حافظه مانع جستجوی مستمر می‌گردد. در این حالت حافظه نیز جستجو را به تعدادی ناحیه‌های زیربهینه هدایت می‌کند. فاکتورهای دخیل در این کار باعث می‌گردد تا حافظه پس از هر بار ذخیره‌سازی نسبت به اصلاح مدخل‌های ذخیره‌سازی شده ناکام بماند که تنها راه تغییر مدخل حافظه جایگزین کردن آن است.

ج) تنوع محدود شده

از آن‌جایی که حافظه‌ها در اضافه شدن تنوع به جمعیت پس از هر تغییر کمک می‌کنند، تنوع‌های موجود محدود به نواحی هستند که قبلا مورد جستجو قرار گرفته‌اند. بنابراین حافظه‌ها با اتکای بسیار به الگوریتم‌های بهینه‌سازی جهت پیدا کردن راه‌حل‌های متنوع اقدام می‌کنند که بعضی اوقات ممکن نیست. حافظه بایستی با تکنیک‌های تنوع همراه شود. حافظه‌ها می‌توانند مقدار زیادی از تنوع را فراهم نمایند. در بهترین حالت زمانی‌که یک حافظه تنوع کافی ندارد یک تکنیک تنوع می‌تواند موجب تزریق مقادیر زیادی ماده ژنتیکی جدید در هر جمعیت شود، بنابراین از زمانی‌که یک حافظه منجر به تنوع زیادی می‌گردد، تکنیک‌های تنوع ممکن است باعث افت کارایی آن‌ها شود.

با تشریح نقاط ضعف برای حافظه‌های استاندارد، می‌توان نتیجه گرفت که باید حافظه‌ای ارائه شود که این نقاط ضعف را در حافظه‌های استاندارد برطرف نماید. حافظه پیشنهادی این مقاله، حافظه‌ای به‌نام حافظه تخمین تراکم گوسی است که در ادامه شرح داده شده است.

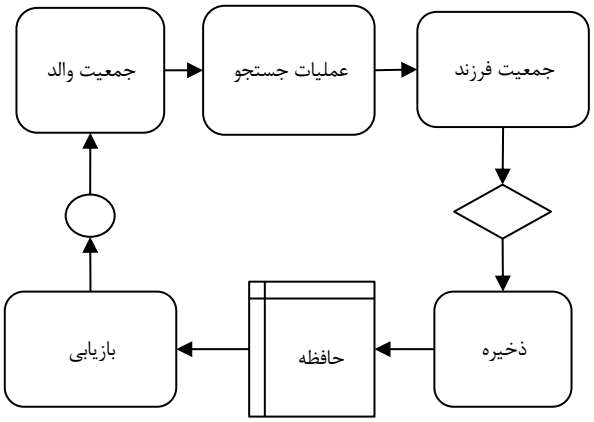
۳،۱،۱. حافظه تخمین تراکم گوسی

به‌جای ذخیره کردن تنها یک نقطه (کروموزوم) در حافظه، پیشنهاد می‌شود خوشه‌هایی از نقاط در هر مدخل حافظه ذخیره شوند و مدلی از هر نقاط در هر خوشه ایجاد شود. با این روش الگوریتم قادر به ذخیره کردن نقاط بیشتری خواهد بود و در عین حال سربار محاسباتی برای حافظه کم می‌شود. حافظه تخمین تراکم مدل‌های احتمالی درون حافظه را با ساختن تخمین تراکم از نواحی امیدبخش فضای جستجو، در طول زمان ایجاد و حفظ می‌کند. حافظه تخمین تراکم اجازه می‌دهد که بسیاری از راه‌حل‌ها در حافظه ذخیره شوند و مدل‌های طولانی مدت از فضای جستجوی پویا به وجود می‌آورد و اجازه می‌دهد تا زمانی‌که سربار کم است مدخل‌های حافظه به‌راحتی تصحیح شوند. در واقع، حافظه تخمین تراکم گوسی، مشکل محدودیت ذخیره‌سازی راه‌حل‌های قبلی، ضعف مدل‌های فضای جستجوی پویا، و مشکلات تصحیح محتوای حافظه را برطرف می‌کند.

این نوع حافظه، به‌جای این‌که فقط یک راه‌حل را در حافظه ذخیره کند، ساختار مشابه را داخل حافظه استاندارد نگهداری می‌کند. حافظه تخمین تراکم تعداد راه‌حل‌های درون یک مدخل را خوشه‌بندی می‌کند، آن‌گاه با استفاده از مدل‌های احتمالی تراکم نقاط درون خوشه‌ها را تخمین می‌زند. این حافظه اجازه می‌دهد تا راه‌حل‌های قبلی زیادی بدون افزایش سربار ذخیره شوند. این حافظه مدل‌های احتمالاتی از راه‌حل‌های گذشته برای بهبود کارایی یادگیری و بهینه‌سازی در محیط‌های پویا ایجاد و نگهداری می‌کند. وقتی مدل‌های غنی‌تر زیادی از فضای جستجوی پویا نسبت به حافظه استاندارد ایجاد می‌شوند، حافظه‌های تخمین تراکم می‌توانند کارآمدتر ایجاد و با میزان کمتری از سربار نگهداری شوند. حافظه‌های تخمین تراکم مدل‌های طولانی‌مدت بهتری از فضای پویا را ایجاد می‌کنند و باعث تصحیح آسان‌تر مدخل‌های حافظه می‌شوند.

برای درک بهتر موضوع شکل (۳) نمودار یک الگوریتم جستجوی مبتنی بر جمعیت با حافظه را نشان می‌دهد. الگوریتم با یک جمعیت والد از راه‌حل‌ها آغاز می‌شود. این راه‌حل‌ها با استفاده از عملیات جستجو به جمعیت فرزندها

تبدیل می‌شود. جمعیت فرزندان با افراد بازیابی شده از حافظه ترکیب می‌شوند و به‌شکل جمعیت والد برای تکرار بعدی از الگوریتم جستجو استفاده می‌شود. افرادی از جمعیت فرزندان ممکن است انتخاب شود که باید در حافظه ذخیره شوند.



شکل (۳): نمودار الگوریتم جستجوی مبتنی بر جمعیت با حافظه

حافظه یک تعداد متناهی از مدخل‌ها را ذخیره می‌کند که حاوی اطلاعات تولید شده توسط فرآیند جستجو است و ممکن است پس از این‌که تغییرات در محیط انجام شود برای کمک به جستجو مورد استفاده قرار گیرد. توابع حافظه به‌عنوان یک نسخه پیچیده‌ای از نخبه‌گرایی هستند که راه‌حل‌های خوب در جمعیت صرف‌نظر از نتایج حاصل از جستجو را حفظ می‌کنند.

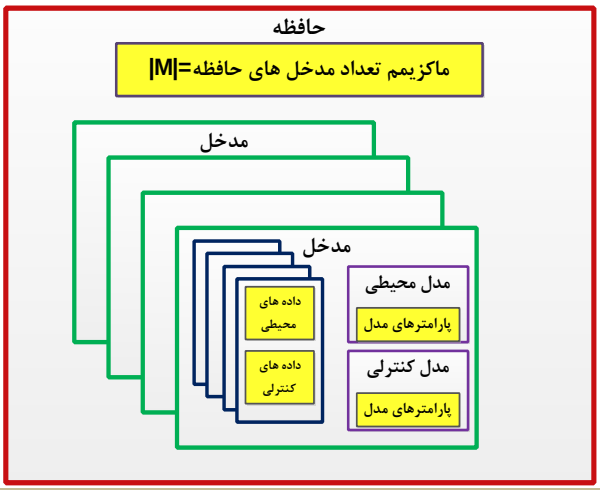
حافظه یک اندازه ثابت دارد، هنگامی‌که با جستجوی مبتنی بر جمعیت مورد استفاده قرارگیرد آن اندازه به‌طور کلی نسبت به اندازه کل جمعیت کوچک است. در یک مساله پویا، همه افراد باید در هر مرحله از فرآیند جستجو ارزیابی شوند که شامل آن دسته از افراد ذخیره شده در حافظه هم می‌باشد. با کوچک نگه داشتن حافظه، بیشتر ارزیابی‌ها برای روند جستجوی اصلی رزرو می‌شوند. اگر اندازه کل جمعیت را P در نظر بگیریم، اندازه حافظه برابر $m = p/10$ است که این حافظه به‌طور جداگانه از جمعیت ذخیره می‌شود.

یک حافظه تخمین تراکم نقاط را در تعداد محدودی از مدخل‌های حافظه ذخیره‌سازی می‌کند. اطلاعات ذخیره شده در یک مدخل حافظه را می‌توان به دو دسته تقسیم کرد: اطلاعات محیطی و اطلاعات کنترلی [۳ و ۲۱]. اطلاعات محیطی کمک می‌کند تا تصویری از وضعیت فعلی مساله داشته باشیم. در صورت تغییر مساله، اطلاعات محیطی ممکن است برای تعیین شباهت بین محیط فعلی و محیطی که در مدخل حافظه ذخیره شده استفاده شود. اطلاعات کنترلی شامل اطلاعاتی است که در فرآیند جستجو استفاده می‌شود. برای مثال اطلاعات کنترلی ممکن است فقط یک راه‌حل ذخیره شده در قبل باشد که می‌تواند دوباره در جمعیت جاگذاری شود یا ممکن است حاوی یک مدل احتمالی از جمعیت بر اساس الگوریتم تکاملی باشد که می‌تواند برای مقداردهی مجدد جمعیت اولیه بعد از یک تغییر استفاده شود.

در حافظه استاندارد، هر مدخل حافظه می‌تواند یک نقطه تکی را ذخیره کند؛ ولی در حافظه تخمین تراکم، هر مدخل ممکن است تعدادی نقاط را ذخیره کند. یک حافظه ممکن است تا $|M|$ مدخل داشته باشد. هر مدخل اصولا از مجموعه‌ای نقاط، یک مدل از اطلاعات محیطی در آن نقاط و همچنین، یک مدل از اطلاعات کنترلی در آن نقاط تشکیل شده است.

شکل (۴) ساختار یک حافظه تخمین تراکم را نشان می‌دهد. مدل‌ها در حافظه تخمین تراکم ممکن است ساده شده‌ای از میانگین تمام نقاط باشد یا از مدل‌های احتمالی پیچیده‌ای نیز استفاده کرده باشد. در این مقاله مدل‌های عامی که به‌کار برده می‌شوند تا یک مدخل حافظه را نشان دهند، مدل خوشه‌بندی گاوسی چندمتغیره هستند. در نسخه گاوسی، متوسط و کوواریانس همه نقاط موجود در مدخل حافظه به جهت توصیف مدل مورد استفاده قرار می‌گیرند.

زمانی‌که مدخل‌ها برای نخستین بار ساخته می‌شوند، ممکن است برای محاسبه کوواریانس معتبر نقاط کافی نباشد. در این-حالت، نقاط تصادفی اطراف میانگین به‌صورت موقت برای تکمیل مدل اضافه می‌شوند. از آن‌جایی‌که مدخل حافظه شامل نقاط متعددی می‌باشد، فقط مدل محیطی و مدل کنترلی جهت برقراری تعامل با آن مدخل لازم است. هم‌چنین به‌سبب این‌که مدخل‌ها از هزاران نقطه تشکیل شده‌اند، این مساله میزان سربار حافظه تخمین تراکم را پایین نگه می‌دارد.



شکل (۴): حافظه تخمین تراکم از تعداد محدودی از مدخل‌ها ایجاد شده است. هر مدخل شامل مجموعه‌ای از نقاط است که هر نقطه شامل داده محیطی و کنترلی از زمانی‌که آن نقطه دخیره شده بوده است.

شکل (۵) نحوه عملکرد متفاوت حافظه تخمین تراکم را نسبت به حافظه استاندارد نشان می‌دهد. در این مثال، یک فضای جستجوی یک بعدی وجود دارد که شامل چهار قله با شکل‌ها و شایستگی مختلف است. فرض کنید که الگوریتم جستجو سعی دارد تا نقاط علامت‌گذاری شده با خطوط ممتد را طی مدت زمانی مشخص درون حافظه ثبت کند. حافظه استاندارد نقطه با بهترین شایستگی را انتخاب می‌کند؛ از این‌رو همیشه نقطه مزبور نزدیک‌ترین نقطه به مرکز حافظه است. حافظه تخمین تراکم نقاط را بر اساس قله خوشه‌بندی می‌کند و آن‌گاه به‌ازای هر خوشه یک مدل می‌سازد. در بالای شکل یک مدل گاوسی به‌ازای هر قله نشان داده شده است. پیرامون این مثال، مقدار میانگین محاسبه شده به‌ازای هر خوشه شایستگی بیش‌تری دارد و نزدیک‌ترین نقطه به قله بیشینه می‌باشد. فراتر از استفاده از هندسه اقلیدسی، مدل‌های گوسی اطلاعات بیش‌تری را به هنگام افزودن یک نقطه جدید به حافظه ارائه می‌کنند. قله در قسمت چپ به حد کافی بزرگ است و هر چه نقاط از حد متوسط دورتر باشند، احتمال پذیرفته شدن آن‌ها نیز بیش‌تر است. پیرامون خوشه‌هایی که تا به‌حال مدل دقیقی برای آن‌ها ارائه نشده است، مدل می‌بایست ضمن افزودن نقاط به حافظه اصلاح گردد. چنان‌چه مدل‌ها دقیق‌تر از قبل شوند، راه‌حل‌های بازیابی شده از حافظه نیز بهتر خواهند شد.

