بهبود حافظه برای حل مسئله زمان‌بندی کار کارگاهی پویا

مجید محمدپور[۱]، کارشناس ارشد؛ حمید پروین[۲،۳]، استادیار؛ صمد نجاتیان[۴]، استادیار

۱- باشگاه پژوهشگران جوان و نخبگان - دانشگاه آزاد اسلامی واحد یاسوج - ایران - m.mohammadpour@iauyasooj.ac.ir
۲- دانشکده فنی و مهندسی - دانشگاه آزاد اسلامی واحد نورآباد ممسنی - ایران - parvin@iust.ac.ir
۳- باشگاه پژوهشگران جوان و نخبگان - دانشگاه آزاد اسلامی واحد نورآباد ممسنی - ایران
۴- دانشکده مهندسی برق - دانشگاه آزاد اسلامی واحد یاسوج - ایران - samad.nej.2007@gmail.com

چکیده: وقتی با یک جهان در حال تغییر مواجه می‌شوید، انسان‌ها نه‌تنها به آینده بلکه به گذشته هم توجه می‌کنند. توجه کردن به راه‌حل‌های مشابه، به ما در تصمیم‌گیری در آینده کمک می‌کند. زمانی‌که با وضعیتی روبرو می‌شویم که قبلاً آن را تجربه کرده باشیم بهتر می‌توانیم با آن روبرو شویم. اگر در حل مسائل بهینه‌سازی با ماهیتی پویا در هنگام جستجو، از اطلاعات گذشته داخل بهینه‌سازی و یادگیری استفاده شود، می‌تواند به فرآیند جستجوی بهتر کمک کند. یکی از راه‌کارهای مناسب برای حفظ اطلاعات گذشته استفاده از یک حافظه است. در اکثر تحقیقات نشان داده‌شده است که به‌کارگیری یک حافظه استاندارد با الگوریتم‌های یادگیر تقلید از طبیعت می‌تواند برای حل مسائلی که ماهیتی پویا دارند مناسب باشد. حافظه استاندارد معمولاً دارای نقطه ضعفی از جمله، ظرفیت محدود حافظه می‌باشد. در این مقاله جهت برطرف نمودن نقاط ضعف و محدودیت‌های حافظه استاندارد، یک نوع جدید از حافظه باعنوان، حافظه مبتنی بر کلاس‌بندی معرفی شده است. این حافظه با الگوریتم ژنتیک ترکیب شده تا برای حل مسائل زمان‌بندی کار کارگاهی پویا به‌کار رود. مسئله زمان‌بندی کار کارگاهی پویا یکی از پیچیده‌ترین حالات زمان‌بندی ماشین به‌شمار می‌رود. استفاده از حافظه مبتنی بر کلاس‌بندی، مسائل پویایی که ممکن است بر اساس تغییر محیط منسوخ شوند را توسعه می‌دهد. این حافظه یک لایه انتزاعی میان راه‌حل‌های عملی و مدخل‌های حافظه ایجاد می‌کند، به‌طوری‌که راه‌حل‌های قدیمی ذخیره شده در حافظه به راه‌حل‌های محیط جاری نگاشت شوند.

واژه‌های کلیدی: حافظه مبتنی بر کلاس‌بندی، زمان‌بندی کار کارگاهی پویا، بهینه‌سازی، الگوریتم ژنتیک.

Improving Memory for Solving Dynamic Job Shop Scheduling

M. Mohammadpour[1], MSc; H. Parvin[2,3], Assistant Professor; S. Nejatian[4], Assistant Professor

1- Young Researchers and Elite Clubs, Yasooj Branch, Islamic Azad University, Yasooj, Iran, Email: m.mohammadpour@iauyasooj.ac.ir
2- Department of Computer Engineering, Noorabad Mamasani Branch, Islamic Azad University, Noorabad Mamasani, Iran, Email: parvin@iust.ac.ir
3- Young Researchers and Elite Clubs, Noorabad Mamasani, Islamic Azad University, Noorabad Mamasani, Iran
4- Department of Computer Engineering, Yasooj Branch, Islamic Azad University, Yasooj, Iran, Email: samad.nej.2007@gmail.com

Abstract: When faced with a changing world, humans are apt to look not just to the future, but to the past. Drawing on knowledge from similar situations we have encountered helps us to decide what to do next. The more experience we've had with a particular situation, the better we can expect to perform. When solving dynamic problems using search, it may be enough to solve the problem completely from scratch when we encounter it again. An appropriate strategy for store past information is memory. In the researches shown that using standard memory with evolutionary algorithms for solving dynamic optimization problem is capable. Standard memory is containing infirmity point memory determinate capacity. In this paper presented a new memory namely Classifier-based memory, which solves standard memory problems. This memory combined with GA for solving dynamic scheduling. The dynamic job shop scheduling problem is one of the most complex forms of machine scheduling. Classifier-based memory is introduced to extend the use of memory to dynamic problems where solutions may become obsolete as the environment changes. Classifier-based memory creates an abstraction layer between feasible solutions and memory entries so that old solutions stored in memory may be mapped to solutions that are feasible in the current environment. The technique presented in this paper improves the ability of memories to guide search quickly and efficiently to good solutions as the environment changes.

Keywords: Classifier-based memory, dynamic job shop scheduling**,** optimization, genetic algorithm.

تاریخ ارسال مقاله: ۱۳۹۵/۰۵/۲۷
تاریخ اصلاح مقاله: ۱۳۹۵/۰۷/۲۴
تاریخ پذیرش مقاله: ۱۳۹۵/۰۸/۲۴
نام نویسنده مسئول: حمید پروین
نشانی نویسنده مسئول: ایران – نورآباد ممسنی فارس - دانشگاه آزاد اسلامی- دانشکده مهندسی.

۱- مقدمه

مسائل بهینه‌سازی در جهان واقعی به دو دسته ایستا و پویا تقسیم می‌شوند. در مسائل بهینه‌سازی ایستا همه‌چیز مشخص است (بهینه ثابت است) اما در مسائل پویا محیط در حال تغییر است، در اکثر مواقع این محیط دارای تغییرات مشخص شده است که با ذخیره کردن راه‌حل‌های گذشته در حافظه می‌توان جهت محیط‌های جدید استفاده کرد.

حافظه به بهینه‌سازی پویا از طرق مختلف می‌تواند کمک نماید. ازجمله می‌توان به موارد زیر اشاره نمود:

۱. نگهداری راه‌حل‌های خوب گذشته.
۲. سرعت بخشیدن به جستجو برای راه‌حل‌های مشابه بعد از رویداد پویا.
۳. مدخل‌های حافظه کمک زیادی به جستجو بعد از تغییر می‌کنند، در واقع به ایجاد تنوع در فرآیند جستجو کمک می‌کند.
۴. حافظه به ایجاد یک مدل از مسائل پویا در هر زمان کمک می‌کند.

حافظه به‌طور گسترده در یادگیری و بهینه‌سازی پویا استفاده می‌شود.

انواع متعددی از حافظه وجود دارد. سیستم حافظه استاندارد برای بهینه‌سازی پویا در ابتدا پدید آمده است. در این سیستم حافظه استاندارد تعداد محدودی راه‌حل‌ها ذخیره شده و آن‌گاه در جستجو به‌کار می‌رود تا در صورت تغییر محیط نیز فرآیند جستجو را به‌سمت راه‌حل‌های خوب هدایت کند. از جمله مشکلات حافظه استاندارد این است که، اندازه حافظه برای ذخیره راه‌حل‌های کافی محدود است و تصحیح کل حافظه مشکل است.

راه‌حل‌های عملی (ممکن) در یک زمان خاص یک زیرمجموعه از کل فضای جستجو هستند، و آن مجموعه راه‌حل‌های ممکن در مسائل پویا تغییر می‌کنند. برای مثال در یک مسئله زمان‌بندی پویا، کارها تکمیل شده و کارهای جدید وارد می‌شوند و زمان‌بند قدیمی حاوی فقط کارهایی است که تکمیل شده‌اند و شامل کارهای جاری نیست. وقتی نواحی در فضای جستجو در طول زمان تغییر می‌کنند، در این‌جا حافظه به الگوریتم‌های یادگیری و بهینه‌سازی کمک می‌کند تا سریع پاسخ دهد و در تغییرات پویا کارآمد باشد. سیستم حافظه استاندارد با ذخیره کردن راه‌حل‌ها کمک زیادی می‌کند، اما تعداد محدودی از راه‌حل‌ها را ذخیره می‌کند، و مدخل‌های حافظه نمی‌توانند به‌طور کامل تصحیح شوند و هم‌چنین با راه‌حل‌های بهتر جایگزین شوند.

جهت برطرف نمودن نقاط ضعف و محدودیت‌های حافظه استاندارد یک نوع جدید از حافظه با عنوان، حافظه مبتنی بر کلاس‌بندی معرفی شده است.

حافظه مبتنی بر کلاس‌بندی برای مسائل پویایی استفاده می‌شود که در فضای جستجو، ناحیه‌ها در هر لحظه جابه‌جا می‌شوند. اگرچه یک راه‌حل ممکن است مدت‌ها قبل در حافظه ذخیره شود، اما می‌تواند از طریق بازیابی استفاده شود. در مسائل زمان‌بندی پویا، کارهای جاری در هر زمان تغییر می‌کنند.

حافظه مبتنی بر کلاس‌بندی جهت حل مسائل زمان‌بندی پویا به‌کار می‌رود. این حافظه کیفیت زمان‌بند را بهبود می‌دهد، و میزان جستجو را برای پیدا کردن زمان‌بندی‌های خوب کاهش می‌دهد.

یک مدخل حافظه به‌جای ذخیره‌سازی لیستی از وظایف خاص، لیستی از کلاس‌بندی‌هایی را ذخیره می‌کند که می‌تواند در هر زمان، به‌کار در حال انتظار نگاشت شود.

در اکثر تحقیقات نشان داده شده است که به‌کارگیری یک راه‌کار مناسب از جمله حافظه در کنار الگوریتم‌های یادگیر تقلید از طبیعت (الگوریتم‌های تکاملی) می‌تواند برای حل مسائل بهینه‌سازی پویا مناسب باشد[۱-۴].

الگوریتم‌های تکاملی برگرفته از طبیعت بوده و مبتنی بر جمعیت هستند. معروف‌ترین الگوریتم‌های تکاملی عبارتند از: الگوریتم ژنتیک، ازدحام ذرات، کلونی زنبور مصنوعی، کلونی مورچه‌ها و ...

