مجله تحقیق در عملیات در کاربردهای آن، سال هجدهم، شماره دوم (پیاپی ۶۹)،تابستان ۰۰، ص ص ۲۳-۱
شاپا ۷۲۸۶-۲۲۵۱

ارایه الگوریتم‌های کارآمد برای زمان‌بندی تولید به‌هنگام در شرایط پردازش انباشته

طاها کشاورز [1*] ، ندا رفیعی پارسا [2]

۱- استادیار، دانشکده فنی و مهندسی، گروه مهندسی صنایع، دانشگاه سمنان، ایران

۲- استادیار، دانشکده فنی و مهندسی، گروه مهندسی صنایع، دانشگاه آزاد اسلامی واحد تهران مرکزی، ایران

رسید مقاله: ۲۶ بهمن ۱۳۹۸

پذیرش مقاله: ۵ اسفند ۱۳۹۹

چکیده

در این مقاله مساله زمان‌بندی تولید به‌هنگام در یک ماشین پردازنده انباشته مورد بررسی قرار می‌گیرد. ماشین‌های پردازنده انباشته قادرند به طور هم‌زمان بیش از یک کار را پردازش کنند و کاربردهای فراوانی در صنایع تولید نیمه هادی دارند. در راستای تامین اهداف تولید به‌هنگام، معیار عملکرد در نظر گرفته شده، کمینه کردن توام هزینه زودهنگامی و دیرهنگامی کارهاست که یک معیار مورد پسند برای تولیدکننده و مشتری است. با توجه به NP-hard بودن مساله مفروض، هدف یافتن جواب نزدیک به بهینه برای مساله در اندازه‌های صنعتی با استفاده از الگوریتم‌های فراابتکاری است. دو الگوریتم برای مساله ارایه می‌شود. الگوریتم نخست، الگوریتم ژنتیک ترکیبی و الگوریتم دوم مبتنی بر الگوریتم جستجوی تطبیقی تصادفی حریصانه است. در هر دو الگوریتم از یک الگوریتم برنامه‌ریزی پویا برای زمان‌بندی انباشته‌ها استفاده می‌شود. نتایج آزمایشات محاسباتی نشان‌دهنده کارایی الگوریتم‌های پیشنهادی برای مسایل با ابعاد بزرگ است به نحوی که متوسط خطای الگوریتم ژنتیک ترکیبی برابر ۶/۸۲٪ و این مقدار برای الگوریتم جستجوی تطبیقی تصادفی حریصانه برابر ۱۱/۶۴٪ است. همچنین کارایی الگوریتم‌های پیشنهادی برای مسایل با کارهای با اندازه کوچک قابل توجه‌تر از مسایل با کارهای دارای اندازه بزرگ است.

کلمات کلیدی: ماشین پردازنده انباشته، تولید به‌هنگام، زود هنگامی و دیرهنگامی، برنامه‌ریزی پویا، الگوریتم‌های فراابتکاری

۱ مقدمه

ماشین‌های پردازنده انباشته قابلیت انجام نوع خاصی از عملیات، به طور هم‌زمان بر روی بیش از یک کار و یا به عبارتی یک انباشته از کارها را دارند. در مدل‌های زمان‌بندی ماشین‌های پردازنده انباشته چند کار به طور هم‌زمان تحت عملیات ماشین قرار می‌گیرند، همچنین ظرفیت ماشین محدود بوده و با توجه به حجم اقلام تشکیل‌دهنده

* عهده‌دار مکاتبات

آدرس الکترونیکی: taha_keshavarz@semnan.ac.ir

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]

[DOR: 20.1001.1.22517286.2021.18.2.7.2]

[DOI: 10.52547/jamlu.18.2.1]

یک انباشته، قابلیت انجام عملیات بر روی تعداد محدودی از اقلام وجود دارد. به عنوان نمونه، عملیات فر درون‌سوز در تولید صفحه‌های مدارات چاپی، داخل فرهایی انجام می‌شود که چندین گروه از صفحه‌ها را می‌توانند در خود جای داده و تحت عملیات حرارتی قرار دهند. در اینجا فرها نقش ماشین‌های پردازنده انباشته را بازی می‌کنند. مدل‌های زمان‌بندی ماشین‌های پردازنده انباشته در صنایع تولیدی مختلف کاربرد دارند، اما حجم وسیعی از تحقیقات صورت گرفته منحصرا به کاربرد این مدل‌ها در صنایع تولید نیمه هادی مربوط می‌شود. این امر از آن جهت است که خاستگاه این مدل‌ها در صنایع تولید نیمه هادی‌ها بوده است.