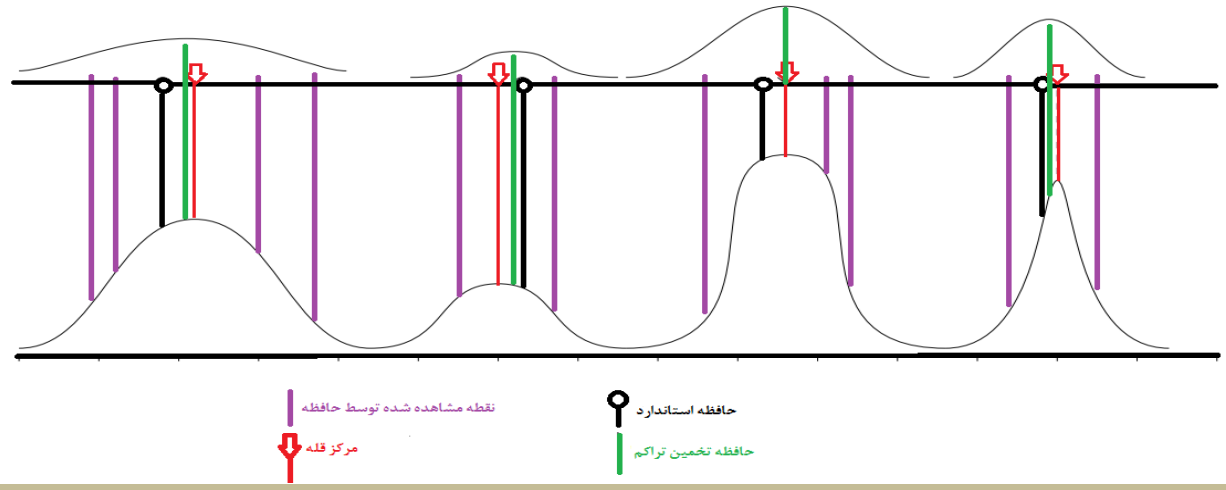
الگوریتم جستجو در راستای بهبود راه‌حل‌های بازیابی شده از حافظه عمل می‌کند، این حلقه بازخورد نیز در جهت بهبود مدل‌های ذخیره شده در حافظه گام برمی‌دارد. یک نوع از خوشه‌بندی برای حافظه‌های تخمین تراکم، خوشه‌بندی گاوسی[1] است، که مدل‌های تخمین تراکم غنی‌تری برای استفاده حافظه فراهم می‌کند. در این روش، هر مدخل حافظه یک مدل گاوسی از نقاط درون آن مدخل را محاسبه می‌کند. ماتریس میانگین و کواریانس برای محاسبه این‌که نقطه جدید به خوشه موجود متعلق است و یا احتمال این‌که دو خوشه می‌باست ادغام شوند، استفاده می‌شود.

۳،۱،۲. ذخیره‌سازی راه‌حل‌ها در حافظه‌ی تخمین تراکم

در حافظه استاندارد، یک نقطه اصولا با استفاده از راه‌کار جایگزینی در حافظه ذخیره می‌شود. شایان ذکر می‌باشد که این رویکرد وظیفه تصمیم‌گیری راجع به نحوه جایگزینی یک نقطه را درون یکی از مدخل‌های حافظه‌ی موجود بر عهده دارد. اکثر راه‌کارهای جایگزینی تنوع در حافظه را حفظ می‌کنند. به‌عنوان مثال، راه‌کار جایگزینی ابتدا یک نقطه جدید به حافظه اضافه می‌کند، سپس دو مدخل حافظه نزدیک به یکدیگر پیدا کرده و آن که از شایستگی کم‌تری برخوردار می‌باشد را حذف می‌کند. به‌غیر از مدخل‌های از بین رفته، حافظه‌ی تخمین تراکم مدخل‌ها را با هم ادغام می‌نماید. در وهله‌ی نخست، نقطه جدید اضافه شده به حافظه به دلیل ایجاد مدخل حافظه‌ی جدید، مورد استفاده قرار می‌گیرد. پس از این‌که مدل‌ها برای این مدخل جدید ساخته شد، دو مدخل که در حافظه بیشتر از همه به یکدیگر شبیه می‌باشند پیدا شده و ادغام می‌گردند. زمانی‌که فضای جستجوی جدیدی برای نخستین بار ساخته می‌شود، نقطه جدید نیز جهت آغاز مدل-سازی این ناحیه از حافظه مورد استفاده قرار خواهد گرفت. به‌واسطه افزایش نقاط خوشه‌بندی، ناحیه‌های جدید فضای جستجو را می‌توان بسته به تغییرات محیط پویا مدل‌سازی کرد. به‌محضی که خوشه‌ها ادغام شدند، خوشه‌ها متشکل از چندین

[1] Gaussian clustering

نقطه خواهند شد. همچنین، چنانچه تراکم آن نقاط افزایش یابد، یک مدل مناسبی نیز جهت برآورد مناسبی از مکان راه-حل‌های خوب در آن ناحیه ارائه خواهد شد. شایان ذکر است که حافظه‌ی تخمین تراکم همانند حافظه استاندارد شدیدا به الگوریتم بهینه‌سازی یا یادگیری وابسته است. حافظه‌ی تخمین تراکم اصولا مدل‌ها را بر مبنای نقاط ذخیره شده در حافظه می‌سازد، لذا اگر الگوریتم مزبور راه‌حل‌های ضعیفی را ذخیره کند، آن‌گاه مدل‌های موجود در حافظه راه‌حل خوبی را نمایش نخواهند داد. در صورتی‌که راه‌حل‌های خوبی از سوی الگوریتم معرفی گردد، آن‌گاه حافظه تخمین تراکم به الگوریتم کمک خواهد کرد تا ناحیه امیدبخش مورد نظر را کشف کند، از این رو قبل از آن الگوریتم باید بتواند به‌خوبی عملیات جستجو را در آن ناحیه انجام دهد. زمانی‌که نقاط جدید در حافظه ذخیره می‌شوند تا حد زیادی به الگوریتم‌های بهینه-سازی یا یادگیری و نوع مساله بستگی دارند. زمانی‌که از الگوریتم تکاملی استفاده می‌شود، نقطه جدید ممکن است در هر چند نسل ذخیره گردد. همچنین، موقعی‌که از الگوریتم یادگیری تقویتی بهره گرفته می‌شود، بهترین راه این است که نقاط جدید در زمان یاد گرفتن راه‌حل‌های مطلوب ذخیره شوند (در ذخیره نقاط جدید یک‌بار راه‌حل‌های خوب آموزش ببینند). در بعضی از مسائل هنگامی‌که تغییرات به‌وضوح قابل رویت باشد، بهترین حالت این است که راه‌حل‌ها قبل از اعمال تغییر ثبت شوند. در رابطه با مسائل دیگری که تغییرات در آن به‌صورت تدریجی رخ می‌دهند، بهتر است که نقاط به‌طور دوره‌ای ذخیره شوند. ذخیره‌سازی یک راه‌حل، درست بعد از اعمال تغییر اصلا کار مطلوبی نیست زیرا هیچ فرصتی برای تطبیق آن وجود ندارد.



شکل (۵): مقایسه حافظه تخمین تراکم گوسی با حافظه استاندارد

سربار محاسباتی استفاده از حافظه‌ی تخمین تراکم به مساله و مدل‌های احتمالاتی به‌کار گرفته شده در حافظه وابسته می‌باشد. زمانی‌که نقاط خوشه‌بندی افزایش می‌یابد، انتخاب دو مدخل برای ادغام می‌تواند در $O(m^2)$ انجام داد. در این حالت، m تعداد مدخل‌های حافظه تلقی می‌شود. وقتی از مدل‌های گاوسی درون حافظه‌ی تخمین تراکم استفاده می‌شود، کواریانس در $O(n^2)$ محاسبه می‌گردد؛ که n تعداد نقاط موجود در مدخل حافظه و d تعداد ابعاد موجود در داده‌های کنترلی یا محیطی می‌باشد. از آن جایی که حافظه به چندین خوشه تقسیم می‌شود، n عمدتا کم‌تر از تعداد کل نقاط موجود در حافظه انتخاب می‌گردد. اگرچه استفاده از حافظه مقداری سربار محاسباتی دارد، زمان لازم برای ارزیابی راه‌حل‌ها به‌طور معمول بر زمان مورد نیاز جهت نگهداری حافظه غالب است. از آن‌جایی‌که حافظه‌ی تخمین تراکم اساسا به ارزیابی راه‌حل‌های اضافی احتیاجی ندارد، در نتیجه سربار حافظه تخمین تراکم تنها به‌واسطه‌ی ساخت و نگهداری مدل‌های

احتمالاتی موجود در حافظه حاصل می‌شود. در بعضی از مسائلی که داده‌های کنترلی یا محیطی ابعاد بسیار زیادی دارند، ممکن است زمان زیادی جهت ساخت حافظه صرف شود و قبل از بازیابی از حافظه می‌تواند در جهت بهبود کارایی موثر باشد. هرچه ابعاد داده بیشتر باشد، نقاط بیشتری جهت ارائه‌ی تخمین تراکم خوب مورد نیاز است. در رابطه با این مسائل، بهتر است که ابتدا حافظه ساخته شود تا این‌که به‌صورت خالی باشد.

۳،۱،۳. بازیابی راه‌حل‌ها از حافظه‌ی تخمین تراکم

راه‌حل‌ها در حافظه‌ی تخمین تراکم شبیه به حافظه‌ی استاندارد بازیابی می‌شوند. پیرامون الگوریتم‌های جستجوی مبتنی بر جمعیت، راه‌حل‌ها ممکن است بلافاصله بعد از شناسایی تغییر در محیط بازیابی شوند. رایج است که راه‌حل‌ها در هر نسل از الگوریتم تکاملی از حافظه بازیابی می‌شوند. زمانی‌که مدخل‌های حافظه تنها به‌هنگام ذخیره‌سازی نقاط جدید به حافظه تغییر پیدا می‌کنند، راه‌حل‌های ارائه‌شده به‌واسطه‌ی بازیابی از حافظه می‌توانند در هر نسل به‌وسیله الگوریتم جستجو مورد استفاده قرار گیرند. برای الگوریتم‌های یادگیری که تنها یک نقطه را جستجو می‌کنند، راه‌حل‌ها می‌بایست از حافظه با تکرار کم‌تر و انتخاب بیشتر بازیابی شوند. از آن-جایی‌که بازیابی از حافظه در هر مرحله باعث می‌شود که الگوریتم راه‌حل مربوطه را اصلاح کند، راه‌حل‌ها یا به‌صورت دوره‌ای یا در هر نسل بازیابی می‌شوند. در مقاله حاضر، راه‌حل حاصل از حافظه زمانی بازیابی می‌گردد که راه‌حل جاری ضعیف عمل کرده و محیط کنونی نیز کاملا با یکی از مدخل‌های حافظه منطبق باشد. برخلاف حافظه‌ی استاندارد، راه‌حل‌های بازیابی شده از حافظه، چندان شباهتی با تک تک راه‌حل‌های ذخیره شده ندارند که این موضوع در واقع، ایجاد راه‌حل‌های متنوع در حافظه را تایید می‌کند. به‌واسطه‌ی تجمع بسیاری از نقاط موجود در مدخل حافظه، راه‌حل بازیابی شده از حافظه تحت تاثیر راه‌حل‌های قبلی که در محیط مشابه به خوبی عمل می‌کرده‌اند، قرار خواهد گرفت.

۳،۲. جهش مبتنی بر تپه شن سلولی

جهش مبتنی بر تپه شن دو-بعدی [۴۱] در روابط (۱۴) تا (۱۹) توصیف شده است. با این‌حال، یک مدل دو بعدی که از قوانین آتوماتای سلولی ساده پیروی می‌کند را می‌توان با این مدل دو-بعدی ایجاد نمود [۴۱]. در این ساختار ساده، مقدار موجود در سلول (x,y) با $Z(x,y)$ نشان داده می‌شود. قانون به‌روزرسانی بیان می‌کند که اگر با افزایش مقدار هر سلول(x,y)، مقدار آن از Z_C بیشتر شود، آن سلول به اندازه C واحد کاهش یافته و به‌ازای آن به تعداد C سلول مجاور (همسایگی ون-نیومن) یک واحد به مقدار هر کدام افزوده می‌شود. مثلا اگر $C=4$ باشد، سیستم توسط روابط زیر شرح داده می‌شود [۴۱].

$$Z(x,y) \rightarrow Z(x,y)+1 \qquad (۱۶)$$

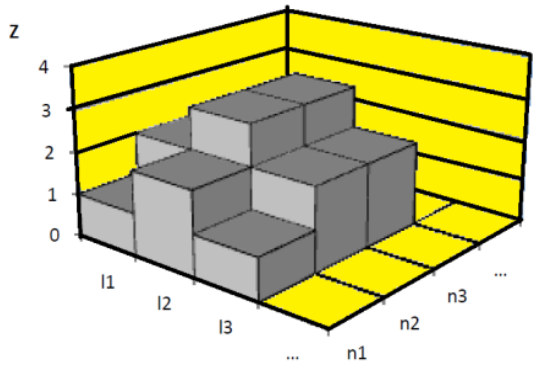
$$\text{if } Z(x,y) > Z_C:\ Z(x,y) \rightarrow Z(x,y)-4 \qquad (۱۷)$$

$$Z(x\pm 1,y) \rightarrow Z(x\pm 1,y)+1 \qquad (۱۸)$$

$$Z(x,y\pm 1) \rightarrow Z(x,y\pm 1)+1 \qquad (۱۹)$$

رابطه (۱۶)، افزودن یک دانه شن در یک سلول انتخاب‌شده تصادفی (x,y) را توصیف می‌کند. سپس، اگر تعداد دانه در آن سلول بیش از مقدار بحرانی باشد (به‌طور معمول، $Z_C=4$). در رابطه (۱۷) مشاهده می‌شود، سلول دانه‌های خود را از دست می‌دهد و آنها به همسایه‌هایش اضافه می‌شوند (روابط (۱۷) تا (۱۹)). سپس تپه شن توسط همان روابط به‌روز می-شود تا زمانی‌که سقوط متوقف شود. هنگامی‌که سقوط متوقف می‌شود، یک دانه دیگر کاهش یافته و این روند به‌طور مرتب تکرار می‌شود (روابط به‌صورت بازگشتی هستند). جهش تپه شن از این مدل دو-بعدی با اصلاحات جزئی به‌منظور تبدیل مقادیر جهش خودتنظیمی در الگوریتم ژنتیک استفاده می‌شود. ابتدا، یک شبکه $N \times L$ با $N=1,\dots n$ و $L=1,\dots l$ را در نظر بگیرید، به‌طوری‌که n اندازه جمعیت و l طول کروموزوم تعریف شده است. سپس جمعیت GA به این شبکه متصل‌شده است. به‌عنوان مثال اولین ژن از اولین کروموزوم با سلول (۱،۱) مرتبط است؛ ژن دوم از کروموزوم اول، به سلول (۱،۲)

مرتبط است. این وقوع در یک سلول شبکه، در ژن مرتبط با آن منعکس می‌شود. شکل (۶) رابطه بین تپه شن و جمعیت را نشان می‌دهد. جهش تپه شن به‌جای جهش سنتی در GA استاندارد جایگزین می‌شود. این در حالی است که، جهش سنتی در سطح بیت انجام می‌شود، یعنی جهش در هر بیت رخ می‌دهد، ولی روش جهش تپه شن در هر نسل انجام می‌شود. یک نسخه اولیه از این عملگر در [۴۱] ارائه شده است و شبه-کد آن در شکل (۷) نشان داده شده است. این روش به‌شرح ذیل کار می‌کند: پس از ایجاد نسل جدید توسط عملگرهای انتخاب و ادغام، آن‌ها بر اساس برازش ارزیابی و رتبه‌بندی می‌شوند. سپس همان‌طور که در بالا توضیح داده شد، هر فرد به یک $N \times L$ نگاشت می‌شود. تپه شن در نسل اول شروع به تکامل می‌کند (یک مقدار اولیه از شبکه به‌منظور جلوگیری از مرحله پیشین بدون رویداد اجرا می‌شود؛ در نسخه اولیه، مقداردهی اولیه توسط تنظیم هر $Z(x,y)$ به مقادیر تصادفی بین ۱ تا ۳ صورت می‌گیرد).