در این پژوهش یک حافظه مبتنی بر کلاس‌بندی ارائه شده است. این حافظه با الگوریتم ژنتیک (GA) [۵] ترکیب شده تا مسئله زمان‌بندی پویا را حل نماید. در این مقاله هدف ارائه یک حافظه مناسب است تا به‌کمک یک الگوریتم تکاملی بتواند یک مسئله پویا را حل کند. درواقع نوآوری اصلی این مقاله، ارائه یک حافظه مبتنی بر کلاس‌بندی برای حل یکی از مهم‌ترین مسائل بهینه‌سازی پویا (مسئله زمان‌بندی کار کارگاهی پویا) است. در این حافظه سعی شده تا علاوه بر داشتن نقاط قوت حافظه استاندارد، نقاط ضعف آن برای حل مسئله زمان‌بندی کار کارگاهی پویا نیز رفع شود (باید دقت شود که مسئله زمان‌بندی کار کارگاهی پویا یکی از سخت‌ترین مسائل بهینه‌سازی است که از نوع NP-Hard است).

مقاله حاضر شامل ۵ بخش است به‌طوری‌که، در بخش اول مقدمه و کلیات موضوع بیان شده است، در بخش دوم ادبیات موضوع و مروری بر کارهای گذشته تشریح شده، در بخش سوم روش پیشنهادی بیان می‌شود و درنهایت بخش پنجم نتیجه‌گیری مقاله را بیان نموده است.

۲- ادبیات موضوع و پیشینه تحقیق

۲-۱- بهینه‌سازی پویا و برخی چالش‌های پیش‌رو

در بسیاری از مسائل بهینه‌سازی دنیای واقعی، تابع هدف، یا محدودیت‌ها می‌توانند در طول زمان تغییر یابند و بنابراین بهینه این مسائل نیز می‌تواند تغییر یابد. اگر هر یک از این رویدادهای نامعین در فرآیند بهینه‌سازی مورد توجه قرار گیرند، این مسئله پویا نامیده می‌شود. البته ساده‌ترین راه برای واکنش در برابر تغییر محیط، در نظر گرفتن هر تغییر به‌عنوان ورود یک مسئله بهینه‌سازی جدید است که باید از ابتدا حل شود. درصورت داشتن زمان کافی، این یک گزینه مناسب است. با وجود این، اغلب زمان برای بهینه‌سازی مجدد، نسبتاً کوتاه است. یک تلاش طبیعی برای تسریع فرآیند بهینه‌سازی پس از

یک تغییر، استفاده از اطلاعات مربوط به فضای جستجوی قبلی برای پیشروی در جستجو پس از تغییر است. به‌عنوان مثال اگر فرض شود که بهینه جدید نزدیک به بهینه قبلی قرار دارد، می‌توان جستجو را محدود به نقطه مجاور بهینه قبلی کرد. این‌که آیا استفاده مجدد از اطلاعات گذشته، مطلوب است یا نه، وابسته به ماهیت تغییر است. اگر تغییر به‌صورت ریشه‌ای و اساسی باشد و مسائل جدید دارای شباهت کمی با مسئله قبلی باشند، عمل شروع مجدد می‌تواند تنها گزینه مناسب باشد و هرگونه استفاده مجدد از اطلاعات جمع‌آوری شده بر مبنا مسئله قبلی گمراه کننده خواهد بود. در اکثر مسائل دنیای واقعی، امید می‌رود که تغییرات نسبتاً هموار باشند. سؤال اساسی که در این‌جا مطرح می‌شود این است که چه اطلاعاتی باید نگهداری شود و این اطلاعات چگونه باید برای تسریع عمل جستجو پس از تغییر محیط استفاده شوند. ولی حتی زمانی‌که اطلاعات مفید بتوانند انتقال یابند، بایستی تضمین شود که آیا الگوریتم بهینه‌سازی به‌قدر کافی برای پاسخ به این تغییرات انعطاف‌پذیر است؟ اکثر روش‌های اکتشافی در طی اجرا، تقارب پیدا می‌کند و از این راه قابلیت انطباق و سازگاری خود را از دست می‌دهند، بنابراین در کنار انتقال اطلاعات، یک روش ذهنی موفق برای مسائل بهینه‌سازی پویا باید دارای انطباق‌پذیری مؤثر باشد.

۲-۲- حافظه[1]

در بسیاری از مسائل پویا حالت فعلی محیط اغلب شبیه به حالات دیده شده قبلی است. استفاده از اطلاعات گذشته ممکن است به سیستم کمک کند تا با تغییرات بزرگ در محیط بهتر تطبیق پیدا کند و در طول زمان بهتر اجرا گردد. یک راه برای حفظ و بهره‌برداری از اطلاعات گذشته استفاده از حافظه است. حافظه محلی است که راه‌حل‌ها به‌صورت دوره‌ای در آن ذخیره می‌گردد و در زمانی که محیط تغییر می‌کند می‌توان آن‌ها را بازیابی کرد.

روش‌های بر پایه حافظه برای بهینه‌سازی پویا ممکن است با توجه به چگونگی ذخیره‌سازی حافظه، به دو دسته حافظه ضمنی و حافظه صریح تقسیم شوند. حافظه ضمنی اطلاعات گذشته را به‌عنوان بخشی از یک عضو ذخیره می‌کند ولی حافظه صریح اطلاعات را مجزا از جمعیت، معمولاً به‌صورت یک دسته از راه‌حل‌های خوب قبلی، ثبت می‌کند. روش حافظه صریح به‌طور گسترده‌تر بررسی شده است و عملکرد بهتری نسبت به روش حافظه ضمنی در مسائل پویا داشته است[۴، ۶].

۲-۲-۱- حافظه ضمنی[2]

حافظه ضمنی در الگوریتم‌های تکاملی، حافظه را درون کروموزوم‌های اعضای جمعیت قرار می‌دهد. چندین نوع مختلف از حافظه ضمنی وجود دارد اما احتمالاً رایج‌ترین آن‌ها استفاده از الگوریتم‌های تکاملی چنددستگی (مولتی پلوئید) است [۱۱-۷]. این روش از تعداد زیادی از اورگانیسم‌های زیست‌شناختی با ژن‌های نهفته الهام گرفته شده است.

چندین حالت دیگر از حافظه ضمنی نیز ایجاد شده است. در مرجع [۱۲] نویسندگان الگوریتم ژنتیک دوگانه را ساختند که یک بیت متغیر واحد را به رشته بیتی کروموزوم اضافه می‌کند. زمانی‌که بیت متغیر خاموش می‌شود رشته بیتی به‌صورت معمول خوانده می‌شود، اما وقتی‌که بیت متغیر روشن می‌گردد بخش تکمیلی رشته بیتی به‌جای آن خوانده می‌شود. روش‌های دیگر حافظه ضمنی که از مفهوم دوگانه بهره می‌برند شامل کارهای ارائه شده در مرجع [۱۳] و هم‌چنین [۱۴] می‌شوند.

۲-۲-۲- حافظه صریح[3]

حافظه صریح در الگوریتم‌های تکاملی حافظه را جدا از جمعیت، درون یک بانک حافظه ذخیره می‌کند. روش‌های حافظه صریح بسیار محبوب بوده و به‌طور گسترده‌ای در بهینه‌سازی پویا استفاده می‌شوند. راه‌حل‌های خاص برای ذخیره و بازیابی اطلاعات از حافظه داری تکنیک‌های متفاوت هستند اما ساختار عمومی حافظه در آن‌ها مشابه است. در ادامه این مقاله، از حافظه به‌عنوان بانک حافظه صریح و از یک مدخل حافظه به‌عنوان یک بخش از حافظه اطلاق می‌گردد.

در اوایل دوران استفاده از حافظه، محققین در مرجع [۱۵] یک حافظه موردی ساختند که راه‌حل‌های خوب قبلی و اطلاعات مربوط به محیطی که راه‌حل‌های مذکور در آن ایجاد شده‌اند ذخیره می‌کرد. هنگامی‌که یک تغییر رؤیت می‌شد، مدخل‌هایی با محیط‌های نسبتاً منطبق پیدا می‌شدند و از راه‌حل‌های آن مدخل‌ها برای مقداردهی مجدد بخشی از جمعیت استفاده می‌گردید. مسئله پویا به‌صورت مقطعی بود به‌همین دلیل مدخل‌های حافظه در هر تناوب ذخیره می‌شدند. به حافظه اجازه رشد بدون هیچ قید و شرطی داده شده بود و اندازه آن هیچ‌گاه کاسته نمی‌شد.

برانک [۶] یک مدل عمومی‌تر از حافظه ارائه کرد که نیازی به ذخیره اطلاعات محیط نداشت. حافظه تلاش می‌کرد تا به‌صورت تناوبی بهترین اعضای جمعیت را ذخیره کند. حافظه یک اندازه محدود و مشخص داشت و در زمان پر شدن حافظه از یک راه‌حل جایگزینی استفاده می‌شد که تصمیم می‌گرفت آیا نیازی به جایگزین بهترین عضو جمعیت با یکی از مدخل‌های حافظه هست یا نه. تعدادی گوناگون از راه‌حل‌های جایگزینی برای حفظ تنوع حافظه در مرجع [۶] ارائه شده است. به‌جای بازیابی انتخابی برخی از مدخل‌های حافظه، کل حافظه بعد از یک تغییر یا در طول اجرا مجدداً وارد جمعیت می‌شود. برانک [۶] هم‌چنین اشاره کرده است که حافظه به‌شدت به تنوع وابسته است و چندین ضمیمه به حافظه پایه را که باعث تقویت آن با تکنیک‌های تنوع می‌شده آزمایش کرده است. تحلیل مفصلی از تمام این راه‌کارها در مرجع [۶] آورده شده است.

نویسندگان در مرجع [۱۶] یک حافظه مشابه با حافظه [۶] با یک راه‌حل جایگزینی قدیمی ارائه کردند. این کار توسط نویسندگان مرجع [۱۷] با اضافه نمودن یک پیش‌بینی‌کننده به حافظه بسط داده شد. اگرچه این پیش‌بینی‌کننده بسیار ساده و به‌شدت وابسته به نوع مسئله بود.

نویسندگان در مرجع [۱۸] حافظه‌ای پیشنهاد دادند که مدخل‌هایی را در پاسخ به موقعیت بهترین عضو جمعیت حرکت می‌داد به‌جای این‌که کل مدخل‌ها را جابه‌جا کند. این امر کمک می‌کرد تا بهینه‌ای که حرکت ناچیزی داشت را دنبال کرد. این راه‌کار از روش حافظه برانک و یک الگوریتم ژنتیک بر روی یک مسئله نمونه عملکرد بهتری داشت.