از طرفی، مدل‌های زمان‌بندی با درنظر گرفتن توام هزینه‌های زودهنگامی و دیرهنگامی در راستای تامین اهداف تولید به هنگام از اهمیت بالایی برخوردار بوده و یک معیار مورد پسند برای تولیدکننده و مشتری است. امروزه استفاده از مدل های زمان بندی تولید به‌هنگام، به علت اهمیت و ارتباط کاربردی آن‌ها و همچنین به عنوان ابزاری برای کاهش قیمت تمام شده محصولات به طور قابل ملاحظه‌ای رو به افزایش است. درحقیقت، مسایل زمان‌بندی با جرایم زودهنگامی و دیرهنگامی، با مفاهیمی نظیر تولید به‌هنگام و مدیریت زنجیره تأمین سازگار هستند. در سیستم‌های تولید به‌هنگام برای کارهایی که قبل از موعد تکمیل می‌شوند، هزینه‌هایی از قبیل هزینه‌های نگهداری در انبار، فاسد شدن محصولات فاسد شدنی، از مد افتادن و مالیات در نظر گرفته می شود. علاوه بر آن کارهایی که دیرتر از موعد تکمیل می شوند نیز هزینه هایی نظیر فروش از دست رفته، ایجاد تراکم در حمل، نارضایتی مشتریان و کاهش اعتبار را به سازمان تحمیل می کنند. بنابراین، بهترین زمان‌بندی، زمان‌بندی است که در آن عملیات همه کارها دقیقاً در موعدهای تحویل‌شان به اتمام برسد.

با توجه به کاربردهای وسیع ماشین های پردازنده انباشته مخصوصا در صنایع تولید نیمه هادی و همچنین اهمیت زمان‌بندی تولید به‌هنگام، هدف این پژوهش ارایه الگوریتم‌های فراابتکاری جهت تولید جواب‌های نزدیک بهینه برای مساله یک ماشین پردازنده انباشته با فرض اندازه کارهای غیریکسان به منظور کمینه کردن مجموع هزینه‌های زودهنگامی و دیرهنگامی کارها است. نوآوری دیگر این پژوهش، ارایه یک الگوریتم برنامه‌ریزی پویا برای تعیین توالی انباشته‌هاست.

در ادامه و در بخش دوم، تحقیقات پیشین مسایل زمان‌بندی پردازنده انباشته مورد بررسی قرار می‌گیرد. در بخش سوم، مشخصات و فرضیات مساله بیان می‌شود. در بخش چهارم الگوریتم‌های پیشنهادی ارایه شده و نتایج آزمایشات محاسباتی آن در بخش پنجم گزارش می‌شود. در نهایت، در بخش ششم، به جمع‌بندی مطالب ارایه شده و تعیین زمینه‌های پژوهش‌های آتی پرداخته می‌شود.

۲ تحقیقات پیشین

نخستین تلاش صورت گرفته در زمان‌بندی ماشین‌های پردازنده انباشته با فرض وجود کارهای با اندازه غیریکسان را می‌توان به دابسون و نامبیمادم [۱] نسبت داد. اما این مدل‌ها به طور جدی پس از این‌که اوزوی [۲] کاربرد آن‌ها را در صنایع تولید نیمه هادی‌ها مطرح کرد، مورد توجه محققین قرار گرفت. وی در مقاله خود به حداقل‌سازی زمان تکمیل آخرین کار ($C_{\max}$) و مجموع زمان تکمیل کارها ($\sum C_j$)پرداخته است. دوپونت و جولای [۳]

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

نیز برای مساله حداقل‌سازی $C_{\max}$ دو الگوریتم ابتکاری ارایه کرده‌اند که یکی از آن‌ها بر مبنای اصلاح الگوریتم‌های ارایه شده توسط اوزوی [۲] است و دیگری بر مبنای تعیین اقلام یک انباشته با استفاده از حل یک مساله کوله‌پشتی است. نتایج محاسباتی نشان داده‌اند که هر دو الگوریتم از عملکرد بهتری نسبت به الگوریتم‌های قبلی برخوردارند. جولای و دوپونت [۴] برای مساله حداقل‌سازی $\sum C_j$ تعدادی الگوریتم ابتکاری ارایه کرده و از طریق محاسبات کامپیوتری نشان داده‌اند که عملکرد این الگوریتم‌ها، در مقایسه با الگوریتم‌های ارایه شده توسط اوزوی [۲]، بهتر است.