شکل (۶): رابطه بین تپه شن و جمعیت [۴۱]

همان‌گونه که در شکل (۶) مشاهده می‌شود، ژن‌های l_1, l_2, l_3 از کروموزوم n_1, n_2, n_3 هستند. در این شکل، $Z(l_2, n_2) = 3$، ارتفاع سلول l_2, n_2 می‌باشد. اگر یک دانه در آن سایت تخلیه شود، ممکن است سقوط رخ دهد. سپس، چهار دانه به محوطه سلول همسایگی ون-نویمن سرازیر خواهد شد و (l_2, n_3) به منطقه بحرانی Z خواهد رسید. اگر شرایط مطلوب باشد، سقوط و جهش ممکن است ادامه یابد. در هر نسل دانه‌ها به-طور تصادفی روی شبکه می‌افتند، به این‌ترتیب ارزش سلول بتدریج افزایش می‌یابد. زمانی‌که یک سلول به مقدار بحرانی Z_c می‌رسد، یک سقوط رخ می‌دهد (تنها درصورتی‌که یک مقدار به‌طور تصادفی از یک توزیع بین صفر و یک، بالاتر از برازش نرمال (رابطه (۲۳)) فرد مرتبط با سلولی باشد که به مقدار بحرانی رسیده است). در این روش، افرادی با برازش بیشتر شانس کم‌تری برای جهش نسبت به ژن‌های ضعیف‌تر دارند. دقت شود که رتبه‌بندی برای هر نسل انجام می‌شود و شبکه را مرتب می‌کند، یعنی زمانی‌که در برخی از مناطق شبکه راه‌حل‌های خوب وجود دارد، سقوط تمایل به متوقف شدن دارد. پس از اولین سقوط، سلول‌های همسایه به‌صورت بازگشتی به‌روز می‌شوند و سقوط ممکن است در شبکه ادامه یابد تا زمانی‌که حالت تپه شن و برازش راه‌حل‌ها برای پیشرفت مطلوب است. آستانه Z_c برابر با ۴ است، که برابر با تعداد دانه‌هایی است که اگر از یک سلول سقوط رخ دهد، سرازیر می‌شوند. شبه‌کد جهش پیشنهادی در شکل (۷) نشان داده شده است.

Algorithm 2: the mutation operator

for g grains do
drop grain at random within the bounds of the lattice $Z(x,y) \rightarrow Z(x,y) + 1$
compute normalized fitness:

$$f_n(j) = \frac{fitness(j) - worstFitness}{bestFitness - worstFitness}$$

where j is the index of the chromosome associated with the cell (x,y),
worstFitness is the lowest fitness in the current population and bestFitness is the best fitness in the population.
Note:
if $bestFitness = worstFitness, f_n(j)$ *is set to* 0.5, *independently of* j.
if $randomValue(0,1.0) > f_n$ *and cell* (x,y) *not active*
mutate *(bit flip mutation)*
avalanche
$Z(x,y) \rightarrow Z(x,y) - 4$
$Z(x \pm 1, y) \rightarrow Z(x \pm 1, y) + 1$
$Z(x, y \pm 1) \rightarrow Z(x, y \pm 1) + 1$
and ***update*** *lattice Z* ***recursively***

شکل(۷): شبه‌کد جهش دو-بعدی مبتنی بر تپه شن نسخه اول [۴۲]

f_n میزان برازش نرمال‌شده مرتبط با سلول (x,y) را تعیین می‌کند. j اندیس کروموزوم مرتبط با سلول (x,y) را نشان می‌دهد. در اینجا مساله بحرانی مقدار $f_n(j)$ می‌باشد. دقت شود که در مدل تپه شنی، کروموزوم‌ها قبل از جهش ارزیابی می‌شوند. این تنها راه برای اطمینان از این است که مقدار برازش بر جهش تأثیر می‌گذارد. با این‌حال، با انجام این کار،

مرحله انتخاب بعدی با مقادیری که با ژنوتیپ فعلی سازگار نیستند، کار می‌کند. یک راه‌حل ممکن این است که بعد از مرحله جهش، مجدد کل جمعیت ارزیابی شود، اما این کار پیچیدگی محاسباتی را در هر نسل دو برابر می‌کند.

$worstFitness$ بدترین میزان برازش برای کروموزوم و $bestFitness$ بهترین میزان برازش را برای یک کروموزوم در جمعیت نشان می‌دهد. جهش تپه شن دارای یک محدودیت است که در مدل اصلی وجود ندارد: اگر یک سلول در حال حاضر در یک سقوط (سلول فعال) وجود داشته باشد و ماهیت بازگشتی فرآیند اجازه تکمیل شدن نداشته باشد، آن‌گاه سلول‌ها نادیده گرفته می‌شوند. به‌عبارت دیگر، اگر یک سلول، دانه به سایت‌های همسایه خود بدهد، پس از آن نمی‌تواند دانه‌ای از همان سایت‌ها بگیرد. این محدودیت چرخه‌های سقوط فرضی و چندین جهش از یک ژن را حذف می‌کند، اگرچه هیچ شواهد تجربی مبنی بر این که این محدودیت عملکرد مدل را بهبود می‌بخشد، وجود ندارد (به‌منظور کاهش زمان محاسبات از روند جهش اتخاذ می‌شود). دو مورد دیگر باید بررسی شود: اول، اگر یک مقدار Z به Z_C برسد اما جهش نکند (به‌علت آزمایش برازش) سپس دانه دور انداخته می‌شود. علاوه‌براین، اگر سقوط در یک سلول در لبه شبکه رخ دهد، پس از آن دانه‌ها شبکه را ترک می‌کنند (در واقع، این چیزی است که در مدل تپه شن اتفاق می‌افتد). نکته نهایی که باید اشاره شود، جهش‌های مربوط به برازش می‌باشد. دقت شود که کروموزوم‌ها قبل از این‌که توسط تپه شن مورد آزمایش قرار گیرند، رتبه‌بندی می‌شوند. این تنها راه برای ایجاد مقادیر برازش موثر بر جهش است، اما مرحله انتخاب بعدی که بر روی مقادیر برازش کار می‌کند مطابق با ژنوتیپ کنونی نیست. یک راه‌حل ممکن ارزیابی مجدد کل جمعیت پس از مرحله جهش است، اما این گزینه تلاش محاسباتی را دو برابر خواهد کرد. اگر چه نتایج منتشر شده در [۴۲] امیدوارکننده بودند، اما آزمایش‌های بیشتر نشان داد که کارایی الگوریتم در مقایسه با سایر GAها به‌همان اندازه که انتظار می‌رفت پس از آزمایش‌های اول، واضح نبود. سپس یک نسخه اصلاح‌شده مورد آزمایش قرار گرفت که به‌طور مداوم اولین نسخه ارائه شده را بهبود بخشید. شبه‌کد نسخه اصلاح‌شده به‌صورت شکل (۸) نشان داده شده است.

Algorithm 3: *The Reconfiguration Mutation Operator*

for g grains do
drop grain at random within the bounds of the lattice $Z(x,y) \rightarrow Z(x,y) + 1$
compute normalized fitness:

$$f_n(j) = \frac{parentsFitness(j) - bestFitness}{worstFitness - bestFitness}$$

where j is the index of the chromosome associated with the cell (x,y),
$parentsFitness(j)$*is the average fitness of chromosome j parents,*
worstFitness is the lowest fitness in parents population and bestFitness is the best fitness in parents population.
Note:
if $bestFitness = worstFitness, f_n(j)$ *is set to* 1.0, *independently of* j.
if $randomValue(0,1.0) > f_n$ *and cell* (x,y) *not active*
mutate (*flips the bit with probability* 0.5)
avalanche
$Z(x,y) \rightarrow Z(x,y) - 4$
$Z(x \pm 1, y) \rightarrow Z(x \pm 1, y) + 1$
$Z(x, y \pm 1) \rightarrow Z(x, y \pm 1) + 1$
and ***update*** *lattice Z* ***recursively***

شکل(۸): شبه‌کد جهش خودتنظیم‌شونده مبتنی بر تپه شن نسخه دوم [۴۲]

در شبه‌کد بالا (شکل (۸)) $parentsFitness(j)$، میانگین برازش برای والدین کروموزوم j می‌باشد. به این‌ترتیب، جهش تپه شنی درست بعد از جمعیت جدید تولیدشده توسط عملگر انتخاب و تقاطع و همچنین قبل از ارزیابی جمعیت، اعمال می‌شود. از آن‌جایی‌که والدین مهربان شانس بیشتری برای تولید فرزندان صاحب فرزند دارند، این رویکرد ممکن است تقریب خوبی برای ایده اصلی باشد. طرح دیگری در مقاله [۴۳] طراحی و آزمایش شده است. رابطه (۲۵) تابع برازش $f_n(j)$ را تعریف می‌کند، به‌طوری که متوسط برازش و همگرایی جمعیت، جهش را کنترل می‌کنند. هنگامی‌که میانگین برازش نزدیک به بهترین برازش در جمعیت باشد، تابع f_n تمایل به افزایش دارد. در نتیجه، احتمال این‌که یک بهمن به‌وسیله آرایه گسترش یابد و کروموزوم‌ها را تغییر دهد نیز افزایش می‌یابد.

$$f_n(j) = 1 - \frac{averageFitness - bestFitness}{worstFitness - bestFitness} \quad (۲۰)$$

در رابطه (۲۰)، علاوه بر میانگین برازش و مرحله همگرایی جمعیت، وضعیت تپه شنی نیز بر شدت جهش در یک نسل

خاص تاثیر می‌گذارد. نرخ جهش نهایی محصول یک تعامل پیچیده بین این عوامل است [۴۳]. در روش [۴۳] هنگامی‌که $bestFitness = worstFitness$، GA ممکن است به‌طور کامل همگرا باشد، رابطه (۲۵) معتبر نمی‌باشد و بنابراین بهتر است که $f_n(j) = 1$ را برای همه j تنظیم کنیم، به این‌معنی که، جهش مطمئنا بعد از بهمن رخ می‌دهد. به این‌ترتیب، مدل تپه شنی برای جهش‌های بزرگ "باز" است. شبه‌کد روش پیشنهادی مقاله [۴۳] در شکل (۹) آورده شده است.

Algorithm 4: *The Mutation Operator*

for g grains do
drop grain at random within the bounds of the lattice $Z(x,y) \rightarrow Z(x,y) + 1$
compute normalized fitness:

$$f_n(j) = 1 - \frac{averageFitness - bestFitness}{worstFitness - bestFitness}$$

where j is the index of the chromosome associated with the cell (x,y),
averageFitness is the average fitness of bestFitness
worstFitness is the lowest fitness in parents population and bestFitness is the best fitness in parents population.
Note:
if $bestFitness = worstFitness, f_n(j)$ *is set to* 1.0, *independently of* j.
if $randomValue(0,1.0) > f_n$ *and cell* (x,y) *not active*
mutate (*flips the bit with probability* 0.5)
avalanche
$Z(x,y) \rightarrow Z(x,y) - 4$
$Z(x \pm 1, y) \rightarrow Z(x \pm 1, y) + 1$
$Z(x, y \pm 1) \rightarrow Z(x, y \pm 1) + 1$
and ***update*** *lattice Z* ***recursively***

شکل(۹): شبه‌کد جهش خودتنظیم‌شونده مبتنی بر تپه شن نسخه پیشنهادی در [۴۳]

۳،۳. ترکیب حافظه پیشنهادی با جهش خودسازمانده بحرانی در الگوریتم ژنتیک

در این بخش، راهکار حافظه تخمین تراکم گوسی پیشنهادی با الگوریتم ارائه شده در مقاله [۴۳] ترکیب می‌شوند تا این الگوریتم بتواند به نحو مطلوبی بهینه (بهینه‌های) جابه‌جا شده را در محیط ردیابی نماید. دقت شود دو راهکار وقتی در کنار الگوریتم ژنتیک قرار گیرند موجب می‌شوند که تنوع (هدف اصلی این مقاله) در حین اجرای الگوریتم افزایش یافته و این موضوع باعث می‌شود که الگوریتم در مواجهه با مسائل بهینه-سازی پویا واکنش مناسبی از خود نشان دهد. شبه‌کد، الگوریتم پیشنهادی در نگاه کلی در شکل (۱۰) آورده شده است. همچنین روندنمای روش پیشنهادی در شکل (۱۱) نیز آورده شده است.