نویسندگان در مرجع [۱۹] از استدلال موردی برای زمان‌بندی با یک الگوریتم ژنتیک استفاده کردند. زمان‌بندی پویا به‌دلیل این‌که کارهای موجود تغییر می‌یابند، یک مسئله دشوار برای استفاده از حافظه است. از مقایسه‌های بین ویژگی‌های کارها برای گذار بین دوره‌ها استفاده شده است. این مسئله بیش‌تر یک نمونه تناوبی و دوره‌ای به‌حساب می‌آمد تا یک مسئله پویای پیوسته.

روش جستجوی باکتریای جهت داده شده با ازدحام ذرات در مرجع [۲۰] پیشنهاد شده است. در این روش از ترکیب الگوریتم جستجوی باکتریای با ازدحام ذرات برای جایابی بهینه خازن‌ها و ژنراتورهای توزیع شده در شبکه‌های توزیع استفاده شده است.

بسیاری از نویسندگان دیگر برای حل مسائل بهینه‌سازی پویا از راه‌کارهای ترکیبی دیگری استفاده نموده‌اند. اغلب این روش‌ها برای حل مسائل پویا از راه‌کارهای حفظ تنوع استفاده نموده‌اند. این روش‌ها از مسئله محک قله‌های متحرک برای شبیه‌سازی محیط‌های پویا استفاده نموده‌اند[۲۶- ۲۱]. در مرجع [۲۱] از راه‌کار چندجمعیتی کرم شب‌تاب برای حفظ تنوع استفاده شده است. در مرجع [۲۲] برای دست‌یابی به جواب بهینه برای توابع چندقله‌ای در محیط‌های پویا از ایجاد تغییرات در الگوریتم تکاملی بهینه‌سازی کرم شب‌تاب استفاده شده است. در این روش از یک مدل، به‌نام مدل اجزا استفاده شده که اجازه توسعه به زیرجمعیت‌های موازی را می‌دهد.

۲-۳- مسائل محک پویا

انواع مختلفی از مسائل محک[۴] پویا برای الگوریتم‌های تکاملی در نظر گرفته شده است، شامل مسائل قله‌های متحرک [۶]، مسئله کوله‌پشتی پویا، انطباق بیت پویا، زمان‌بندی پویا، و دیگر مسائل [۲۷]. وجه تشابه بین بیش‌تر مسائل محک این است که، درحالی‌که شایستگی دورنما تغییر می‌کند، فضای جستجو تغییر نمی‌کند. به‌عنوان مثال، در مسئله قله‌های متحرک، برای هر نقطه‌ای در چشم‌انداز که با برداری از اعداد حقیقی نمایش داده شده است، همیشه راه‌حلی امکان‌پذیر است.

یک استثنای این موضوع در بین مسائل محک رایج، زمان‌بندی پویاست، که در آن وظایف در حال انتظار در طول زمان با تکمیل شدن وظایف و از راه رسیدن وظایف جدید تغییر می‌کنند. تحقیقات قبلی در زمینه الگوریتم‌های تکاملی برای مسائل زمان‌بندی پویا اصولاً بر بسط زمان‌بندی‌کننده‌هایی که برای مسائل ایستا طراحی شده بودند، متمرکز بوده‌اند [۲۸]، مسائلی با ازکارافتادگی ماشین‌ها و منابع اضافی [۲۹]، عملگرهای ژنتیک بهبودیافته [۳۰]، کاهش ابتکاری فضای جستجو [۳۱] و پیش‌بینی ایجاد زمان‌بندی‌های قوی [۳۲، ۳۳]. نویسندگان مرجع [۳۴] نشان داده‌اند که حافظه *پایه-بنیان*[۵] با داشتن مسائل زمان‌بندی ایستای مشابه مفید است.

نویسندگان مرجع [۳۵] از استدلال *پایه-بنیان* برای بهبود کیفیت زمان‌بندی برای یک مسئله، از زمان‌بندی مجدد پویا استفاده کردند. تحقیق کمی در زمینه استفاده از یک حافظه برای نگهداری کل زمان‌بندی‌های مورد استفاده در زمان‌بندی مجدد پویا انجام گرفته است. از آن‌جایی که افزودن حافظه در بهبود عملکرد الگوریتم‌های تکاملی در دیگر مسائل پویا موفقیت‌آمیز بوده است، بنابراین استفاده از حافظه برای حل مسائل زمان‌بندی پویا دور از انتظار نیست.

در اغلب مسائل بهینه‌سازی پویا، کاربرد یک حافظه صریح نسبتاً ساده است. نقاط ذخیره شده در چشم‌انداز، به‌عنوان راه‌حل عملی باقی می‌مانند، با این‌که چشم‌انداز در حال تغییر است، درنتیجه یک حافظه می‌تواند فرد را مستقیماً از جمعیت ذخیره کند [۶].

در مسائل زمان‌بندی پویا، وظایف در دسترس برای زمان‌بندی، با زمان تغییر می‌کنند و ویژگی‌های هر کار معین به دیگر وظایف وابسته است. اگر یک فرد در جمعیت، نماینده فهرست اولویت‌بندی شده از کارها باشد که قرار است به سازنده زمان‌بندی داده شود، هر حافظه‌ای که یک فرد را مستقیماً ذخیره کند، به‌سرعت اطلاعاتش برای زمان آتی بی‌ربط خواهد شد. برخی یا همه کارها در حافظه ممکن است کامل باشند، کارهایی که باقی می‌مانند ممکن است در گذشته مهم‌تر یا کم اهمیت‌تر بوده باشند، و مشخص کردن ترتیب وظایفی که پس از به‌وجود آمدن حافظه رسیده‌اند اصلاً در حافظه مورد بررسی قرار نخواهند گرفت. برای این‌گونه مسائل که هم دارای یک چشم‌انداز شایستگی پویا و هم یک فضای جستجوی در حال تغییر هستند، یک حافظه باید مقادیری را برای کارها بر حسب خواص‌شان فراهم کند تا عملیات نگاشت را برای راه‌حل‌های مشابه در حالت‌های زمان‌بندی آینده امکان‌پذیر سازد.

این مقاله حافظه‌ای برای زمان‌بندی پویا به‌نام حافظه بر پایه کلاس‌بندی ارائه می‌دهد. یک مدخل حافظه به‌جای ذخیره‌سازی لیستی از وظایف خاص، لیستی از کلاس‌بندی‌هایی را ذخیره می‌کند که می‌تواند در هر زمان به کار در حال انتظار نگاشت شود.

۲-۳-۱- زمان‌بندی وظایف پویای کار کارگاهی

مسئله زمان‌بندی وظایف پویای کار کارگاهی که برای این آزمایش‌ها استفاده شده است، بسطی از مسئله زمان‌بندی وظایف استاندارد است. در این مسئله، تعداد n وظیفه باید روی تعداد m ماشین از mt

گونه زمان‌بندی شود. به پردازش یک وظیفه در یک ماشین خاص یک عملیات اطلاق می‌شود. تعداد محدودی عملیات متمایز ot وجود دارد که به آن‌ها گونه‌های عملیات گفته می‌شود. گونه‌های عملیات توسط زمان‌های پردازش p_j و زمان‌های برپایی s_{ij} تعریف می‌شود. اگر در یک ماشین دلخواه، عملیات j پیرو عملیات i باشد، باعث به‌وجود آمدن زمان برپایی s_{ij} می‌شود.

زمان‌های برپایی وابسته به‌صورت یک توالی هستند، پس s_{ij} الزاماً مساوی با s_{ik} یا s_{kj} نیست $(i \neq j \neq k)$، و متقارن نیستند، پس s_{ij} الزاماً با s_{ji} برابر نیست. هر وظیفه از تعداد k عملیات مرتب تشکیل شده است؛ زمان کل پردازش یک وظیفه، به‌سادگی برابر است با، مجموع همه زمان‌های برپایی و زمان‌های پردازش همه عملیات وظایف. وظایف دارای سررسیدهای از پیش تعیین‌شده d_j، وزن‌های w_j و r_j زمان‌های رهایی هستند. رهاسازی وظایف یک فرآیند پواسون[۶] غیر ایستا است، پس زمان‌های *بین-ورود* متوالی وظیفه به‌طور نمایی با متوسط λ توزیع می‌شود. زمان متوسط *بین-ورود* متوالی λ با تقسیم متوسط زمان پردازش وظیفه $\bar{p}$ بر تعداد ماشین‌ها m و یک نرخ بهره‌برداری مطلوب، مثلاً U تعیین می‌شود. رابطه (۱) بیان‌گر این موضوع است.

$$\lambda = \frac{\bar{P}}{mU} \quad (۱)$$

متوسط زمان پردازش وظیفه برابر است با:

$$\bar{P} = \left(\zeta + \bar{p}\right)\bar{k} \quad (۲)$$

که در آن، ζ زمان برپایی مورد توقع، $\bar{p}$، متوسط زمان پردازش عملیات و $\bar{k}$ متوسط تعداد عملیات به‌ازای هر وظیفه است. ρ وظیفه با زمان‌های رهایی صفر وجود دارد، و وظایف جدید با گذر زمان به‌طور غیرقطعی از راه می‌رسند. زمان‌بند تا قبل از زمان رهایی وظیفه، کاملاً از آن بی‌اطلاع است. مسیریابی وظایف تصادفی بوده و عملیات به‌طور یکنواخت بر روی انواع ماشین توزیع شده‌اند؛ اگر عملیاتی نیازمند نوع خاصی از ماشین باشد، عملیات می‌تواند توسط هر ماشینی از آن‌گونه در کار کارگاهی پردازش شود. c_j زمان تکمیل آخرین عملیات است. یک هدف منفرد، تأخیر سنگین، در نظر گرفته می‌شود. تأخیر برابر با قدرمطلق تفاضل بین زمان تکمیل و سررسید وظیفه است.

$$T_j = \max(c_j - d_j, 0) \quad (۳)$$

تأخیر وزن‌دار برابر است با:

$$WT_j = w_j T_j \quad (۴)$$

به‌عنوان یک رویداد پویای اضافی، آزمایش‌ها، ازکارافتادگی (خرابی) و تعمیر ماشین را مدل‌سازی می‌کنند. فرکانس ازکارافتادگی‌های ماشین به‌وسیله درصد مدت استراحت یک ماشین تعیین می‌شود. زمان‌های تعمیر با استفاده از متوسط زمان تعمیر ε تعیین می‌شوند. زمان‌های خرابی و زمان‌های تعمیر از راه قیاس توسط زمان‌بند شناخته نمی‌شوند.