ملوک و همکاران [۵] یک الگوریتم شبیه‌سازی تبرید برای مساله حداقل‌سازی $C_{\max}$ ارایه کرده‌اند. آن‌ها نشان داده‌اند که روش ارایه شده در مقایسه با جواب‌های حاصل از حل مدل ریاضی مساله با استفاده از نرم افزار CPLEX نتایج بهتری را در زمان کم‌تر به دست می‌دهد. دامودارن و همکاران [۶] نیز یک الگوریتم ژنتیک ساده برای مساله حداقل‌سازی $C_{\max}$ ارایه کرده و عملکرد آن را با الگوریتم شبیه‌سازی تبرید ارایه شده توسط ملوک و همکاران [۵] مقایسه کرده‌اند. برای این مساله حسین‌زاده کاشان و همکاران [۷] یک الگوریتم ژنتیک مبتنی بر نمایش کروموزومی بر اساس کلیدهای تصادفی و یک الگوریتم ژنتیک ترکیبی که از یک نمایش کروموزومی ابداعی بهره می‌برد ارایه کرده‌اند. نتایج محاسبات نشان می‌دهد که الگوریتم ژنتیک ترکیبی ارایه شده بسیار کارا عمل نموده و جواب‌های بسیار بهتری را نسبت به الگوریتم شبیه‌سازی تبرید ارایه شده توسط ملوک و همکاران [۵] و الگوریتم ژنتیک مبتنی بر نمایش کروموزومی بر اساس کلیدهای تصادفی فراهم می‌کند.

رفیعی پارسا و همکاران [۸] مساله حداقل‌سازی $C_{\max}$ را با فرض وجود کارهای با اندازه غیریکسان در نظر گرفتند. آن‌ها نخست با استفاده از روش تولید ستون یک حد پایین قوی برای مساله ارایه کردند، سپس الگوریتم ابتکاری برای تولید جواب اولیه روش تولید ستون توسعه داده شده، و در نهایت از حد پایین به دست آمده در بدنه الگوریتم شاخه و قیمت برای به دست آوردن جواب بهینه استفاده کرده‌اند. نتایج آن‌ها نشان‌دهنده برتری الگوریتم‌های ارایه شده توسط آن‌ها در مقایسه با الگوریتم‌های موجود در ادبیات است.

چنگ و همکاران [۹] یک مدل فازی برای مساله حداقل‌سازی $C_{\max}$ ارایه کرده‌اند. عدم قطعیت پردازش کارها روی ماشین به عنوان پارامتر فازی در مدل درنظر گرفته شده است. آن‌ها همچنین یک الگوریتم فراابتکاری مبتنی بر بهینه‌سازی مورچگان برای مساله ارایه کردند. چن و همکاران [۱۰] الگوریتمی بر اساس تکنیک‌های خوشه‌بندی برای مساله حداقل‌سازی $C_{\max}$ ارایه کردند. ماتیراجان و همکاران [۱۱] مساله حداقل‌سازی مجموع وزنی هزینه‌های دیرهنگامی ($\sum w_j T_j$) با فرض وجود زمان دسترسی برای کارها را در نظر گرفتند. آن‌ها چند الگوریتم ابتکاری و همچنین الگوریتم شبیه‌سازی تبرید برای این مساله ارایه کردند. ونگ [۱۲] الگوریتم ابتکاری دومرحله‌ای برای به دست آوردن حل تخمینی همین مساله ارایه کرد. مالاپرت و همکاران [۱۳] مساله حداقل‌سازی بیشترین تاخیر ($L_{\max}$) را مورد بررسی قرار دادند. آن‌ها از رویکرد برنامه‌ریزی محدودیت[1] برای حل مساله استفاده کرده‌اند و نتایج آن‌ها نشان‌دهنده برتری روش ارایه شده می‌باشد. کوروگا و همکاران [۱۴]

[1] Constraint programming

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

اخیرا یک الگوریتم جستجوی همسایگی برای کمینه کردن مجموع وزنی هزینه‌های دیرهنگامی روی یک ماشین پردازش انباشته ارایه کرده‌اند.