Algorithm 5: ***proposed method***

1. $pop\_size = POP\_MAX$
2. $mem\_size = MEM\_MIN$
3. *Initialize Memory Randomly*
4. *Initialize Population Randomly*
5. $TM = rand(5,10)$
6. $t = 0$;
7. ***repeat***
8. *Evaluate memory*
9. *Evaluate population*
10. *Select mating pool*
11. *Recombination*
12. *Mutation*
13. ***if*** *is time to update memory* ***then***
14. $TM = t + rand(5,10)$
15. *Select best individual of the population*
16. ***if*** *there is room to one more individual in memory* ***then***
17. *Store best individual in memory*
18. *Increase mem_size*
19. ***else***
20. ***if*** *cleanMemory is successful* ***then***
21. *Update mem_size*
22. *Store best individual in memory*
23. ***else***
24. *Replace individual of memory according the replacing scheme*
25. ***if*** *change is detected* ***then***
26. *Select best individual from memory*
27. ***if*** *there is room to one more individual in population* ***then***
28. *Store individual in population*
29. *Increase pop_size*
30. ***else***
31. ***if*** *cleanMemory is successful* ***then***
32. *Update mem_size*
33. *Store individual in population*
34. ***else***
35. *Replace worst individual of population*
36. ***until*** *stop_condition*

شکل (۱۰): شبه‌کد الگوریتم پیشنهادی

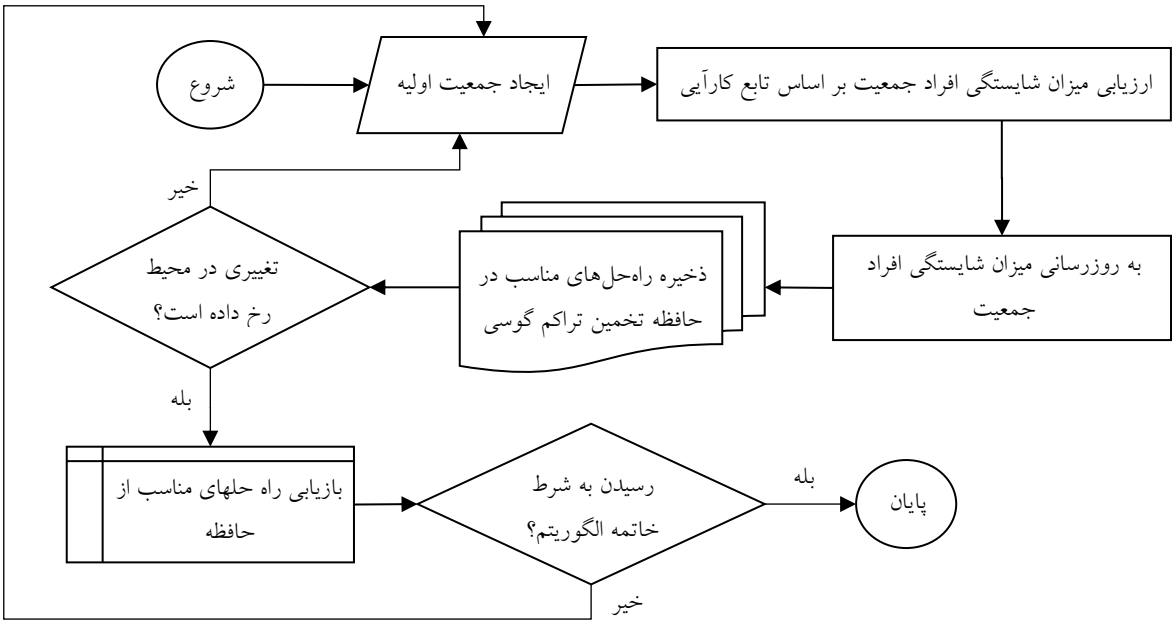
۴. آزمایش‌ها و نتایج تجربی

برای انجام آزمایشات بر روی الگوریتم پیشنهادی و مقایسه آن با سایر الگوریتم‌ها در یک محیط پویا از تابع محک پویا برای ارزیابی کارآیی الگوریتم پیشنهادی استفاده می‌نماییم. در تمامی

آزمایشات سعی شده تا الگوریتم‌ها در شرایط یکسان با هم مقایسه شوند. تعداد اجراهای الگوریتم ۳۰ بار بوده و در هر اجرا، الگوریتم ۱۰۰ بار تکرار می‌شود. همچنین تمام آزمایش‌ها بر روی الگوریتم‌ها در شرایط مشابه صورت گرفته است. آزمایشات در محیط نرم‌افزار MATLAB و سیستم‌عامل ویندوز ۷ با پردازنده $Intel\ CORE\ i7$ انجام شده است.

آزمایش‌ها به‌گونه‌ای طراحی شده‌اند که بتوانیم کارایی الگوریتم‌های مختلف را بر روی تولیدکننده محیط پویا بررسی کنیم. در الگوریتم‌های حافظه‌دار اندازه حافظه شامل $m = 0.1 \times 100 = 10$ عضو برای حافظه می‌باشد (در اجرای هر الگوریتم ۱۰۰ تغییر محیط را در نظر گرفته‌ایم).

در این بخش از عملگر جهش پیشنهادی برای سایر روش‌های مبتنی بر ژنتیک استفاده نموده و نتایج به‌دست آمده با روش‌های پیشین مقایسه می‌شود. در این آزمایشات نشان داده خواهد شد که روش جهش پیشنهادی توانسته به کرات دیگر الگوریتم‌های مبتنی بر GA را بهبود بخشد. در نهایت نتایج حاصل از ترکیب دو راهکار پیشنهادی (حافظه و جهش) با روش‌های مشابه جدید از جمله روش ارائه شده در [۲۲] مقایسه می‌شود. معیار ارزیابی استفاده شده برای مقایسه الگوریتم‌ها، معیار کارآیی برون‌خطی می‌باشد که در ادامه این معیار تشریح شده است.



شکل (۱۱): روندنمای روش پیشنهادی از دید کلی

۴،۱. معیار ارزیابی کارآیی

کارایی کلی الگوریتم را با کارآیی برون‌خطی محاسبه کرده‌ایم. کارآیی برون‌خطی از رابطه زیر محاسبه می‌شود [۴۴ و ۴۵].

$$\bar{F}_{BOG}(X) = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N} F_{BOG_{ij}}\right) \qquad (۲۱)$$

$$G = 100 \times \tau$$

در این رابطه G برابر تعداد نسل، $F_{BOG_{ij}}$ برابر برازندگی بهترین فرد در نسل i و اجرای j می‌باشد. هر چه میزان کارآیی برون‌خطی برای الگوریتمی بیشتر باشد، نشان از برتری آن الگوریتم نسبت به سایر الگوریتم‌ها دارد. نتایج آزمایش‌های صورت گرفته با روش فوق در جداول (۱) تا (۴) آمده است. برای به‌دست آوردن نتایج هر الگوریتم را ۳۰ بار به‌صورت مستقل با مقداردهی یکسان انجام داده‌ایم. در ضمن برای مقایسه آماری الگوریتم‌ها از آزمون t استفاده شده است. در این آزمون ما از ۵۸ درجه آزادی با سطح معنی‌دار برابر با ۰/۰۵ استفاده کرده و

نوع آزمون را paired و one-tailed قرار داده‌ایم [۳۹]. بعد از انجام آزمایش نتایج بدین‌صورت نمایش داده می‌شوند که اگر الگوریتم اول از الگوریتم دوم به‌صورت معناداری بهتر باشد از علامت + استفاده شده است و اگر الگوریتم اول از الگوریتم دوم به‌صورت معناداری بدتر باشد از علامت − استفاده شده است. اگر شواهد لازم برای معناداری هیچ کدام از الگوریتم‌ها پیدا نشود از علامت ~ استفاده شده است.

۴،۲. ایجادکننده محیط پویا

محیط پویا که برای ارزیابی کارایی الگوریتم استفاده شده، تولیدکننده‌ی XOR یانگ [۲۰] می‌باشد. با این تولیدکننده این توانایی وجود دارد تا هر تابع ایستای $f(\vec{x})$ با کدینگ باینری، $\vec{x} \in \{0,1\}^l$ را به یک مساله پویا تبدیل شود. ایده کار این‌است که در هر دوره‌ی k قبل از ارزیابی $\vec{x}$ ابتدا $\vec{x} \oplus \vec{M}$ محاسبه می‌شود که $\vec{M}$ بردار ماسک می‌باشد:

$$M(k) = M(k-1) \oplus T(k), \qquad M(1) = 0 \qquad (۲۲)$$

$T(k)$ بردار موقت میانی است که دارای $\rho \times l$ یک می‌باشد و برای نسل t، $k = [\, t/\tau]$ است. با این تولیدکننده می‌توان به سادگی هر مساله با پیچیدگی دلخواه (به شرط دامنه ۰ و ۱) را با دو پارامتر فرکانس تغییرات و شدت تغییرات (τ و ρ) تبدیل به مساله‌ای پویا کرد. $\rho \in [0,1]$ شدت تغییرات را کنترل می‌کند، به این‌ترتیب که هر چه مقدارش بیشتر باشد، تغییرات محیط از دوره‌ای به دوره‌ی دیگر بیشتر خواهد بود. τ بیان می‌کند بعد از چند نسل محیط تغییر خواهد یافت (فرکانس تغییرات).

برای جلوگیری از ابهام در نتایج به‌دست آمده در این مقاله از تعداد ارزیابی‌ها ($\varepsilon = \tau \times n$) استفاده شده است. n اندازه جمعیت در GA می‌باشد. GA ها ممکن است جمعیتی از هر اندازه و استراتژی‌های مختلف انتخاب و جایگزینی داشته باشند، اندازه‌گیری دوره از لحاظ نسل‌ها اطلاعات کافی در مورد دوره تغییرات ارائه نمی‌دهد، مگر این‌که تعداد جمعیت و تعداد افراد ایجادشده در هر نسل مشخص شود.

توابع استفاده شده در مقاله که به‌عنوان پایه برای تولید توابع پویا هستند به‌ترتیب، Knapsack dynamic، Royal road و Deceptive می‌باشند. تابع اول Knapsack dynamic می‌باشد. کوله پشتی ۰-۱ پویا یکی از اولین مسائلی است که جهت ارزیابی کارایی الگوریتم‌های تکاملی روی محیط‌های غیرایستا استفاده شده است. مساله کوله پشتی ۰-۱ به عنوان یک مساله ترکیبی NP–کامل شناخته شده است [۴۷]. یک مجموعه از m وارده ($m = 100$) با وزن‌های مختلف ($\vec{w}$) و منافع ($\vec{p}$) مشخص و الگوریتم می‌بایست سود $p(\vec{x})$ را افزایش دهد [۴۷]. این تابع از رابطه (۲۲)، محاسبه می‌شود [۴۷]:

$$f(x) = \begin{cases} \sum_{i=1}^{100} p_i x_i \quad if \sum_{i=1}^{100} w_i x_i \le C \\ 10^{-10} \times (\sum_{i=1}^{100} w_i - \sum_{i=1}^{100} w_i x_i), \\ \qquad else \end{cases} \qquad (۲۳)$$

که $\vec{x} = (x_1 \ldots x_m)$ و x_i نیز صفر یا یک است و C ظرفیت کوله پشتی است. همچنین این تابع مشمول محدودیت زیر است:

$$\sum_{i=1}^{m} w_i x_i \le C \qquad (۲۴)$$

تابع دوم Royal Road می‌باشد، هدف در این تابع جفت کردن حداکثر تعداد بسته‌های ۴-تایی یک در کنار هم می‌باشد.

$$f(x) = \sum_{i=1}^{\frac{l}{4}} 4 \times \sigma_l(x_{i \ldots i+4}) \qquad (۲۵)$$
$$l = \text{chromosome length}$$
$$\sigma(x) = \{1, \text{if } x = 1111; 0, \text{otherwise}\}$$

حداکثر مقدار تابع هدف برابر l می‌باشد و به‌دلیل دارا بودن کمینه محلی زیاد، حل این مساله برای الگوریتم ژنتیک سخت است. تابع سوم Deceptive (تابع تله) می‌باشد. یک تابع تله یک تابع خطی تکه‌ای است که در یک واحد تعریف شده است که دارای دو ناحیه متمایز در فضای جستجو است، یکی منجر به بهینه سراسری و دیگری منجر به بهینه محلی می‌شود. به‌طور خاص، تابع تله (order-k) می‌تواند به‌صورت توسط زابطه (۲۶) بیان شود.

$$trap(u(\vec{x})) = \begin{cases} \frac{a}{z}(z-u(\vec{x})) & \text{if } u(\vec{x}) < z \\ \frac{b}{k-z}(u(\vec{x})-z) & \text{otherwise} \end{cases} \quad (۲۶)$$

در رابطه (۲۶)، $u(\vec{x})$ تابع متحد (تعداد ۱ها در یک رشته)، a بهینه محلی، b بهینه سراسری، k اندازه مساله و z موقعیت شیب-تغییر است.

بنا به تنظیمات پارامترها، توابع تله ممکن است (نیست) فریب‌آمیز[1] باشد. این تابع جزو توابع بسیار سخت برای حل کردن مسائل پویا می‌باشد، زیرا دارای اجزای ساختاری از مرتبه پایین می‌باشد که با به‌هم پیوستن آن‌ها جواب بهینه به دست نمی‌آید. ساختاری از مرتبه پایین ممکن است جستجو را به‌سمت بهینه محلی گمراه کند، بنابراین راهکارهای جستجوی GA را به چالش می‌کشد. برای عملکرد تله فریبنده، نرخ r بین بهینه محلی (a) و بهینه سراسری (b) باید به صورت رابطه (۲۷) باشد.

$$r \geq \left(2-\frac{1}{k-z}\right)/\left(2-\frac{1}{z}\right) \quad (۲۷)$$

برای آزمایشات در این مقاله، برای توابع تله از $\text{Order}-4$ با استفاده از پارامترهای زیر استفاده شده است:

$$a = k-1;\ b = k;\ z = k-1$$

معیار ارزیابی دیگری که در این مقاله از آن استفاده می‌شود، معیار تنوع است. همان‌طور که قبلا نیز ذکر گردید، هدف اصلی این مقاله ارائه راه‌کارهایی جهت افزایش تنوع ژنتیکی است. معیار تنوع از رابطه (۲۸) به‌دست می‌آید [۴۶]:

$$d(P) = \frac{\sum_{i=1}^{l} \min(F_i, 1-F_i)}{\frac{l}{2}} \quad (۲۸)$$

در رابطه (۲۸)، l طول کروموزوم می‌باشد و همچنین اگر n اندازه جمعیت باشد آن‌گاه $F_i = \sum_{j=1}^{n} P_i(j)$ می‌باشد که $P_i(j)$ ژن j-ام از کروموزوم i-ام است [۴۶].