۲-۴- الگوریتم‌های تکاملی برای زمان‌بندی پویا

در نقطه‌ای دلخواه از زمان، زمان‌بند از مجموعه وظایفی که رها شده‌اند ولی هنوز کامل نشده‌اند، آگاه است. تعداد عملیات کامل نشده این وظایف مجموعه عملیات در حال انتظار نامیده خواهد شد و مطابق رابطه (۵) محاسبه می‌شود.

$$P = \{O_{j,k} | r_j \leq t, \neg complete(O_{j,k})\} \quad (۵)$$

که در رابطه (۵)، $O_{j,k}$ عملیات k از وظیفه j است. همه عملیات دارای اجبار اولویت هستند و عملیات $O_{j,k}$ نمی‌تواند شروع شود مگر زمانی‌که عملیات $O_{j,k-1}$ تکمیل شده باشد (عملیات $O_{j,k-1}$ نیز $\forall j$ کامل شده است، از آن‌جایی‌که عملیات $O_{j,0}$ هیچ سابقه‌ای (اجداد) ندارد). زمانی‌که سابقه فوری یک عملیات کامل شود، می‌توان گفت که عملیات قابل زمان‌بندی است. مجموعه عملیات قابل زمان‌بندی به‌صورت رابطه (۶) تعریف می‌شود.

$$S = \{O_{j,k} | O_{j,k} \in P, complete(O_{j,k-1})\} \quad (۶)$$

همانند اغلب رویکردهای الگوریتم تکاملی برای مسائل زمان‌بندی، راه‌حل‌ها به‌عنوان فهرست‌هایی اولویت‌بندی شده از عملیات کدگذاری می‌شوند. از آن‌جایی‌که مسئله پویا است و در آن وظایف با گذر زمان فرا می‌رسند، یک راه‌حل، فهرستی اولویت‌بندی‌شده از عملیات در حال انتظار در یک زمان خاص است. از آن‌جایی‌که عملیات در حال انتظار با زمان تغییر می‌کنند، هر فرد در جمعیت در هر گام شبیه‌ساز، به‌روزرسانی می‌شود.

در الگوریتم معروف گیفلر[۷] و تامپسون[۸] [۳۶] برای ساختن زمان‌بندی‌های فعال از یک فهرست اولویت‌بندی شده استفاده می‌شود. ابتدا، از مجموعه عملیات در حال انتظار P، مجموعه عملیات قابل زمان‌بندی S به‌وجود آورده می‌شود. از S، عملیات o' با زودترین زمان تکمیل t_c پیدا می‌شود. حال، اولین عملیات از فهرست اولویت‌بندی شده که قابل زمان‌بندی بوده و می‌تواند بر روی ماشین مشابه o' اجرا شود انتخاب گردیده و قبل از t_c شروع شود. S، به‌روزرسانی شده و تا جایی ادامه داده که همه وظایف زمان‌بندی شوند.

۱. مجموعه همه عملیات قابل زمان‌بندی S را بسازید.

۲. الف: o' را بر روی ماشین M' با زودترین زمان تکمیل t_c بیابید.

ب: عملیات $O^*_{i,k}$ را از S انتخاب کنید که زودتر از همه در فهرست اولویت‌بندی شده روی می‌دهد، می‌تواند در M' اجرا شود و می‌تواند قبل از t_c شروع شود.

۳. $O^*_{i,k}$ را به زمان‌بندی اضافه کرده و زمان شروعش را محاسبه نمایید.

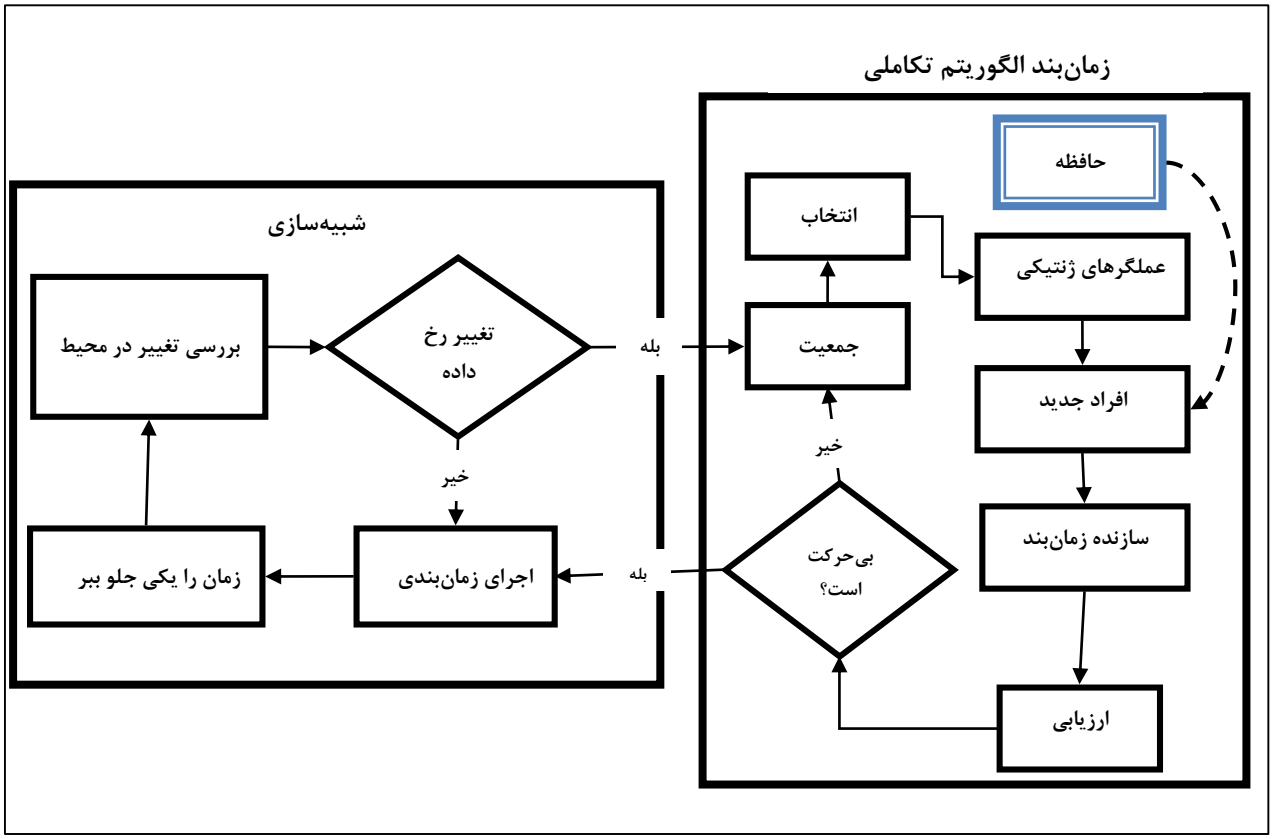
۴. $O^*_{i,1}$ را از S حذف کرده و اگر $O^*_{i,k+1} \in E$ است، آن‌گاه $O^*_{i,k+1}$ را به S اضافه کنید.

۵. تا زمانی که S خالی نیست، به مرحله ۲ بروید.

الگوریتم تکاملی با جمعیتی از ۱۰۰ نفر تولید می‌شود. عملگر کراس‌آور PPX [۷] با احتمال ۰/۶، یک عملگر جهش مبادله با احتمال ۰/۲، نخبه‌گرایی از اندازه ۱ و انتخاب مبتنی بر رتبه خطی استفاده شده است. زمان‌بندی مجدد توسط رویدادها رانده می‌شود؛ هر زمان که یک وظیفه جدید از راه می‌رسد، یک ماشین خراب باشد، یا یک ماشین تعمیر شود، الگوریتم تکاملی اجرا می‌شود تا زمانی‌که بهترین فرد در طول ۱۰ نسل یکسان جمعیت باقی بماند. شکل ۱ سیستم زمان‌بندی الگوریتم تکاملی را نشان می‌دهد.

۳- حافظه مبتنی بر کلاس‌بندی

استفاده از الگوریتم جستجوی مبتنی بر جمعیت به ما اجازه می‌دهد راه‌حل‌های خوب را از گذشته نزدیک منتقل کنیم، اما چگونه می‌توان اطلاعاتی که از راه‌حل‌های خوبی که در گذشته‌ای دورتر به‌دست آمده را به‌کار برد؟ بعضی یا همه وظایفی که در گذشته در دسترس بوده‌اند، ممکن است کامل باشند. ممکن است تعداد زیادی وظایف جدید وجود داشته باشد، یا وظیفه‌ای که دارای اولویت پایینی بوده است، اکنون می‌تواند ضروری باشد. برخلاف بسیاری از مسائل بهینه‌سازی پویا، این فضای جستجوی در حال تعویض به این معنی است که نمی‌توان افراد را به‌طور مستقیم برای فراخوانی بعدی ذخیره نمود. به‌جای آن، یک حافظه باید به ما اجازه دهد ویژگی‌های راه‌حل‌های خوب گذشته را به راه‌حل‌های موجود در محیط جدید نگاشت دهیم. این بخش یک چنین حافظه‌ای را برای زمان‌بندی پویا به‌نام حافظه بر پایه کلاس‌بندی ارائه می‌دهد. به‌جای ذخیره‌سازی لیست‌های اولویت‌بندی شده از همه عملیات، یک نمایش غیرمستقیم به‌کار گرفته می‌شود، که به ذخیره‌سازی لیستی اولویت‌بندی‌شده از کلاس‌بندی می‌پردازد. نمودار گردش فرآیند روش پیشنهادی در شکل ۱ آورده شده است. به‌منظور دسترسی به مدخل حافظه در آینده، وظایف درحال انتظار کلاس‌بندی شده و با کلاس‌بندی‌های موجود در مدخل حافظه انطباق داده می‌شوند، که منجر به تولید لیست اولویت‌بندی شده از همه عملیات می‌شود.