در انجام آزمایش‌ها برای شبیه‌سازی شرایط مختلف، τ برابر ۵۰، ۱۰ و ۱۰۰ نسل قرار داده شده است. به‌همین ترتیب برای بررسی کارایی الگوریتم‌ها در شرایط مختلف مقدار ρ برابر با ۰/۵، ۰/۲، ۰/۱ و ۱ در نظر گرفته شده است و برای بررسی کارایی در محیط‌های تصادفی، عددی تصادفی از بازه (۰/۹۹،۰/۰۱) انتخاب شده است. با احتساب این مقادیر برای هر تابع ۱۵ محیط مختلف شبیه‌سازی شده است و چون سه تابع که در مجموع ۴۵ آزمایش مختلف برای بررسی کارایی الگوریتم‌ها ترتیب داده شده است. توجه شود که هدف تمامی توابع محک ذکر شده در این مقاله بیشینه‌سازی این توابع می‌باشد.

باید دقت شود که الگوریتم ژنتیک دارای پارمترهای مختلفی از جمله، اندازه جمعیت، نرخ تقطیع و نرخ جهش می‌باشد که باید برای آزمایش‌های مختلف و مقایسه با دیگر روش‌ها مورد توجه خاصی قرار گیرند. برای مقایسه در شرایط یکسان در هر آزمایش این سه پارامتر برای همه روش‌ها یکسان در نظر گرفته می‌شود (مقادیر پارامترها در هر آزمایش ذکر می‌گردد). دقت شود که اندازه جمعیت (n) یکی از مسائل مهم است. جمعیت‌های کوچک مانع همگرایی به بهینه سراسری می‌شوند، اما جمعیت‌های بزرگ ممکن است سرعت همگرایی را کاهش دهند. علاوه‌براین، محیط‌های پویا نیز محدودیت دیگری برای اندازه جمعیت ایجاد می‌کنند: n باید از مقدار ε کوچک‌تر باشد (در عمل باید حداقل یک مرتبه از مقدار ε کوچک‌تر باشد) [۴۶]. مقدار نرخ جهش (p_m) در الگوریتم‌های ژنتیک که برای مسائل بهینه‌سازی پویا طراحی شده‌اند بسیار مهم می‌باشد [۴۶]. برای مثال اگر نرخ جهش روی $1/l$ تنظیم شود، در آن‌صورت برای مسائل بهینه‌سازی با ماهیت پویا، الگوریتم نیازمند ایجاد تعادل متفاوتی میان اکتشاف و بهره‌وری می‌باشد [۴۶]. در این بخش ابتدا از جهش پیشنهادی برای دو روش معروف EIGA و SORIGA [۴۶] استفاده شده و نشان داده می‌شود که آن توانسته این دو روش را نسبت به قبل بهبود دهد. برای این‌که مقایسه در شرایط کاملا یکسانی انجام گیرد، مطابق مرجع [۴۶] نرخ جهش مابین $1/16 \times l$ تا $2/l$ (l طول کروموزوم می‌باشد) در نظر گرفته شده و همچنین اندازه جمعیت ۳۰، ۶۰ و ۱۲۰ در نظر گرفته شده است. در این آزمایش‌ها مطابق [۴۶]، g در محدوده $(n \times l)/32$ تا $n \times l$

[1] Deceptive

تنظیم شده است (n اندازه جمعیت و l طول کروموزوم است). در این تحقیق مطابق با مرجع [۴۶]، عملگر جفت‌گزینی دو نقطه‌ای با احتمال ۱٫۰ تنظیم شده است. در این بخش، جهش پیشنهادی با سه روش مطرح شده در [۴۲] و [۴۳] از نظر میزان کارآیی برون‌خطی مقایسه شده است.

جدول (۱)، مقایسه روش پیشنهادی با سه روش مطرح شده در [۴۲] و [۴۳]، بر روی تابع Knapsack پویا با پارمترهای ذکر شده در همین جدول را نشان می‌دهد. جدول (۲)، مقایسه روش پیشنهادی با سه روش مطرح شده در [۴۲] و [۴۳]، بر روی تابع Order-4 trap با پارمترهای ذکر شده در همین جدول را نشان می‌دهد. جدول (۳)، مقایسه روش پیشنهادی با سه روش مطرح شده در [۴۲] و [۴۳]، بر روی تابع Royal Road با پارمترهای ذکر شده در همین جدول را نشان می‌دهد. جدول (۴) هر چهار روش را از نظر آماری (آزمون آماری t زوجی) با هم مقایسه نموده است. جداول (۱) تا (۳) نتایج ارزیابی روش پیشنهادی را در مقایسه با ۳ روش رقیب نشان داده‌اند. برای اثبات این‌که این مقادیر عددی تصادفی نبوده‌اند، از آزمون آماری معناداری استفاده شده است.

آزمون‌های آماری Kolmogorov-Smirnov غیرپارامتریک و بدون توزیع هستند. نتایج عددی استراتژی ۲ با استفاده از آزمونهای زوجی paired Kolmogorov-Smirnov با سطح معنی‌داری ۰/۰۵ و درجه آزادی ۹۸ با دو مقدار دیگر مقایسه می‌شود بعد از انجام آزمایش نتایج بدین‌صورت نمایش داده می‌شوند که اگر الگوریتم اول از الگوریتم دوم به‌صورت معناداری بهتر باشد از علامت + استفاده شده است؛ اگر الگوریتم اول از الگوریتم دوم به‌صورت معناداری بدتر باشد از علامت − استفاده شده است؛ اگر شواهد لازم برای معناداری هیچ کدام از الگوریتم‌ها پیدا نشود از علامت ~ استفاده شده است (تعداد جمعیت ۱۰۰، ۱۰۰ بار اجرا و در هر اجرا ۳۰ بار ارزیابی برای تعیین میزان شایستگی در تابع کارآیی انجام می‌گیرد) [۴۸و۵۱]. نتایج به‌دست آمده برای روش‌های رقیب از مرجع [۴۶] آورده شده‌اند.

جدول (۱): مقایسه روش پیشنهادی و سایر روش‌ها از نظر کارآیی برون‌خطی و انحراف استاندارد بر روی تابع KNAPSACK و اندازه جمعیت ۳۰

	$\varepsilon = 1200$				$\varepsilon = 4800$				$\varepsilon = 12000$			
$\rho \rightarrow$	0.05	0.3	0.6	0.95	0.05	0.3	0.6	0.95	0.05	0.3	0.6	0.95
$Strategy1$ [۴۲]	1808.2 ± 1.27	1797.9 ± 0.87	1792.3 ± 0.80	1788.1 ± 0.91	1821.97 ± 1.27	1812.3 ± 0.95	1808.2 ± 0.70	1806.1 ± 0.63	1828.0 ± 0.80	1820.7 ± 0.71	1818.2 ± 0.59	1817.1 ± 0.64
$Strategy2$ [۴۲]	1802.9 ± 0.98	1796.9 ± 0.71	1791.3 ± 0.93	1782.7 ± 0.92	1828.3 ± 0.91	1814.8 ± 0.75	1804.3 ± 0.74	1792.9 ± 0.58	1833.9 ± 0.77	1824.4 ± 0.53	1818.4 ± 0.43	1811.8 ± 0.49
$Strategy3$ [۴۳]	1801.0 ± 0.92	1796.3 ± 0.76	30.09 ± 0.21	1785.3 ± 0.77	1825.4 ± 0.85	33.64 ± 0.31	1806.2 ± 0.65	1797.4 ± 0.50	1831.3 ± 0.69	1823.2 ± 0.52	1818.4 ± 0.51	1813.8 ± 0.49
Proposed	**1809.8 ± 0.96**	**1798.8 ± 0.88**	**1792.0 ± 0.73**	**1789.3 ± 0.98**	**1828.9 ± 0.96**	**1814.3 ± 0.82**	**1809.4 ± 0.79**	**1798.4 ± 0.56**	**1834.2 ± 0.85**	**1825.1 ± 0.64**	**1818.9 ± 0.55**	**1818.6 ± 0.76**

جدول (۲): مقایسه روش پیشنهادی و سایر روش‌ها از نظر کارآیی برون‌خطی و انحراف استاندارد بر روی تابع ORDER-4 TRAP، و اندازه جمعیت ۳۰

	$\varepsilon = 1200$				$\varepsilon = 4800$				$\varepsilon = 12000$			
$\rho \rightarrow$	0.05	0.3	0.6	0.95	0.05	0.3	0.6	0.95	0.05	0.3	0.6	0.95
$Strategy1$ [۴۲]	32.10 ± 0.72	30.01 ± 0.26	30.13 ± 0.16	33.90 ± 0.11	36.01 ± 0.48	32.25 ± 0.26	32.43 ± 0.16	34.65 ± 0.05	37.89 ± 0.27	34.04 ± 0.23	34.85 ± 0.06	34.86 ± 0.07
$Strategy2$ [۴۲]	34.65 ± 0.89	29.92 ± 0.31	29.91 ± 0.24	34.40 ± 0.07	37.87 ± 0.38	33.51 ± 0.33	33.19 ± 0.17	34.82 ± 0.05	39.01 ± 0.18	35.57 ± 0.28	34.85 ± 0.06	35.00 ± 0.07
$Strategy3$[۴۳]	34.94 ± 0.74	30.17 ± 0.25	30.09 ± 0.21	34.26 ± 0.08	37.58 ± 0.38	33.64 ± 0.31	33.31 ± 0.17	34.65 ± 0.06	38.75 ± 0.19	35.64 ± 0.23	34.37 ± 0.15	35.02 ± 0.09
Proposed	**35.24 ± 0.76**	**32.70 ± 0.32**	**31.94 ± 0.23**	**35.84 ± 0.19**	**38.24 ± 0.46**	**34.81 ± 0.45**	**34.28 ± 0.23**	**35.15 ± 0.10**	**39.93 ± 0.21**	**36.02 ± 0.24**	**35.06 ± 0.11**	**35.89 ± 0.12**

جدول (۳): مقایسه روش پیشنهادی و سایر روش‌ها از نظر کارآیی برون‌خطی و انحراف استاندارد بر روی تابع ROYAL ROAD، و اندازه جمعیت برابر ۳۰

	$\varepsilon = 1200$				$\varepsilon = 4800$				$\varepsilon = 12000$			
$\rho \rightarrow$	0.05	0.3	0.6	0.95	0.05	0.3	0.6	0.95	0.05	0.3	0.6	0.95
$Strategy1$ [۴۲]	35.91 ± 2.93	17.88 ± 0.68	14.93 ± 0.52	15.86 ± 0.66	52.26 ± 2.76	29.84 ± 0.96	24.21 ± 0.59	23.31 ± 0.70	57.34 ± 1.97	36.97 ± 1.04	31.60 ± 0.68	28.15 ± 0.45
$Strategy2$ [۴۲]	33.43 ± 1.61	18.84 ± 0.59	15.22 ± 0.48	15.93 ± 0.29	52.14 ± 2.25	32.36 ± 1.11	26.30 ± 0.73	24.46 ± 0.61	57.50 ± 1.21	39.97 ± 0.98	34.36 ± 0.55	30.47 ± 0.67
$Strategy3$ [۴۳]	37.51 ± 2.43	19.24 ± 0.84	14.16 ± 0.66	16.45 ± 0.63	37.58 ± 0.38	30.37 ± 5.48	25.88 ± 0.60	23.89 ± 0.71	52.71 ± 1.98	37.73 ± 0.97	33.41 ± 0.64	30.32 ± 0.51
Proposed	**38.86 ± 3.12**	**20.13 ± 0.98**	**16.43 ± 0.68**	**17.84 ± 0.76**	**47.33 ± 1.89**	**33.65 ± 5.62**	**26.12 ± 0.65**	**25.39 ± 0.79**	**53.45 ± 1.38**	**38.12 ± 0.99**	**33.87 ± 0.69**	**31.12 ± 0.71**

جدول (۴): نتایج آزمون آماری t زوجی برای روش پیشنهادی و سایر الگوریتم‌ها در مسائل محک پویا: در جدول R معرف RANDOM است.

statistical tests	*Knapsack dynamic*					Order − 4 trap					Royal Road				
$\tau = 10, \quad \rho \Rightarrow$	0.1	0.2	0.5	1.0	R	0.1	0.2	0.5	1.0	R	0.1	0.2	0.5	1.0	R
$Strategy1$ [۴۲], $Strategy2$ [۴۲]	+	-	+	+	~	+	-	+	+	-	~	+	-	+	~
$Strategy1$ [۴۲], $Strategy3$ [۴۳]	-	+	+	-	+	-	-	+	-	~	+	+	+	~	-
$Strategy2$ [۴۲], $Strategy3$ [۴۳]	+	+	~	+	-	-	+	+	~	-	+	-	+	+	~
$proposed, Strategy1$ [۴۲]	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
$proposed, Strategy2$ [۴۲]	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
$proposed, Strategy3$ [۴۳]	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
$\tau = 50, \quad \rho \Rightarrow$	0.1	0.2	0.5	1.0	R	0.1	0.2	0.5	1.0	R	0.1	0.2	0.5	1.0	R
$Strategy1$ [۴۲], $Strategy2$ [۴۲]	+	-	+	~	~	+	+	+	~	-	~	+	+	-	~
$Strategy1$ [۴۲], $Strategy3$ [۴۳]	-	+	+	+	~	+	+	+	-	~	+	+	+	~	-
$Strategy2$ [۴۲], $Strategy3$ [۴۳]	-	-	~	+	+	+	+	-	~	-	+	+	-	+	~
$proposed, Strategy1$ [۴۲]	+	+	+	+	+	+	+	~	+	+	+	+	~	+	+
$proposed, Strategy2$ [۴۲]	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
$proposed, Strategy3$ [۴۳]	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
$\tau = 100, \quad \rho \Rightarrow$	0.1	0.2	0.5	1.0	R	0.1	0.2	0.5	1.0	R	0.1	0.2	0.5	1.0	R
$Strategy1$ [۴۲], $Strategy2$ [۴۲]	-	+	+	+	~	-	+	+	-	+	~	+	-	-	~
$Strategy1$ [۴۲], $Strategy3$ [۴۳]	+	+	+	+	+	+	+	+	-	~	-	+	+	~	-
$Strategy2$ [۴۲], $Strategy3$ [۴۳]	+	+	~	+	-	-	+	+	~	-	+	-	-	+	~
$proposed, Strategy1$ [۴۲]	+	+	+	+	+	+	+	~	+	+	+	+	+	+	+
$proposed, Strategy2$ [۴۲]	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
$proposed, Strategy3$ [۴۳]	+	+	+	~	+	+	+	+	+	+	+	+	+	+	+