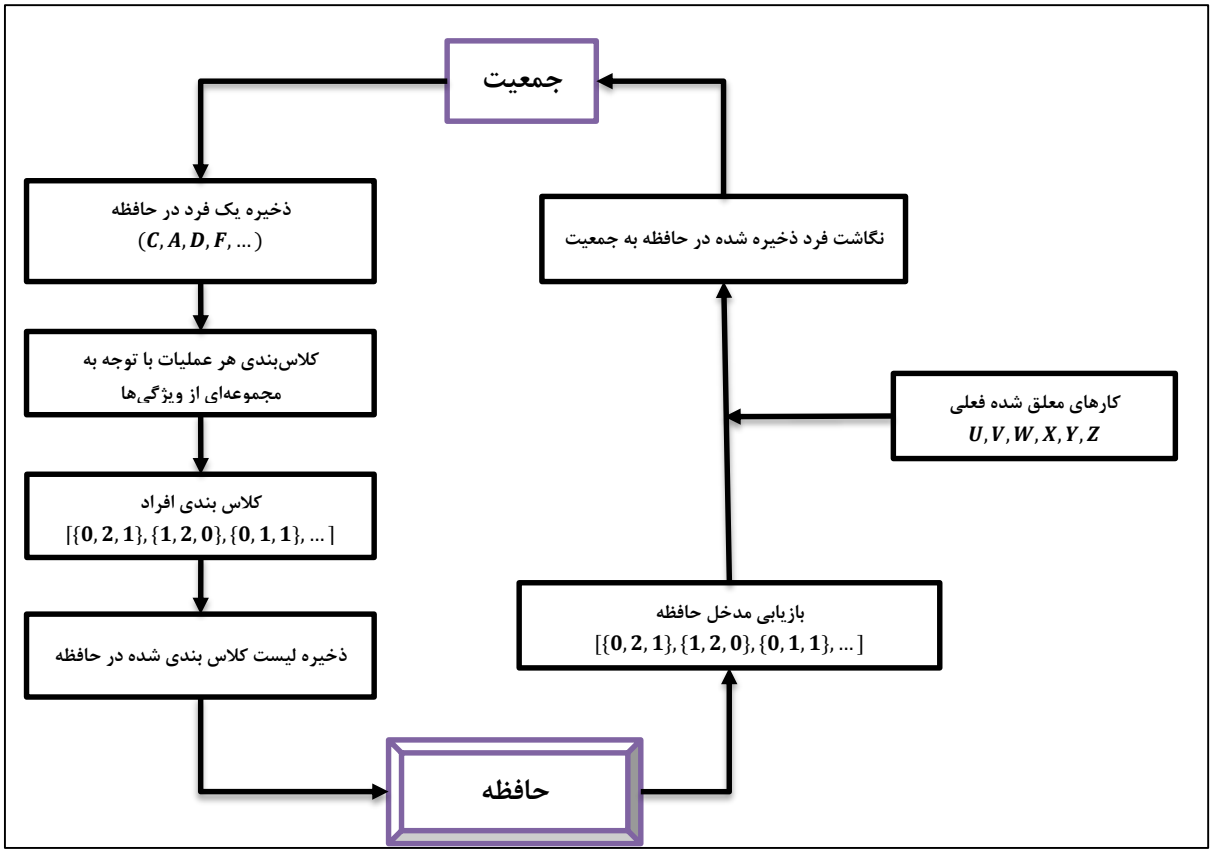


شکل ۱: سیستم زمان‌بندی الگوریتم تکاملی در شبیه‌سازی. شبیه‌ساز یک زمان‌بندی را تا جایی اجرا می‌کند که یک تغییر در محیط کشف شود. سپس زمان‌بندی‌کننده الگوریتم تکاملی یک زمان‌بندی جدید باز می‌کند تا به شبیه‌سازی بازگردد.

یک مدخل حافظه به‌طور مستقیم از لیست اولویت‌بندی شده از همه عملیات در حال انتظار ایجاد می‌شود. ابتدا، همه عملیات با توجه به مجموعه‌ای از ویژگی‌ها، رتبه‌بندی شده و سپس به‌منظور کلاس‌بندی هر عملیات، مقادیری[۹] برای هر رتبه‌بندی، با توجه به هر ویژگی تعیین می‌شوند. تعداد ویژگی‌ها a و تعداد زیرمجموعه‌ها q و نیز تعداد کل کلاس‌بندی‌های ممکن q^a می‌باشد. به‌عنوان مثال، عملیات می‌توانند با توجه به سررسید وظیفه و زمان پردازش عملیات رتبه‌بندی شوند $(a=2)$ و سپس با استفاده از مقادیری دسته‌بندی شوند $(q=4)$. این بدین معنی است که $q^a=4^2=16$ کلاس‌بندی ممکن ایجاد می‌کند. یک عملیات نمونه می‌تواند بخشی از یک وظیفه باشد که بسیار نزدیک به سررسیدش است، به‌گونه‌ای که در اولین مقدار سررسید رتبه‌بندی باشد. پردازش آن عملیات ممکن است زمان بسیار زیادی ببرد، به‌گونه‌ای که در چهارمین مقدار رتبه‌بندی زمانی پردازش عملیات، بوده باشد. این باعث ایجاد کلاس‌بندی $\{۳،۰\}$ خواهد شد.

در طول این فرآیند، لیست اولویت‌بندی شده از عملیات‌ها به لیست اولویت‌بندی شده از کلاس‌بندی‌ها تبدیل می‌شود. این لیست کلاس‌بندی‌ها، می‌تواند به‌عنوان یک مدخل حافظه ذخیره شود. به‌منظور بازیابی یک زمان‌بندی معتبر از یک مدخل حافظه، همه عملیات در حال انتظار به لیست اولویت‌بندی شده کلاس‌بندی‌هایی که در یک مدخل ذخیره شده است، نگاشت می‌شوند. هر عملیات با توجه به ویژگی‌های یکسان رتبه‌بندی می‌شود، سپس مقادیری به‌منظور دسته‌بندی هر عملیات تعیین می‌شوند. سپس، برای هرکدام از این کلاس‌بندی‌های جدید، بهترین مورد منطبق از بین کلاس‌بندی‌های موجود در مدخل حافظه یافت می‌شود.

شکل ۲ بررسی اجمالی از عملیات ذخیره‌سازی و بازیابی حافظه مبتنی بر کلاس‌بندی را نشان می‌دهد.



شکل ۲: این شکل بررسی اجمالی از عملیات ذخیره‌سازی و بازیابی حافظه مبتنی بر کلاس‌بندی را نشان می‌دهد. افرادی که در حافظه ذخیره می‌شوند کلاس‌بندی شده و لیست کلاس‌بندی آن‌ها در حافظه ذخیره می‌شود. برای بازیابی یک مدخل حافظه، مدخل به یک فرد با استفاده از ویژگی‌هایشان و وظایف در حال در انتظار نگاشت می‌شود. این فرد پس از آن می‌تواند در جمعیت درج شود.

هر عملیات در حال انتظار به‌ترتیب بر اساس موقعیت بهترین کلاس‌بندی، در لیست اولویت‌بندی حافظه قرار می‌گیرد. کلید مرتب‌سازی برای کلاس‌بندی x در مدخل حافظه y نیز j است، به‌طوری‌که:

$$\min_{j=0,\, j<|\gamma|} \sum_{i=0}^{a} |x_i - Y(j)_i| \quad (۷)$$

که در رابطه (۷)، $Y(j)$ کلاس‌بندی j در لیست y است. درصورتی‌که بیش از یک بهترین همتا (جفت) وجود داشته باشد، متوسط موقعیت‌ها به‌عنوان کلید مرتب، استفاده می‌شود. آن‌گاه، عملیات‌های در حال انتظار، توسط کلیدهای مرتب، برای ایجاد یک لیست اولویت‌بندی از عملیات‌هایی که می‌توانند به‌عنوان یک فرد برای الگوریتم تکاملی استفاده شوند، مرتب می‌شوند.

شکل ۲ نشان می‌دهد که چگونه افراد جمعیت کلاس‌بندی شده و در حافظه ذخیره می‌شوند و چگونه مدخل‌های حافظه به افراد معتبر نگاشت شده و در جمعیت درج می‌شوند. در هر نسل، یک فرد از هر مدخل حافظه ایجاد شده و آن افراد داخل جمعیت درج می‌شوند. هر نسل φ و در پایان هر دوره زمان‌بندی مجدد، یک راه‌کار جایگذاری انتخاب می‌کند که بهترین فرد در جمعیت داخل حافظه درج شود. اگر حافظه پر است، لیست کلاس‌بندی بهترین فرد، جایگزین یک مدخل حافظه جاری می‌شود. برای حفظ تنوع در حافظه، راه‌کار جایگزینی بدین‌صورت است که، دو لیست کلاس‌بندی، i و j که به‌یکدیگر نزدیک‌تر (شبیه‌تر) هستند را تعیین می‌کند که کلاس‌بندی از بهترین فرد در جمعیت و تمام مدخل‌های حافظه باشد. آن لیست j که شایستگی کم‌تری دارد به‌عنوان کاندید برای جایگزینی انتخاب می‌شود. فاصله بین دو لیست کلاس‌بندی S و T مجموع تفاضلات بین یک موقعیت کلاس‌بندی در یک لیست و موقعیت بهترین همتاهای خودش در لیست دیگر است. مثل قبل اگر بیش‌تر از یک بهترین همتا وجود داشته باشد، میانگین موقعیت‌ها استفاده می‌شود. اگر S دارای طول s و T دارای طول t باشد، آن‌گاه:

$$d = \sum_{i=0}^{s} |i - \text{best match}(S(i), T)| + \sum_{i=0}^{t} |i - \text{best match}(T(i), S)| \quad (۸)$$

تازمانی‌که کلاس‌بندی از بهترین فرد در جمعیت، کاندید جایگزینی نیست، لیست کلاس‌بندی j، با لیست کلاس‌بندی جدید جایگزین می‌شود. زمانی‌که:

$$f_j \frac{d_{ij}}{d_{max}} \le f_{best} \quad (۹)$$

که در رابطه (۹)، f_x میزان برازش[۱۰] لیست اولویت تولید شده توسط لیست کلاس‌بندی x است، و d_{ij} فاصله بین لیست‌های کلاس‌بندی i و j است، و d_{max} حداکثر فاصله ممکن است.

شکل ۳ یک مثال ساده را نشان می‌دهد. فرض کنید یک حافظه با $q = 2$ و $a = 3$ و خصوصیات زیر داریم:

وظایف سررسید (dd)، زمان پردازش عملیات (pt)، و وزن کار (W) است. در زمان ۴۰۰، ما یک لیست اولویت‌بندی شده از چهار عملیات داریم که می‌خواهیم در حافظه ذخیره کنیم. با $q = 2$ و چهار عملیات، آن دو مقادیر پایین‌تر برای هر صفت یک کلاس‌بندی از ۰ و آن دو مقادیر بالاتر یک کلاس‌بندی از ۱ دریافت می‌کنند.

بنابراین کار A یک کلاس‌بندی سررسید ۱ دارد، کلاس‌بندی زمان پردازش از ۰، و یک کلاس‌بندی وزن از ۱، برای کلاس‌بندی کلی از کلاس $(A) \rightarrow 101$. پس از کلاس‌بندی، لیست اولویت‌بندی شده $[C, B, A, D]$ به $[011,000,101,110]$ تبدیل می‌شود. این لیست کلاس‌بندی در حافظه ذخیره می‌شود. فرض کنید که در زمان ۱۰۰۰۰، ما می‌خواهیم این مدخل حافظه را برای ایجاد یک لیست اولویت‌بندی شده از چهار عملیات در حال انتظار Z, W, X, Y بازیابی کنیم. این عملیات‌های جدید کلاس‌بندی‌شده و سپس یک نمره بر اساس بهترین همتای خود درون مدخل حافظه می‌گیرند. عملیات‌های در حال انتظار به‌وسیله این نمرات، به‌منظور ایجاد یک لیست اولویت‌بندی شده جدید، که ممکن است داخل جمعیت به‌عنوان فرد جدید درج شود، مرتب می‌شوند.