نتایج تجربی مربوط به Strategy 1، Strategy 2، Strategy 3 و روش پیشنهادی در دو معیار مورد مطالعه (فرکانس تغییرات و شدت تغییرات (τ و ρ)) در محیط‌های مختلف در جدول‌های (۱) تا (۴) نشان داده شده است. بهترین نتایج به‌دست آمده توسط روش پیشنهادی با حروف درشت مشخص شده است. همان‌طور که می‌بینیم، بسته به فرکانس تغییرات و شدت تغییرات، انتخاب جمعیت و اندازه حافظه بر عملکرد الگوریتم تأثیر می‌گذارد. بنابه نتایج کسب شده از جداول (۱)

تا (۳)، برای مساله Knapsack، در روش پیشنهادی، بهترین انتخاب جمعیت با ۶۰ یا ۵۰ نفر (و حافظه ۴۰ و ۵۰ نفر) برای تغییر سریع سرعت ($\tau = 10$) می‌باشد. در محیط‌هایی با تغییرات کندتر(($\tau = 50$) و ($\tau = 100$))، بهترین نتایج به‌ترتیب با اندازه جمعیت ۸۰ یا ۷۰ و حافظه با ۲۰ یا ۳۰ نفر به‌دست آمده است. هر دو جمعیت بیش از حد کوچک یا بیش از حد بزرگ منجر به کاهش عملکرد می‌شوند. برای مسئله Onemax، در روش پیشنهادی از اندازه جمعیت ۴۰ نفر با استفاده از تغییرات سریع‌تر و نسبت‌های تغییر پایین‌تر ($\rho = 0.1$ و $\rho = 0.2$) و جمعیت با ۶۰ نفر برای نسبت‌های تغییر بزرگتر ($\rho = 1.0$ و $\rho = 0.5$) بهترین نتایج را به‌دست آورد. با افزایش نسبت تغییر و دوره تغییر، جمعیت‌های بیشتری برای دستیابی به بهترین نتایج لازم هستند: ۷۰ تا ۹۰ نفر.

۴،۲،۱. تحلیل پیچیدگی محاسباتی در روش پیشنهادی

مولفه های اصلی روش پیشنهادی تقسیم کردن جمعیت ذرات در داخل سلول های آتوماتای سلولی و حافظه تخمین تراکم گوسی است. از آن‌جایی‌که تقسیم ذرات در داخل سلول‌ها به‌صورت راهکار خودسازماندهی بحرانی صورت می‌گیرد بنابراین این میزان زمان برای الگوریتم می‌تواند ناچیز فرض شود. پس در روش پیشنهادی بازیابی از حافظه را می‌توان مولفه اصلی در زمان اجرای الگوریتم در نظر گرفت. بنابراین اگر $\forall i,j \in \{1,2,\dots,\dot{K}\}: i \neq j \rightarrow \zeta_j^N \subset \left\{1,2,\dots,\alpha \times \frac{Cf}{2}\right\}$ و $\zeta_j^N \cap \zeta_i^N = \emptyset$ باشد که در آن α نرخ اندازه حافظه (معمولا 0.1)، CS فرکانس تغییرات محیط به‌طور کلی و $P = (1+\alpha) \times \frac{Cf}{2}$ اندازه کل جمعیت باشد، آن‌گاه زمان اجرای الگوریتم به‌صورت زیر محاسبه می‌شود:

$$O\left(\frac{iteation}{Cf} \times \left(P \times \alpha \times \log_2^{fitness}\right)\right)$$

که در آن P، $iteation$، α و $fitness$ به‌ترتیب اندازه جمعیت، تعداد تکرار، نرخ اندازه حافظه میزان شایستگی هر فرد در تابع کارایی است.

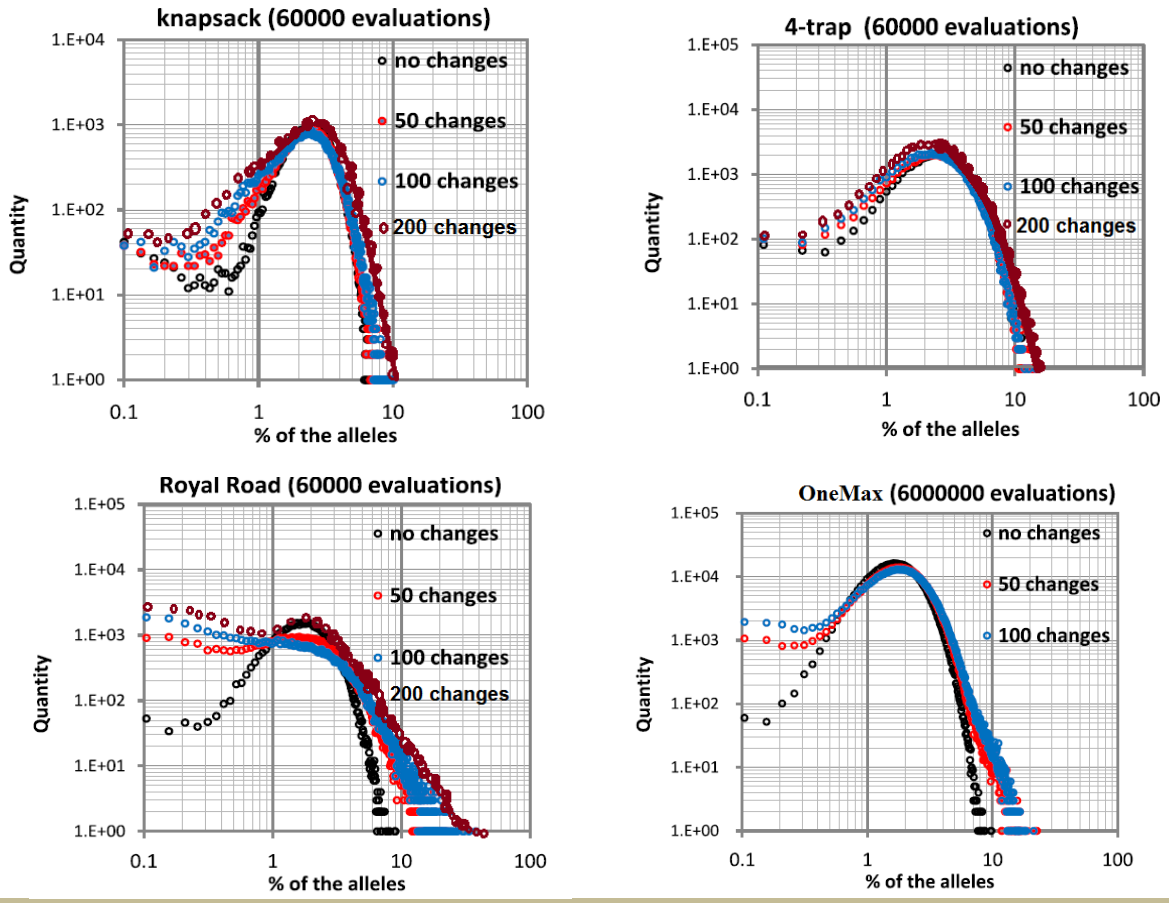
بنابه این که عمده زمان مصرفی در مرتب سازی شایستگی راه‌حل‌ها صرف شده است، مرتبه الگوریتم می‌تواند $O\left(\frac{iteation}{Cf} \times P \times \log P\right)$ فرض شود که مشابه حالت ساده آن یعنی $O(P \times \log P)$ است.

۴،۲،۲. تاثیر تغییرات فرکانس محیطی بر روی روش پیشنهادی با نرخ‌های جهش مختلف

در روش پیشنهادی تاثیر تغییرات فرکانس محیطی بر روی روش پیشنهادی با نرخ‌های جهش مختلف نشان داده شده است. همان‌گونه که مشخص می‌باشد روش پیشنهادی توانسته با کاهش فرکانس تغییرات (افزایش تعداد تغییرات محیطی)، در توابع محک مختلف نتایج مناسب و قابل قبولی از خود نشان دهد. شکل‌های (۱۲) بیانگر این موضوع می‌باشند.

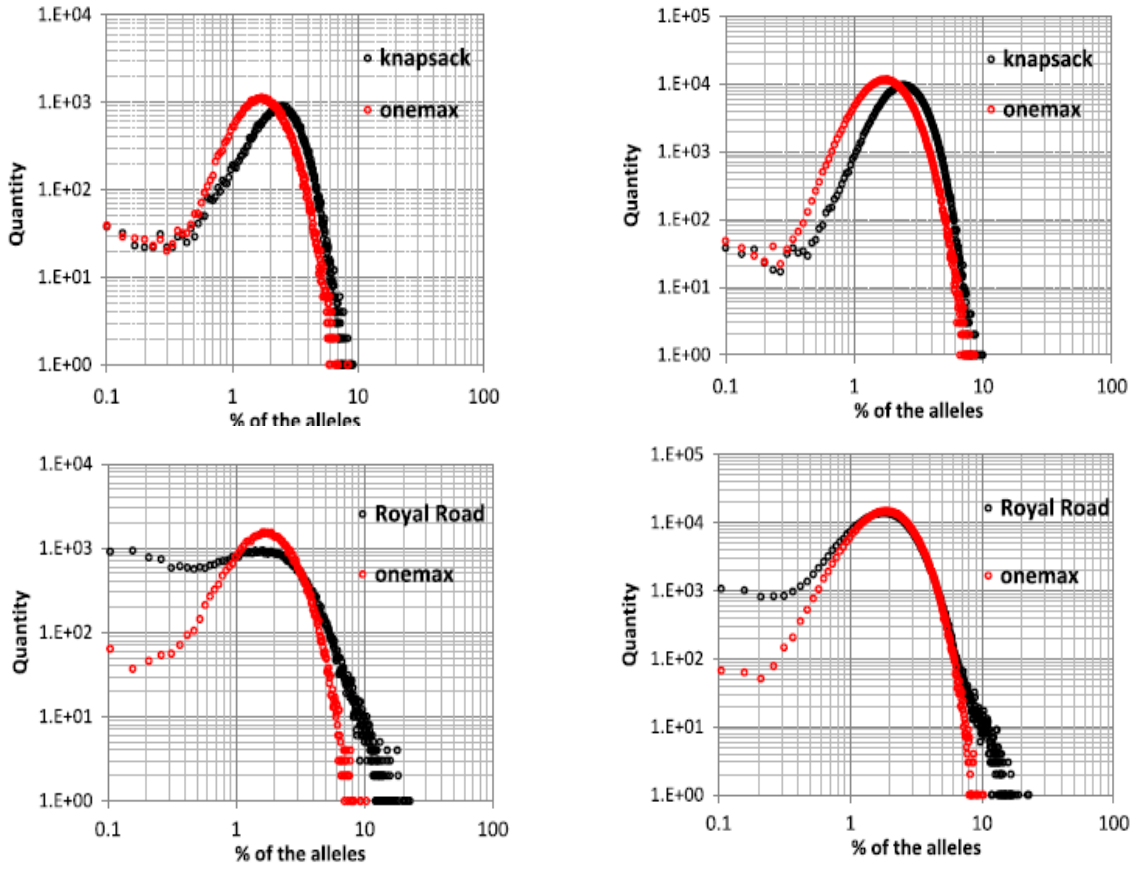
شکل (۱۳) نتایج را با دو مقدار متفاوت ε و ρ به‌صورت تصادفی تنظیم می‌کند. تفاوت در نمودارهای توزیع به‌وضوح مشخص است. هنگامی‌که با مسائل پویای Onemax مقایسه می‌شود، Knapsack قله با اندازه متوسط را تغییر می‌دهد. در توابع Royal Road، آن‌ها تمایل به افزایش فعالیت در ناحیه با جهش کم در مقایسه با توزیع Onemax دارند. روش پیشنهای با راهکار پیشنهادی قادر است نرخ جهش خود را با توجه به نوع تابع پایه ثابت تنظیم کند (به‌صورت خودسازمانده بحرانی). نمودار توزیع جهش، تأثیر تغییر در پارامترهای مولد مساله را نشان می‌دهد. با این‌حال، چنین نمایش داده‌ای هیچ اطلاعاتی در مورد توزیع مقادیر در طول اجرا، یا اگر مربوط به تغییرات محیطی باشد، نمی‌دهد.

بررسی میزان تنوع در جمعیت اصلی و حافظه برای روش پیشنهادی در شکل (۱۴) اندازه‌گیری و ترسیم شده است. همان‌طور که قبلا بیان شد، یکی از مهم‌ترین چالش‌ها در محیط‌های پویا حفظ تنوع می‌باشد. از شکل (۱۴)، می‌توان نتیجه گرفت که سطح تنوع در جمعیت اصلی پایین‌تر از حافظه است. در حافظه، روش جایگزینی پیشنهادی، در اولین تغییرات محیطی، تنوع بیشتری را در حافظه حفظ می‌کند. بعد از آن، تنوع حافظه ثابت می‌ماند زیرا افراد مختلف مدام به حافظه معرفی می‌شوند.