حافظه مبتنی بر کلاس‌بندی اجازه می‌دهد تا کارهای جدید در کنار کارهای قدیمی‌تر مرتب شود و هنگامی‌که مدخل حافظه ایجاد شده در دسترس باشد، حافظه مبتنی بر کلاس‌بندی اطلاعات خاص در مورد عملیات را ذخیره نمی‌کند.

	Due-date	Process time	Weight
↑	450	100	3
0	500	110	5
1	800	130	7
↓	900	150	9

$At_t = 400, store[C, B, A, D]$
$A = \{dd: 800, pt = 100, w: 7\} \rightarrow 101$
$B = \{dd: 450, pt = 110, w: 5\} \rightarrow 000$
$C = \{dd: 500, pt = 130, w: 9\} \rightarrow 011$
$D = \{dd: 900, pt = 150, w: 3\} \rightarrow 110$

$$[C, B, A, D] \rightarrow [011,000,101,110] \rightarrow memory$$

	Due-date	Process time	Weight
↑	10070	50	1
0	10100	60	2
1	10400	70	5
↓	10500	80	6

$At_t = 10000, retrive[011,000,101,110]$
$W = \{dd: 10400, pt = 80, w: 1\} \rightarrow 110 \rightarrow (3)$
$X = \{dd: 10100, pt = 70, w: 5\} \rightarrow 011 \rightarrow (0)$
$C = \{dd: 10500, pt = 50, w: 6\} \rightarrow 101 \rightarrow (2)$
$D = \{dd: 10070, pt = 60, w: 2\} \rightarrow 000 \rightarrow (1)$

$$[011,000,101,110] \rightarrow [X, Z, Y, W] \rightarrow population$$

شکل ۳: یک مثال از حافظه مبتنی بر کلاس‌بندی. یک حافظه با q=2، a=3، و صفت‌های زیر معین می‌کند: کار سررسید(dd)، زمان پردازش عملیات(pt)، و وزن کار(w). در t=400، یک فرد در حافظه ذخیره شده است.

ذخیره‌سازی یک راه‌حل برای حافظه، نیاز به مرتب‌سازی n عملیات در حال انتظار توسط ویژگی‌های a است، که در $O(a.nlogn)$ انجام می‌شود. بازیابی یک راه‌حل از حافظه نیاز به کلاس‌بندی n عملیات در حال انتظار توسط a صفات دارد، هم‌چنین تطبیق هر عمل به یک موقعیت در یک مدخل حافظه از طول e، می‌تواند در $O(a.e.nlogn)$ انجام شود. اگرچه استفاده از حافظه مبتنی بر کلاس‌بندی نیاز به محاسبات اضافی روی الگوریتم ژنتیک دارد، اما این محاسبات در مقایسه با زمان لازم برای ارزیابی یک راه‌حل قابل توجه نیست.

شبه‌کد، ترکیب الگوریتم ژنتیک با حافظه مبتنی بر کلاس بندی در شکل ۴ آورده شده است.

Algorithm: function for combine GA and memory

1. $Initialize\ Pop(0))$ % Initialize the population
2. $Evaluate\ (Pop(0))$
3. $t = 1$
4. $WHILE\ (termination\ criteria\ not\ fulfilled)$
5. $Pop(t)\ =\ \emptyset$
6. $Parents(t)\ = Pop(t-1) \cup Memory$ % combine old population with memory
7. $WHILE\ |Pop(t)| < popsize$
8. $M(t)\ = select(Parents(t))$ % select individuals and copy to mating pool
9. $M'(t) = crossover(M(t))$ % perform crossover
10. $M''(t)\ = mutation(M(t))$ % perform mutation
11. $Pop(t)\ =\ Pop(t) \cup M''$ %update population
12. $Evaluate\ (Pop(t))$ %evaluate individuals in the population
13. $t\ =\ t\ +1$

شکل ۴:شبه‌کد ترکیب الگوریتم ژنتیک با حافظه پیشنهادی

۴- آزمایش‌ها و نتایج تجربی

برای بررسی اثرات حافظه مبتنی بر کلاس‌بندی، چندین روش معمول با روش پیشنهادی مقایسه شده است. از آن‌جا که زمان‌بندی مطلوب برای هر یک از موارد مسئله شناخته شده نیست، یک الگوریتم ژنتیک به‌عنوان مبنا استفاده می‌شود. الگوریتم ژنتیک با حافظه مبتنی بر کلاس‌بندی (GAm) نیز در نظر گرفته شده است. نتایج قبلی در مسائل محک مختلف مانند مسئله قله‌های متحرک نشان داد که روش‌های مبتنی بر حافظه هنگامی‌که با یک راه‌کار تنوع ترکیب می‌شوند بهتر عمل می‌کنند [۳۷-۳۹]. روش مهاجران تصادفی[11] [۴۰]، روش مهاجران تصادفی با حافظه مبتنی بر کلاس‌بندی[12] (RIM)، روش حافظه/جستجو[13][۶]، روش حافظه/جستجو با حافظه مبتنی بر کلاس‌بندی، روش محمدپور و پروین [۳۸] در دو حالت باحافظه مبتنی بر کلاس‌بندی و بدون‌حافظه مبتنی بر کلاس‌بندی و روش[۳۷] در دو حالت باحافظه مبتنی بر کلاس‌بندی و بدون‌حافظه مبتنی بر کلاس‌بندی، با روش GAm مقایسه می‌شود. دقت شود تمام روش‌های مورد مقایسه از الگوریتم ژنتیک به‌عنوان الگوریتم پایه بهره گرفته‌اند. در روش حافظه/جستجو، جمعیت به یک زیرجمعیت حافظه و یک زیرجمعیت جستجو تقسیم شده است. جمعیت حافظه می‌تواند افراد را در حافظه ذخیره کند و مدخل‌های حافظه را بازیابی نماید. برای زیرجمعیت جستجو فقط درصورتی‌که مسئله تغییر کند، جمعیت دوباره مقداردهی اولیه تصادفی می‌شود.

پارامتر تنگی سررسید[14] (τ) درصد کارهای است که برای رسیدن سررسیدشان منتظر هستند. با تغییر پارامتر تنگی سررسید، سختی مسئله ممکن است متفاوت باشد. $\rho \bar{P} \tau$ میانگین زمان تکمیل یک کار می‌باشد. r_i زمان‌های رهایی هستند که در بخش‌های گذشته توضیح داده شده است. حال سررسید کارها طبق رابطه (۱۰) محاسبه می‌شود:

$$d_j = r_i + \bar{P} + [0.2\rho \bar{P} \tau] \quad (۱۰)$$

شدت زمان برپایی برابر است با:

$$\eta = \frac{\bar{s}}{\bar{P}} \quad (۱۱)$$

که $\bar{s}$ متوسط زمان برپایی است و $\bar{P}$ میانگین زمان پردازش عملیات است. تعداد خرابی هر ماشین به‌صورت یکنواخت با متوسط تعداد خرابی‌های هر ماشین توزیع شده است و برابر با $\frac{n\bar{P}}{m}\frac{\gamma}{\varepsilon}$ است. زمان شکست (خرابی) برای هر ماشین با توزیع یکنواخت در بازه $[0, \frac{n\bar{p}}{m}]$ هستند. زمان تعمیر، یک توزیع یکنواخت در بازه $\left[\frac{1}{2}\varepsilon, \frac{3}{2}\varepsilon\right]$ هستند.

برای آزمایش‌ها در این بخش، تنظیمات زیر برای ایجاد نمونه‌های مسئله استفاده شده است. کارگاهی شامل ۲ ماشین است، هرکدام از $\mathrm{mt} = 3$ نوع ماشین، برای مجموع از $\mathrm{m} = 6$ ماشین است. پنجاه نوع عمل، با متوسط زمان پردازش عمل $\bar{P} = 100$ و زمان‌های پردازش وجود دارند و در بازه $[50,150]$ توزیع یکنواخت می‌شوند. شدت زمان برپایی $\eta = 0.5$ است، هم‌چنین متوسط زمان برپایی $\bar{s} = 50$ است. زمان برپایی توزیع یکنواخت روی $[0, 2\bar{s}]$ است. زمان برپایی برآورد شده $\varsigma = 35$ است. یک نمونه مسئله از ۵۰۰ وظیفه، هرکدام با $k = 3$ عملیات تشکیل شده است. هم‌چنین $\rho = 25$ وظیفه با زمان رهایی صفر وجود دارند. وزن وظیفه، دارای توزیع یکنواختی از بازه $[1,10]$ است. نرخ بهره‌وری $U = 0.7$ است، نرخ خرابی (شکست) $\gamma = 0.1$ است، متوسط زمان تعمیر $\varepsilon = 10\bar{P} = 1000$ است. برای کنترل سختی مسئله، تنگی سررسید از وظایف متفاوت می‌شود. چنان‌چه تنگی سررسید تغییر کند، انواع وضعیت زمان‌بندی نیز تغییر می‌کند. در این آزمایش‌ها تنگی سررسید برای $\tau \in \{0.5, 0.8, 1.1\}$ آزمایش شده‌اند، برای هر مقدار از τ، ۱۰ نمونه مسئله ایجاد شده است. به‌روزرسانی حافظه

به‌صورت نرمال اتفاق می‌افتد: جایگزینی حافظه هر $\varphi = 10$ نسل و در پایان هر دوره زمان‌بندی مجدد اتفاق می‌افتد.

شبیه‌سازی اجرا شده برای هر نوع الگوریتم تکاملی عملکردش بر روی هر یک از ۳۰ نمونه مسئله متغیر بوده است. از آن‌جاکه این یک مسئله پویا است، باعث بهبود برازش نمی‌شود اما در بهبود سرعت جستجو تأثیر دارد.

برای مجموع وزن تأخیر، از متوسط ۳۰۰ وظیفه به‌عنوان برازش استفاده می‌شود. جستجو در یک روش مشابه اندازه‌گیری می‌شود. تنها جستجوی بهینه، زمانی رخ می‌دهد که متوسط ۳۰۰ وظیفه در میان وظایف در حال انتظار در سیستم می‌باشد. در پایان هر رویداد زمان‌بندی مجدد، نیاز است ۱۰ نسل را که بهترین فرد آن‌ها بهبود نیافته است را جستجو نمود.