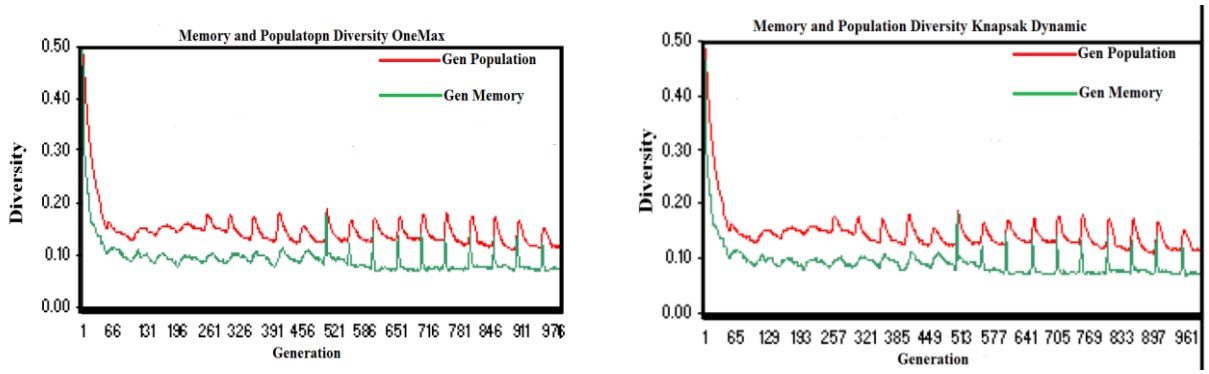


شکل (۱۲): تاثیر فرکانس تغییرات بر روی نرخ‌های جهش

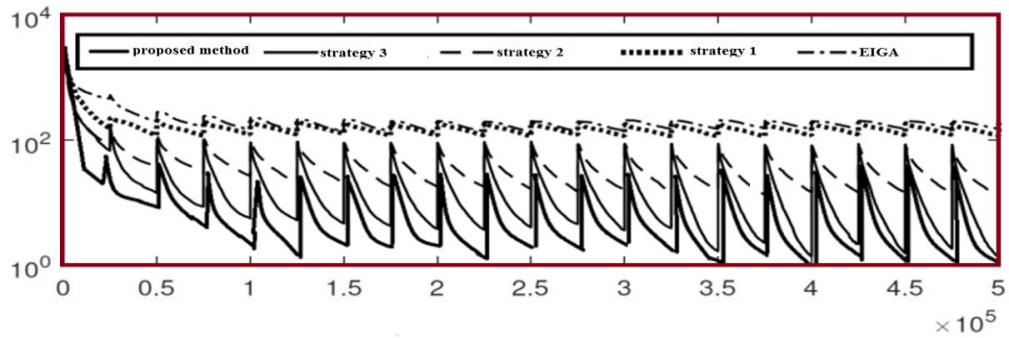
شکل (۱۵) روش پیشنهادی را از نظر میزان خطای جاری با سایر روش‌های رقیب مقایسه نموده و نتایج حاصله حاکی از برتری روش پیشنهادی در مقایسه با سایر روش‌ها است. اشکال ترسیم شده در نمودار (۱۶) روند تولید خطای جاری را برای روش پیشنهادی با فرکانس‌های تغییر مختلف و تغییرات مختلف محیطی نشان می‌دهند. دو نمودار ترسیم شده در شکل (۱۷)، میزان همگرایی روش پیشنهادی را در مقایسه با سایر روش‌های رقیب نشان می‌دهد. برای مقایسه میزان همگرایی در روش پیشنهادی و سایر روش‌های رقیب از پارامترهای استاندارد تعریف شده در [۴۶] استفاده شده تا شرایط مقایسه یکسان باشد. در این نمودارها محور افقی تعداد تکرار و محور عمودی میزان شایستگی را در تابع کارآیی برای هر فرد از جمعیت نشان می‌دهد. در مسائل بهینه‌سازی پویا همگرایی زودرس یا همگرایی محض می‌تواند الگوریتم را در حل این مسائل دچار مشکل نماید. پس باید الگوریتم به گونه‌ای باشد که، علاوه‌بر این‌که بهینه (بهینه‌های) در حال نوسان را ردیابی می‌کند، بتواند حداکثر تنوع در فضای مسئله ایجاد کند. این بدین معنی است که الگوریتم نباید همگرایی زودرس داشته باشد یا فقط به یک بهینه همگرا شود. نتایج نمودار ترسیم شده در شکل (۱۷) نشان می‌دهند که الگوریتم پیشنهادی توانسته به‌نحو مطلوبی این چالش را برآورده سازد. این نمودارها نشان می‌دهند که روش پیشنهادی در عین این‌که سرعت همگرایی بیشتری نسبت به سایر رقبا دارد، توانسته بعد از تعدادی تکرار به یک پایداری در همگرایی برسد و این بدین معنی است که افراد جمعیت بعد از ردیابی هر بهینه، در فضای مساله پخش شده و مجدد به دنبال سایر بهینه‌ها در فضای مساله هستند (اصل تنوع در حل مسائل بهینه‌سازی پویا را رعایت نموده است).



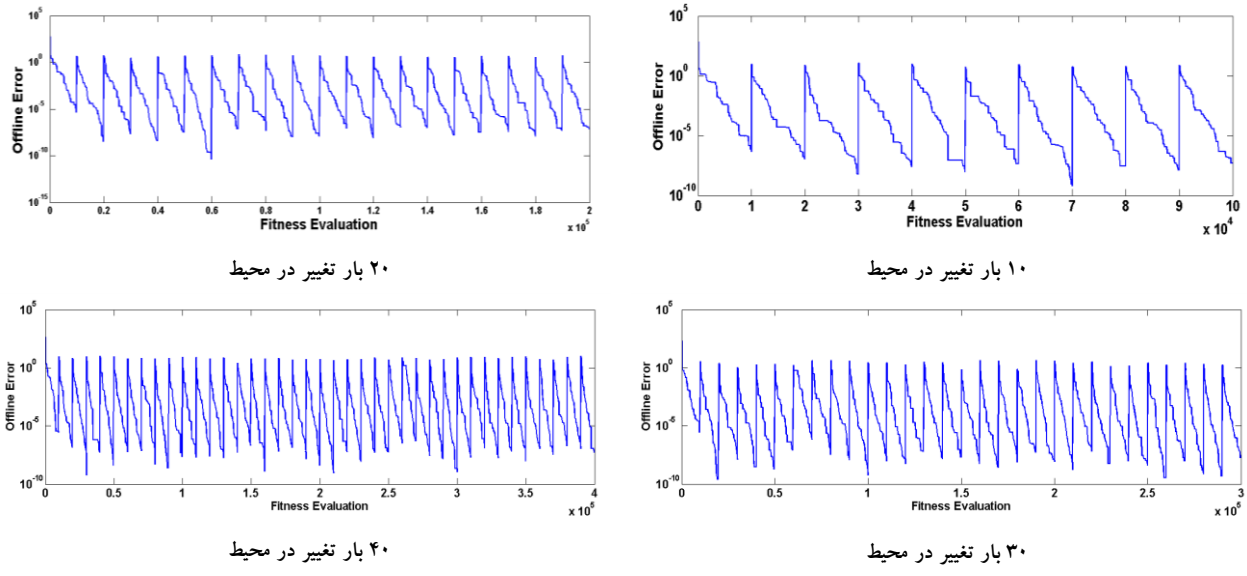
شکل (۱۳): مقایسه توابع محک مختلف بر روی روش پیشنهادی با نرخ‌های جهش مختلف



شکل (۱۴): بررسی میزان تنوع در حافظه و جمعیت اصلی



شکل (۱۵): میزان خطای جاری روش پیشنهادی در ارزیابی‌های مختلف در مقایسه با روش‌های رقیب با پارامترهای استاندارد [۴۶] و تابع کوله‌پشتی



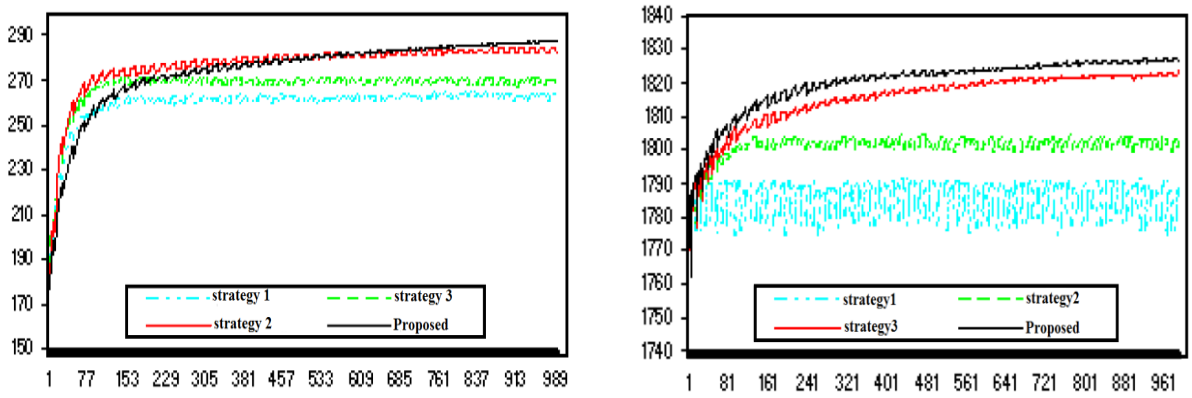
۱۰ بار تغییر در محیط

۲۰ بار تغییر در محیط

۳۰ بار تغییر در محیط

۴۰ بار تغییر در محیط

شکل (۱۶): روند تولید خطای جاری در ۵۰۰۰۰ بار ارزیابی کارآیی با تعداد تغییر متفاوت در محیط



OneMax

dynamic Knapsack

شکل (۱۷): مقایسه همگرایی روش پیشنهادی با سه استراتژی رقیب

۵. نتیجه‌گیری و پیشنهادات آتی

این مقاله جهت افزایش کارایی الگوریتم ژنتیک، راه کارهایی جهت حفظ تنوع جمعیت ارائه می‌دهد. این مقاله، جهت حفظ تنوع، یک الگوریتم برای نحوه جایگزینی و یک الگوریتم برای نحوه انتخاب مبتنی بر یک حافظه با عنوان تخمین تراکم گوسی، ارائه داده است. همچنین این الگوریتم یک عملگر جهش خودسازمانده مبتنی بر تپه شن ارائه کرده است. در حل مسائل بهینه‌سازی نیز، دقت استراتژی پیشنهادی بهتر از دیگر روش‌های به‌روز بوده، طوری که در حل سه تابع تله پویا، در اکثر موارد به‌طور قابل ملاحظه‌ای از رقبای دیگر پیشی می‌گیرد. برای حل مسائل بهینه‌سازی پویا، روش پیشنهادی با حافظه ترکیب گردید تا با حفظ سطح مناسبی از تنوع در حین اجرای الگوریتم، کارآیی آن افزایش یابد. نتایج حاصله از الگوریتم پیشنهادی در بخش نتایج تجربی نشان می‌دهند که روش پیشنهادی توانسته نسبت به سایر الگوریتم‌های رقیب برتری خود را نشان دهد. این برتری به‌دلیل استفاده از نرخ جهش خودتنظیم شونده، حافظه و نحوه به‌روزرسانی مناسب آن با روش‌های حافظ تنوع پیشنهادی در این مقاله می‌باشد. کارآیی برون‌خطی و انحراف استاندارد در روش پیشنهادی توانسته نسبت به سایر روش‌های رقیب نتایج بهتری را از خود نشان دهد. همچنین نتایج آماری به‌دست آمده از بخش نتایج تجربی به‌خوبی بیانگر این موضوع است که روش پیشنهادی از

نظر معناداری نسبت به روش‌های رقیب بهتر عمل نموده است. میزان خطای برون‌خطی نیز می‌تواند معیار مناسبی برای سنجش کارآیی الگوریتم‌ها در محیط‌های پویا باشد. در این راستا آزمایش‌هایی بر روی روش پیشنهادی انجام گرفت و نتایج به‌دست آمده نشان دادند که روش پیشنهادی خطای کمتری را نسبت به سایر روش‌های رقیب تولید نموده است. هر چه تغییرات محیطی بیشتر در نظر گرفته شود، پیچیدگی فضای مساله نیز افزایش پیدا می‌کند، پس در همین راستا آزمایش‌هایی انجام گرفت و نتایج حاکی از این بودند که روش پیشنهادی با افزایش تغییرات محیطی، مقاوم می‌باشد. روش اجماع خوشه‌بندی‌ها[1]، دارای قدرت استحکام، پایداری، دقت، و عمومیت بیشتری هستند. خوشه‌بندی، فرآیند سازمان‌دهی الگوها در چند دسته است به نحوی که اعضای هر دسته از جنبه‌هایی به هم شبیه و از جنبه‌هایی با اعضای دیگر دسته‌ها متفاوت باشند. الگوریتم‌های خوشه‌بندی منفردی وجود دارند که نتایج با کیفیت بدست می‌دهند، اما توانایی یک الگوریتم خوشه‌بندی منفرد محدود است. روش‌های ترکیبی این امکان را فراهم می‌کنند که در برخی موارد خوشه‌بندی با کیفیتی بالاتر از هر الگوریتم منفردی انجام شود. در خوشه‌بندی ترکیبی، نتایج چند خوشه‌بندی با یکدیگر ترکیب می‌شود و برآیند آن‌ها افرازهای جدیدی به‌دست می‌دهد که درک بهتر از الگوها فراهم می‌کنند. برای ادامه کار پیشنهاد می‌شود این مدل خوشه‌بندی را در حافظه گوسی پیاده‌سازی نمود. روش پیشنهادی از یک مدل احتمالی به نام مدل آمیخته گاوسی برای خوشه‌بندی پایه استفاده می‌کند. این روش در دو فاز ساختن مجمع و تعیین راه‌حل نهایی اجرا می‌شود. در فاز ساختن مجمع، نخست یک مجمع موقت ساخته می‌شود که اعضای آن برای بهبود مجمع نهایی به کار گرفته می‌شوند. در ساختن مجمع نهایی، هر خوشه‌بندی توسط یکی از الگوریتم‌های تکاملی چنان بهینه می‌شود که خوشه‌های آن با خوشه‌هایی که در مجمع موقت موجود هستند، شباهت داشته باشد. پس از ساختن مجمع نهایی، خوشه‌های موجود در آن خوشه‌بندی می‌شوند. با توجه به اینکه هر خوشه با یک توزیع نرمال بازنمایی می‌شود، یک افرازبندی روی مجموعه‌ای از توزیع‌های نرمال تولید می‌شود. در فاز دوم، راه‌حل نهایی که افرازبندی نمونه‌های مجموعه‌داده اصلی است، تولید می‌شود. برای این کار، میانگین احتمال عضویت نمونه‌ها به هر یک از خوشه‌های توزیع نرمال محاسبه می‌شود و نمونه به خوشه‌ای از توزیع‌های نرمال نسبت داده می‌شود که بیشترین میانگین احتمال عضویت در آن را دارا باشد. برای کارهای آینده پیشنهاد می‌شود که روش پیشنهادی با روش خوشه‌بندی ترکیبی (اجماعی از خوشه‌بندی‌ها) ترکیب شده و نتایج با روش پایه مقایسه گردد. همچنین بنا به این‌که اکثر مسائل موجود در دنیای واقعی مسائلی با ماهیت پویا هستند، پیشنهاد می‌شود روش پیشنهادی بر روی مسائل واقعی از جمله مساله زمانبندی کارگاهی پویا مورد آزمایش قرار گیرد.

تعارض منافع: نویسندگان اعلام می‌کنند که هیچ تعارض منافعی ندارند.

[1] Ensemble Clustering

مراجع

[1] Grefenstette J. J. and Ramsey C. L., "An approach to anytime learning", Proceedings of the Ninth International Workshop on Machine Learning (ML 1992), Aberdeen, Scotland, UK, July 1-3, pp. 189–195, 1992.

[2] Branke J., Kaußler T., Schmidt C., and Schmeck H., "A multi population approach to dynamic optimization problems". In Adaptive Computing in Design and Manufacturing, pp. 299–308, 2000.

[3] Branke J., "Evolutionary Optimization in Dynamic Environments", Universität Karlsruhe, pp. 1-174, 2000.

[4] Darwin C., "On the Origins of species by means of natural selection, or the preservation of favoured race in the struggle for life", LONDON: JOHN MURRAY (1STEDITION), 1859.

[5] Jin Y., and Branke J., "Evolutionary optimization in uncertain environments: a survey", IEEE Transactions On Evolutionary Computation 9(3): 303-317, 2005.

[6] Whitley L.D. and Kauth J., "Genitor: A different genetic algorithm", In Proceedings of the Rocky Mountain Conference on Artificial Intelligence, 118-130, 1988.

[7] Cobb H.G., "An investigation into the use of hyper-mutation as an adaptive operator in genetic algorithms having continuous, time dependent nonstationary environments", Technical Report Aic-90-001, Naval Research Laboratory, Washington, USA, 1990.

[8] Grefenstette J.J., "Evolvability in dynamic fitness landscapes: A genetic algorithm approach", In Proceedings of the 1999 IEEE Congress on Evolutionary Computation (Cec'1999), Vol. 3, IEEE Press, 2031-2038, 1999.

[9] Bak P. and Sneppen K., "Punctuated equilibrium and criticality in a simple model of evolution", Physical review of letters, 71(24):4083-4086, 1993.

[10] Stephens C.R., Olmedo I.G., Vargas J.M., and Waelbroeck H., "Self-adaptation in evolving systems", Artificial life, 4(2):183-201, 1998.

[11] Goldberg D.E. and Smith R.E., "Nonstationary function optimization using genetic algorithms with dominance and diploidy", In J.J. Grefenstette (Ed.): Proceedings Of The 2nd International Conference On Genetic Algorithm, L. Erlbaum Associates, Hillsdale, Nj, USA, 59-68, 1987.

[12] Etaner-Uyar A.S. and Harmanci A.E., "A new population based adaptive domination change mechanism for diploid genetic algorithms in dynamic environments", Soft Computing, 9(11):803–815, 2005.

[13] Ryan C., "Diploidy without dominance", In J.T. Alander (Ed.), Proceedings of the 3rd Nordic Workshop on Genetic Algorithms and Their Applications, 63-70, 1997.

[14] Lewis J., Hart E., and Ritchie G., "A comparison of dominance mechanisms and simple mutation on non stationary problems", In A.E. Eiben, T. Bäck, M. Schoenauer, H.-P. Schwefel (Eds.), Proceedings Of 5th International Conference On Parallel Problem Solving From Nature (PPSN-V), Lecture Notes In Computer Science, Vol. 1498, Springer-Verlag, London, UK, 139-148, 1998.

[15] Dasgupta D. and Mcgregor D.R., "Nonstationary function optimization using the structured genetic algorithm", In R. Manner R And B. Manderick (Eds.), Proceedings Of The 2nd International Conference on Parallel Problem Solving From Nature (Ppsn-Ii), Elsevier Science, New York, 145-154, 1992.

[16] Klinkmeijer L.Z., De Jong E., and Wiering M. "A serial population genetic algorithm for dynamic optimization problems", In Y. Saeys, E. Tsiporkova, B. De Baets And Y..Van De Peer (Eds.), Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands, 41-48, 2006.

[17] Hollstein R.B., "Artificial genetic adaptation in computer control systems", Doctoral Thesis. University of Michigan, 1971.

[18] Ramsey C.L. and Grefenstette J.J., "Case-based initialization of genetic algorithms", In S. Forrest (Ed.), Proceedings of the 5th International Conference Genetic Algorithms (Icga'1993), Morgan Kaufmann, San Francisco, 84–91, 1993.

[19] Trojanowski K. and Michalewicz Z., "Evolutionary optimization in non-stationary environments", Journal of Computer Science and Technology, 1(2):93-124, 2000.

[۲۰] محمدپور م.، پروین ح.، «الگوریتم ژنتیک آشوب‌گونه مبتنی بر حافظه و خوشه‌بندی برای حل مسائل بهینه-سازی پویا»، مجله مهندسی برق دانشگاه تبریز، جلد ۴۶،

[30] Martello S. and Toth P., "Knapsack problems: Algorithms and computer implementations", John Wiley & Sons, New York, 1990.

[۳۱] سلیمی سرتختی ج.، گلی بیدگلی س.، «ارائهٔ یک الگوریتم ترکیبی با استفاده از الگوریتم کرم شب‌تاب، الگوریتم ژنتیک و جست‌وجوی محلی»، مجله محاسبات نرم، جلد ۸، شماره ۱، ص ۲۸-۱۴، ۱۳۹۸.

[32] Agarwal R., Schuurmans D., and Norouzi M., "An optimistic perspective on offline reinforcement learning", In International Conference on Machine Learning 119:104-114, 2020.

[33] Cabi S., Colmenarejo S.G., Novikov A., Konyushkova K., Reed S., Jeong R., Zołna K., Aytar Y., Budden D., Vecerik M., Sushkov O., Barker D., Scholz J., Denil M., de Freitas N., and Wang Z., "Scaling datadriven robotics with reward sketching and batch reinforcement learning", arXiv preprint:1909.12200, 2019.

[34] Castro P.S., Moitra S., Gelada C., Kumar S., and Bellemare M.G., "Dopamine: A research framework for deep reinforcement learning", 2018. URL: http://arxiv.org/abs/1812.06110.

[35] Bäck T., "Evolutionary algorithms in theory and practice", Oxford University Press, New York, 1996.

[36] Goldberg D.E. "The design of innovation lessons from and for competent genetic algorithms", Kluwer Academic Publishers, Norwell, Ma, 2002.

[37] Baluja S., "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning", Technical Report CMU-Cs 94-163, Carnegie Mellon University, USA, 1994.

[38] Harik G.R., Lobo F., and Sastry K., "Linkage learning via probabilistic modeling in the ecga", In M. Pelikan, K. Sastry, E. Cantú-Paz (Eds.), Scalable Optimization Via Probabilistic Modeling: From Algorithms To Applications, Springer-Verlag New York Inc., Secaucus Inc., Nj, Usa,39-61, 2006.

[39] Bak P., Tang C., and Wiesenfeld K., "Self organized criticality: An explanation of 1/F noise", Physical Review of Letters, 59(4):381-384, 1987.

[40] Bak P. and Sneppen K., "Punctuated equilibrium and criticality in a simple model of evolution", Physical Review of Letters, 71(24):4083-4086, 2003.

[41] Fernandes C.M., "Pherographia: Drawing by ants",

شماره ۳، پاییز ۱۳۹۶.

[۲۱] محمدپور م.، مینایی بیدگلی ب.، پروین ح.، «ارائه یک الگوریتم فرااکتشافی جدید مبتنی بر رفتار پرنده تیهو برای حل مسائل بهینه‌سازی پویا»، مجله محاسبات نرم، جلد ۸، شماره ۲، ص ۶۵-۳۸، ۱۳۹۸.

[22] Mohammadpour M., Parvin H., and Sina M., "Chaotic genetic algorithm based on explicit memory with a new strategy for updating and retrieval of memory in dynamic environments", J AI Data Min 6:191–205, 2018 (in press).

[23] Mohammadpour M. and Parvin H., "Genetic algorithm based on explicit memory for solving dynamic problems", In Journal of Advances in Computer Research Sari Branch Islamic Azad University, 7(2):53-68, 2016.

[24] Parvin H., Nejatian S., and Mohammadpour M., "Explicit memory based ABC with a clustering strategy for updating and retrieval of memory in dynamic environments", In Applied Intelligence Springer Nature, 2018. doi: 10.1007/s10489-018-1197-z.

[25] Ursem R.K., "Multinational GA optimization techniques in dynamic environments", In Proceedings of the 2000 Genetic and Evolutionary Computation Conference (Gecco'2000), Morgan Kaufmann, San Francisco, 19-26, 2000.

[26] Park T. and Ryu K.R., "A dual population genetic algorithm with evolving diversity", In Proceedings of the 2007 IEEE Congress on Evolutionary Computation (Cec'2007), IEEE Press, 3516-3522, 2007.

[27] Park T., Choe R., and Ryu K.R., "Adjusting population distance for the dual-population genetic algorithm", In Proceedings of The Australian Conference On Artificial Intelligence (Ai'2007), Lecture Notes On Computer Science, Vol. 4830, Springer Verlag, Berlin171-180, 2007.

[28] Park T., Choe R., and Ryu K.R., "Dual population genetic algorithm for nonstationary optimization", In Proceedings of the 2008 Genetic and Evolutionary Computation Conference (Gecco'2008), Acm, New York, Ny, 1025-1032, 2008.

[29] Yang S., "Constructing test environments for genetic algorithms based on problem difficulty", In Proceedings of the 2004 IEEE Congress On Evolutionary Computation, Vol. 2, IEEE Press, 2004, 1262 1269.238, 2004.

Leonardo–Journal of the International Society for the Arts, Sciences and Technology, Mit Press, 2009, In Press.

[42] Fernandes C.M., Merelo J.J., Ramos V., and Rosa A.C., "A selforganized criticality mutation operator for dynamic optimization problems", In: Proceedings of the 2008 genetic and evolutionary computation conference. ACM, New York, pp. 937–944, 2008.

[43] Fernandes C.M., Laredo J.L.J., Rosa A.C., and Merelo J.J., "The sandpile mutation Genetic Algorithm: an investigation on the working mechanisms of a diversity-oriented and self-organized mutation operator for non-stationary functions", Applied Intelligence 39: 279–306, 2013.

[44] Tinós R. and Yang S., "A self-organizing RIGA for dynamic optimization problems", Genet Program Evol Mach 8:255–286, 2007.

[45] Yang S., "Genetic algorithms with memory- and elitismbased immigrants in dynamic environments", Evol Comput 6(3):385–416, 2008.

[46] Fernandes C.M., Merelo J.J., and Rosa A.C., "A comparative study on the performance of dissortative mating and immigrants' strategies for evolutionary dynamic optimization", Inf Sci 181(20):4428–4459, 2011.

[47] Martello S. and Toth P., "Knapsack problems: algorithms and computer implementations", John Wiley & Sons, New York, 1990.

[48] Jana B., Mitra S., and Acharyya S., "Repository and mutation based particle swarm optimization (rmpso): a new PSO variant applied to reconstruction of gene regulatory network," Applied Soft Computing, 74:330–355, 2019.

[49] Pampara G. and Engelbrecht A.P., "Self-adaptive quantum particle swarm optimization for dynamic environments," International Series in Operations Research and Management Science, 272:163–175, 2018.

[50] Dzalbs I. and Kalganova T., "Accelerating supply chains with Ant Colony Optimization across range of hardware solutions," arXiv:2001.08102, 2020.

[۵۱] محمدی ح.، اخوان ع.، «مقایسه عملکرد الگوریتم‌های HSA، ICA و PSO به‌منظور حذف انتخابی هارمونیک‌ها در اینورتر چندسطحی آبشاری با وجود منابع DC متغیر»، مجله محاسبات نرم، جلد ۳، شماره ۲، ص ۳۰-۲۰، ۱۳۹۳.

[52] Parvin H., Minaei B., and Ghatei S., "A new particle swarm optimization for dynamic environments," Part of the Lecture Notes in Computer Science book series (LNCS, volume 6694), Computational Intelligence in Security for Information Systems pp 293-300, 2014.

[53] Hashemi A. and Meybodi M., "Cellular PSO: A PSO for dynamic environments," Advances in Computation and Intelligence, pp. 422–433, 2009.

[54] Assimi H., Harper O., Xie Y., Neumann A., and Neumann F., "Evolutionary bi-objective optimization for the dynamic chance-constrained Knapsack problem based on tail bound objectives," 24th European Conference on Artificial Intelligence - ECAI 2020 Santiago de Compostela, Spain, 2020.

[55] Roostapour V., Neumann A., and Neumann F., "On the performance of baseline evolutionary algorithms on the dynamic knapsack problem", in Parallel Problem Solving from Nature, PPSN XV 2018, Lecture Notes in Computer Science, pp. 158–169. Springer, 2018.

[56] Xie Y., Harper O., Assimi H., Neumann A., and Neumann F., "Evolutionary algorithms for the chance-constrained knapsack problem", in Proceedings of the Genetic and Evolutionary Computatio Conference, GECCO 2019, pp. 338–346, 2019.

[57] Zhao Z., Gu F., and Cheung Y.M., "Co-operative Prediction Strategy for Solving Dynamic Multi-Objective Optimization Problems," In: 2020 IEEE Congress on Evolutionary Computation (CEC), 2020, DOI: 10.1109/CEC48606.2020.9185721.

[58] Hu W., Jiang M., Gao X., Tan K.-C., and Cheung Y.-M., "Solving dynamic multi-objective optimization problems using incremental support vector machine", 2019 IEEE Congress on Evolutionary Computation (CEC), pp. 2794-2799, 2019.