هر نسل در رویداد زمان‌بندی مجدد از این ۱۰ جستجو بهینه را تشکیل می‌دهند. از آن‌جا که عملکرد زمان‌بندی تعیین می‌کند که چه مدت این دوره طول می‌کشد، تعداد خرابی ماشین‌ها در طول این مدت ثابت نیست.

الگوریتم ژنتیک با یک مجموعه از قوانین اولویت-اعزام[۱۵] جهت بررسی مزایای جستجو برای این مسئله مقایسه شده است. قوانین اولویت-اعزام از شش قانون با سناریوهای مشابه برای ارزیابی زمان‌بندی الگوریتم تکاملی استفاده شده است که این قوانین عبارتند از: اولین سررسید (EDD [۴۱]، اولین سررسید وزن شده (EWDD)، کوتاه‌ترین زمان پردازش (SPT) هزینه تأخیر آشکار (ATC) [۴۲]، قانون رامان [۴۳]، و هزینه تأخیر آشکار با برپایی (ATCS) [۴۴].

دقت شود نتایج ارائه شده در جداول مختلفی در ادامه آورده شده است. در این جداول علامت (+) بدین معنی است که الگوریتم چند درصد GA را بهبود داده است و علامت (-) بدین معنی است که الگوریتم چند درصد GA را بدتر نموده است.

جدول ۱: بهبود شایستگی زمان‌بند الگوریتم ژنتیک روی یک دنباله از قوانین اولویت-اعزام.

$\tau = 0.5$	$\tau = 0.8$	$\tau = 1.1$
54.66% (+)	**68.57%** (+)	50.94% (+)

جدول ۲: بهبود شایستگی روی الگوریتم ژنتیک

	$\tau = 0.5$	$\tau = 0.8$	$\tau = 1.1$
GA with memory	**0.9%**(+)	**1.5%**(+)	**15.7%** (+)
Random immigrants	−5.7% (−)	−49.6%	−30.3% (−
Random immigrants with memory	−8.3%(−)	−51.2%(−)	13.1% (+)
memory/search	0.2% (+)	−18.0%(−)	2.1% (+)
مرجع [۳۸] بدون‌حافظه پیشنهادی	−10.1%(−)	−38.5%(−)	9.1% (+)
مرجع [۳۸] باحافظه پیشنهادی	+0.1%(+)	+1.3%(+)	13.3% (+)
مرجع [۳۷] بدون‌حافظه پیشنهادی	−6.1%(−)	−25.1%(−)	12.7% (+)
مرجع [۳۷] باحافظه پیشنهادی	+0.4%(+)	−1.1%(+)	11.6% (+)

جدول ۱ درصد بهبود شایستگی زمان‌بند الگوریتم ژنتیک را روی یک دنباله اولویت-اعزام نشان می‌دهد. در این مقاله، اهمیت آماری نتایج ارزیابی شده با استفاده از Kruskal-Wallis، با اطمینان ۹۵٪ آزمایش شده است. آزمایش Kruskal-Wallis، یک راه آنالیز واریانس به‌وسیله رتبه‌بندی، یک معادله بدون پارامتریک تجزیه و تحلیل یک طرفه کلاسیک واریانس (ANOVA) است. این معیار فرض نمی‌کند داده‌ها در یک توزیع نرمال آمده‌اند [۴۵].

در جدول ۱ زمان‌بند الگوریتم تکاملی کیفیت زمان‌بندی روی قوانین اولویت-اعزام برای همه تنگی‌های سررسید را بهبود می‌بخشد.

جدول ۲، درصد بهبود متوسط شایستگی را روی الگوریتم ژنتیک نشان می‌دهد. عملکرد GAm کمی بهتر از GA با سررسید متوسط و سخت است. وقتی سررسید سست و آزاد هستند، عملکرد GAm بهتر از GA است. با مهاجران تصادفی، برازش در تمام حالات، به‌ویژه برای متوسط سررسیدها بدتر شده است. با RIm، کارایی سررسید سخت واقعاً GAm را بهبود می‌دهد. با جستجو/حافظه، کارایی خیلی کمی برای سررسیدهای سست و محکم حاصل می‌شود، و کارایی بدتری برای تنگی متوسط ایجاد می‌شود. با روش [۳۸]، زمانی‌که از حافظه مبتنی بر کلاس‌بندی استفاده نشده است، کارایی برای سررسیدهای محکم بهبود یافته است، و کارایی بدتری برای تنگی متوسط و سست ایجاد می‌شود. با روش [۳۸]، زمانی‌که از حافظه مبتنی بر کلاس‌بندی استفاده شده است، کارایی برای سررسیدهای سست، متوسط و محکم بهبود یافته است. هم‌چنین با روش [۳۷]، زمانی‌که از حافظه مبتنی بر کلاس‌بندی استفاده شده است، کارایی برای سررسیدهای سست و محکم بهبود یافته است. جدول ۳، درصد بهبود جستجو در متوسط نسل‌های انتخابی را روی الگوریتم ژنتیک نشان می‌دهد.

GAm کاهش جستجو خوبی را برای سررسیدهای محکم و سست با بهبود خیلی کمی برای سررسیدهای متوسط نشان می‌دهد. RIm به‌طور واقع سرعت جستجو را روی GAm برای سررسیدهای متوسط بهبود می‌بخشد، اما اگر توجه شود که چقدر برازش بدتری در این نمونه هست، این بهبود واقعاً معنی‌دار نیست.

با روش [۳۸]، زمانی‌که از حافظه مبتنی بر کلاس‌بندی استفاده نشده است، سرعت جستجو روی GAm برای سررسیدهای محکم بهبود یافته است، و کارایی بدتری برای تنگی متوسط و سست ایجاد می‌شود. هم‌چنین با روش [۳۸]، زمانی‌که از حافظه مبتنی بر کلاس‌بندی استفاده شده است، سرعت جستجو روی GAm برای سررسیدهای سست، متوسط و محکم بهبود یافته است.

با روش [۳۷]، زمانی‌که از حافظه مبتنی بر کلاس‌بندی استفاده نشده است، سرعت جستجو روی GAm برای سررسیدهای محکم بهبود یافته است، و کارایی بدتری برای تنگی متوسط و سست ایجاد می‌شود. هم‌چنین با روش [۳۷]، زمانی‌که از حافظه مبتنی بر کلاس‌بندی استفاده شده است، سرعت جستجو روی GAm برای

سررسیدهای سست و محکم بهبود یافته و برای سررسیدهای متوسط، سرعت جستجو روی GAm بهبودی نیافته است.

از میان تمام روش‌ها، حافظه/جستجو تنها موردی است که نمی‌تواند سرعت جستجو را برای هر تنگی سررسید بهبود ببخشد. برای هم برازش و هم جستجو، حافظه مبتنی بر کلاس‌بندی کارایی را بهبود می‌دهد.

جدول ۳: بهبود جستجو روی الگوریتم ژنتیک

	$\tau = 0.5$	$\tau = 0.8$	$\tau = 1.1$
GA with memory	**10.0**% (+)	2.9%(+)	22.8% (+)
Random immigrants	9.4% (+)	−5.3% (−)	−13.5% (−)
Random immigrants with memory	9.5% (+)	**8.5**% (+)	**23.4**% (+)
memory/search	−18.5% (−)	−35.6%(−)	−16.8% (−)
مرجع [۳۸] بدون‌حافظه	−13.4%(−)	−28.1%(−)	+2.1% (+)
مرجع [۳۸] باحافظه	+5.2%(+)	+6.8%(+)	+11.5% (+)
مرجع [۳۷] بدون‌حافظه	+3.1%(+)	−18.5%(−)	+3.1% (+)
مرجع [۳۷] باحافظه	+5.1%(+)	−1.3%(+)	+10.3% (+)

درحالی‌که بهبودهای قابل توجه در میزان برازش، فقط برای سررسیدهای سست و آزاد آشکار بود، برای بیشتر نمونه‌های مسئله، جستجو ارتقا یافته است. GAm برای هر سه مقدار τ بهبود نشان داده است، اما کم‌ترین بهبود برای $\tau = 0.8$ است.

درحالی‌که ترکیب حافظه و تکنیک‌های تنوع نتایج خوبی برای بیشتر مسائل محک پویا به‌دست آورده است، برای این مسئله زمان‌بندی پویا هیچ‌کدام از روش‌های تنوع خوب عمل نکرده‌اند.

شاید به‌دلیل شکل محیط جستجو، تکنیک‌های تنوع بیشتر از این‌که برای یافتن نواحی با برازش بالا مفید باشند فقط ایجاد اختلال می‌کنند. حافظه/جستجو که نیمی از جمعیت خود را به جستجوی افراد جدیدی که در حافظه هستند اختصاص می‌دهد، در محیط حالت پایداری که این‌جا مورد توجه قرار گرفته دارای اشکال هستند. با این وجود، در این روش حافظه‌های از قبل سازنده که برای زمان جستجو اهمیت ندارد هنوز مفید هستند.

۵- نتیجه‌گیری

این مقاله یک روش بر مبنای الگوریتم تکاملی با حافظه برای مسئله زمان‌بندی کار کارگاهی پویا را بیان کرد. الگوریتم‌های تکاملی ارتقایافته از نظر حافظه به‌صورت گسترده‌ای برای مسائل بهینه‌سازی پویا در تحقیقات زیادی بررسی شده‌اند اما برای مسائلی مانند زمان‌بندی پویا که، برازش با جابه‌جایی‌هایی در فضای جستجو همراه است بررسی نشده است. در این مقاله، یک حافظه مبتنی بر کلاس‌بندی ارائه شده است که، نگاشت اطلاعات وظایف را برای ساخت زمان‌بندهای معتبر در نقطه دیگری از زمان ممکن می‌کند. چندین گونه الگوریتم تکاملی، با حافظه و بدون حافظه، روی نمونه‌های مسئله با سختی‌های مختلف با هم مقایسه شده است. این نتایج نشان می‌دهد که حافظه مبتنی بر کلاس‌بندی، برازش زمان‌بند و سرعت جستجو روی الگوریتم ژنتیک را بهبود می‌دهد.

نتایج این بخش نشان می‌دهد که حافظه بر پایه کلاس‌بندی می‌توانند کیفیت زمان‌بندی و سرعت جستجو را در الگوریتم ژنتیک بهبود بخشد. این نتایج هم‌چنین نشان می‌دهند، درحالی‌که ترکیب حافظه و تکنیک‌های تنوع نتایج خوبی برای بیشتر مسائل محک پویا به‌دست آورده است، برای این مسئله زمان‌بندی پویا هیچ‌کدام از روش‌های تنوع خوب عمل نکرده‌اند.

مقایسه کارایی حافظه مبتنی بر کلاس‌بندی با استفاده از ویژگی‌های مختلف، انواع اندازه چارک‌ها، حافظه‌های بزرگ‌تر یا سایر تغییرات در ساختار حافظه توجه بیشتری به‌سمت پتانسیل حافظه‌های مبتنی بر کلاس‌بندی برای زمان‌بندی پویا جلب می‌کند. انواع دیگر حافظه را نیز می‌توان برای به‌کارگیری راه‌هایی برای حفظ اطلاعات در مورد زمان‌های برپایی، تغییرات دوره‌ای در ترکیب انواع عملگر در طول زمان یا سایر اطلاعات که حافظه نمی‌تواند آن‌ها را به‌راحتی به‌دست آورد ساخت.

برای کارهای آتی پیشنهاد می‌شود، روش پیشنهادی را برای مسائل محک پویای دیگر از جمله مسئله قله‌های متحرک آزمایش نمود و کارآیی این روش را برای دیگر مسائل بهینه‌سازی پویا سنجید.

مراجع

[1] S. Yang, “Explicit memory schemes for evolutionary algorithms in dynamic environments,” In S. Yang, Y.-S. Ong, and Y. Jin, editors, “Evolutionary Computation in Dynamic and Uncertain Environments,” volume 51 of Studies in Computational Intelligence, pages 3-28. Springer-Verlag, 2007.

[2] S. Yang, “ Genetic algorithms with elitism-based immigrants for changing optimization problems,” In Applications of Evolutionary Computing, Lecture Notes in Computer Science 4448, pages 627–636, 2007.

[3] S. Yang, “Memory-based immigrants for genetic algorithms in dynamic environments,” In H.-G. Beyer, editor, In Proceedings of the Seventh International Genetic and Evolutionary Computation Conference (GECCO 2005), volume 2, pages 1115-1122. ACM Press. 2005.

[4] J. Branke, “Memory enhanced evolutionary algorithms for changing optimization problems,” In Proceeding of the 1999 IEEE Congress on Evolutionary Computation (CEC-1999), Volume 3, pages 1875–1882, 1999.

[5] J. Holland, “Adaptation in Natural and Artificial Systems,” University of Michigan Press, Ann Arbor, MI, 1975.

[6] J. Branke. “Evolutionary Optimization in Dynamic Environments,” Volume 3 of Genetic algorithm and evolutionary computation, Kluwer Academic Publisher, Massachusettes, USA, 2001.

[7] E. Goldberg and E. Smith. “Nonstationary function optimization using genetic algorithm with dominance and diploidy,” In Proceeding of the second International Conference on Genetic Algorithms and their Application, Hillesdale, NJ, USA, 1987, pages 59–68, 1987.

[8] J. Lewis, E. Hart, and G. Ritchie. “A comparison of dominance mechanisms and simple mutation on non-stationary problems,” In Proceeding of the 5th Int. Conf. on Parallel Problem Solving from Nature, pages 139–148, 1998.

[9] K. Peow Ng and K. Cheong Wong. “A new diploid scheme and dominance change mechanism for non-stationary function

optimization", In International Conference on Genetic Algorithms, pages 159–166, 1995.
[10] C. Ryan. The degree of oneness. In European Conference on Artificial Intelligence Workshop on Genetic Algorithms, 1994.
[11] C. Ryan. "Diploidy without dominance," In Nordic Workshop on Genetic Algorithms, pages 45–52, 1997.
[12] P. Collard, C. Escazut, and A. Gaspar. "An evolutionary approach for time dependent optimization," International Journal on Artificial Intelligence Tools, 6(4):665–695, 1997.
[13] A. Gaspar and P. Collard. "Time dependent optimization with a folding genetic algorithm," In Proceeding of the International Conference on Tools for Artificial Intelligence, pages 207–214, 1997.
[14] S. Yang. "Non-stationary problems optimization using the primal-dual genetic algorithm," In Proceeding of the 2003 Congress on Evolutionary Computation, vol. 3, pages 2246–2253, 2003.
[15] L. Connie, Ramsey and J. Grefenstette. "Case-based initialization of genetic algorithms," In Proceeding of the 5th Int. Conf. on Genetic Algorithms, pages 84–91, 1993.
[16] J. Eggermont, T. Lenaerts, S. Poyhonen, and A. Termier. "Raising the dead: Extending evolutionary algorithms with a case-based memory," In European Conference on Genetic Programming, pages 280–290, 2001.
[17] J. Eggermont and T. Lenaerts. "Dynamic optimization uses evolutionary algorithms with a case-based memory," In Belgium-Netherlands Conference on Artificial Intelligence, pages 107–114, 2002.
[18] C. Bendtsen and T. Krink. "Dynamic memory model for non-stationary optimization", In Proceeding of the 2002 Congress on Evol. Comput, pages 145–150, 2002.
[19] G. Chryssolouris and V. Subramaniam. "Dynamic scheduling of manufacturing job shops using genetic algorithms," Journal of Intelligent Manufacturing, 12:281–293, 2001.
[۲۰] رحمت‌الله هوشمند، حسین محکمی، امین خدابخشیان، "روشی جدید در جایابی بهینه خازن‌ها و ژنراتورهای توزیع شده در شبکه‌های توزیع با استفاده از الگوریتم جستجوی باکتریای جهت داده شده با pso". مجله مهندسی برق دانشگاه تبریز. جلد ۳۹. شماره ۲. ۱۳۸۹.
[21] FB. Ozsoydan and Baykasoglu. "A multi population firefly algorithm for dynamic optimization problems," In Proceeding of the 2015 IEEE International Conference on Evolving and Adaptive Intelligent Systems (EAIS), pages 1-7., Dec.2015.
[22] B. Nasiri and M. R. Meybodi, "Speciation based firefly algorithm for optimization in dynamic environments," International Journal of Artificial Intelligence, vol. 8, pages. 118-132, 2012.
[23] X. Chen, D. zhang, X. Zeng, "A Stable Matching-Based Selection and Memory Enhanced MOEDA/D for Evolutionary Dynamic Multiobjective Optimization," Tools with Artificial intelligence (ICTAI), IEEE 27th International Conference Artificial intelligence, pages 478-485, 2015.
[24] R. Mukherjee, G. R. Patra, R. Kundu, and S. Das, "Cluster-based differential evolution with Crowding Archive for niching in dynamic environments," Information Sciences, vol. 267, pages. 58-82, 2014.
[25] J. K Kordestani, A. Rezvanian, and M. R. Meybodi, "CDEPSO: a bi-population hybrid approach for dynamic optimization problems," Applied intelligence, vol. 40, pages. 682-694, 2014.
[26] T.T. Nguyen, "Solving dynamic optimization problems by combining Evolutionary Algorithms with KD-Tree," Soft Computing and Pttern Recogonition (SoCPaR), International Conference, pp. 247-252, 2013.
[27] Y. Jin and J. Branke. "Evolutionary optimization in uncertain environments—a survey," IEEE Transactions on Evolutionary Computation, 9(3):303–317, 2005.
[28] W. Richard, Conway, William, L. Maxwell, and L. Miller. Theory of Scheduling. Dover, 1967.
[29] G. Chryssolouris and V. Subramaniam. "Dynamic scheduling of manufacturing job shops using genetic algorithms," Journal of Intelligent Manufacturing, 12:281–293, 2001.
[30] D. Curtis. Adaptive control software. Technical Report HRTS-04-037, U.S. Department of Transportation, Federal Highway Administration, 2004.
[31] D. C. Mattfeld and C. Bierwirth. "An efficient genetic algorithm for job shop scheduling with tardiness objectives," European Journal of Operations Research, 155:616–630, 2004.
[32] J. Branke and D. C.Mattfeld. "Anticipatory scheduling for dynamic job shop problems," In Parallel Prob. Solving from Nature PPSN VI, pages 253–262, Berlin, Springer Verlag., pages 3–10, 2002.
[33] J. Branke and D. C. Mattfeld. "Anticipation and flexibility in dynamic scheduling," International Journal of Production Research, 43(15):3103–3129, 2005.
[34] S. J. Louis and J. McDonnell. "Learning with case-injected genetic algorithms", IEEE Transactions on Evolutionary Computation, 8(4):316–328, 2004.
[35] K. Miyashita and K. Sycara. "CABINS: a framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair", Artificial Intelligence, 76(1-2):377–426, 1995.
[36] B. Giffler and G. L. Thompson. "Algorithms for solving production scheduling problems", Operations Research, 8(4):487–503, 1960.
[۳۷] مجید محمدپور و حمید پروین."الگوریتم ژنتیک آشوب‌گونه مبتنی بر حافظه و خوشه‌بندی برای حل مسائل بهینه‌سازی پویا". مجله مهندسی برق دانشگاه تبریز. جلد ۴۶. شماره ۳. پاییز ۱۳۹۶.
[38] M. Mohammadpour, H. Parvin. "Genetic Algorithm Based on Explicit Memory for Solving Dynamic Problems," In Journal of Advances in Computer Research Sari Branch Islamic Azad University. (Vol. 7, No. 2, May 2016), Pages: 53-68., 2016.
[39] F. Tian, N. Yang, J. Chen, "memory based differential algorithm for dynamic constrained optimization problems," In Proceeding of the 11th International Conference on Computatioanal Intelligounce and Security (CSI), Pages: 30-33, 2015.
[40] J. Grefenstette. "Genetic algorithms for changing environments," In Parallel Problem Solving from Nature, pages 137–144, 1992.
[41] W. Conway, L. Maxwell, and W. Miller. Theory of Scheduling. Dover, 1967.
[42] P.J. Vepsalainen and E. Morton. "Priority rules for job shops with weighted tardiness costs," Management Science, 33(8):1035–1047, 1987.
[43] N. Raman, V. Rachamadugu, and F. Brian Talbot. "Real-time scheduling of an automated manufacturing center," European Journal of Operational Research, 40(2):222–242, 1989.
[44] G. McLachlan and D. Peel. *Finite mixture models*. Wiley, 2000.
[45] W. Corder and I. Foreman. "Nonparametric Statistics for Non-Statisticians," Wiley, 2009.

زیرنویس‌ها

[1] Memory
[2] Implicit memory
[3] Explicit memory
[4] Benchmark
[5] Case-base memory
[6] Poisson
[7] Giffler
[8] Thompson
[9] quantiles
[10] fitness
[11] Random immigrants
[12] Random immigrants with memory
[13] memory/search
[14] due-date tightness
[15] priority-dispatch