دامودارن و همکاران [۱۵] یک الگوریتم ابتکاری تصادفی برای مساله حداقل‌سازی $C_{\max}$ ارایه کردند و عملکرد الگوریتم ارایه شده را با الگوریتم‌های ژنتیک و شبیه‌سازی تبرید مقایسه کرده‌اند. جیا و لئونگ [۱۶] مساله حداقل‌سازی $C_{\max}$ را به صورت حداقل‌سازی فضای هدررفته مدل‌سازی کردند. آن‌ها با ترکیب الگوریتم ابتکاری مبتنی بر فضای هدررفته و الگوریتم مورچگان، یک الگوریتم بهبودیافته برای مساله ارایه کردند. همچنین از یک الگوریتم جستجوی محلی برای یافتن جواب‌های بهتر استفاده کردند. السلامه [۱۷] با استفاده از رویکرد کلونی زنبورعسل، یک الگوریتم فراابتکاری برای همین مساله ارایه کرد. کابو [۱۸] برای مساله حداقل‌سازی $L_{\max}$ یک روش جستجوی همسایگی جدید معرفی کرده‌است. نتایج محاسباتی کارایی الگوریتم ارایه شده را نشان می‌دهد.

رفیعی پارسا و همکاران [۱۹] پس از مدلسازی مساله حداقل‌سازی مجموع زمان تکمیل کل کارها با استفاده از رویکرد شبکه عصبی، الگوریتم ترکیبی مبتنی بر شبکه عصبی برای این مساله ارایه کرده‌اند. این الگوریتم، ترکیبی از یک الگوریتم ابتکاری و رویکرد یادگیری شبکه عصبی است. بلدار و همکاران [۲۰] مساله حداقل‌سازی مجموع زمان تکمیل کارها با درنظر گرفتن زمان دسترسی برای کارها را مورد بررسی قرار دادند و یک الگوریتم بهینه‌سازی مبتنی بر آموزش-یادگیری ارایه داده‌اند. در میان تحقیقات داخلی می‌توان به پژوهش موسوی و همکاران [۲۱] در زمینه زمان‌بندی یکپارچه تولید سفارشات در محیط تولیدی تک ماشینه اشاره کرد. بهشتی نیا و حسنی [۲۲] نیز به ارایه ترکیبی از الگوریتم‌های فراابتکاری ژنتیک و شبیه‌سازی تبرید برای حل مساله زمان‌بندی پرداخته‌اند.

در خصوص مساله زمان‌بندی ماشین پردازنده انباشته با فرض اندازه غیریکسان برای کارها و هدف کمینه‌کردن هزینه‌های زودهنگامی و دیرهنگامی کارها که در راستای اهداف تولید به‌هنگام به شمار می‌رود، تحقیقات اندکی انجام شده است. لی و همکاران [۲۳] در سال ۲۰۱۵ به ارایه الگوریتمی فراابتکاری به نام GAMARB برای مساله مذکور پرداخته‌اند. رفیعی پارسا و همکاران [۲۴] در سال ۲۰۱۷ این مساله را مورد بررسی قرار دادند و بعد از ارایه مدل ریاضی، یک الگوریتم برنامه‌ریزی پویا ارایه کردند و بر اساس آن چند الگوریتم ابتکاری توسعه داده‌اند. همچنین یک الگوریتم شاخه و حد بر اساس حد پایین کارا ارایه شده، برای یافتن جواب بهینه مساله ارایه کرده‌اند. با توجه به اهمیت این مساله و همچنین کاربردهای آن، هدف این مقاله ارایه الگوریتم‌های تولید جواب نزدیک بهینه برای مساله مذکور است. نوآوری‌های اصلی این پژوهش، ارایه دو الگوریتم فراابتکاری برای حل مساله و همچنین ارایه یک الگوریتم برنامه‌ریزی پویا برای تعیین توالی انباشته‌هاست.

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

۳ مشخصات مساله

در این پژوهش، مساله یک ماشین پردازنده انباشته با فرض اندازه کارهای غیریکسان به منظور کمینه کردن مجموع هزینه‌های زودهنگامی و دیرهنگامی کارها مورد بررسی قرار می‌گیرد. مفروضات حاکم بر مساله مورد بررسی به شرح زیر است:

۱. تعداد n کار برای زمان‌بندی وجود دارند که همگی در لحظه صفر برای زمان‌بندی بر روی یک ماشین پردازنده انباشته در دسترس هستند. کلیه کارها با یکدیگر سازگار بوده و فرض می‌شود تنها متعلق به یک خانواده هستند (زمان‌بندی بدون خانواده‌های کارها).

۲. برای هر کار j مقدار کمیتی معادل s_j واحد از منابع محدود مورد نیاز است، در اینجا منبع مذکور ظرفیت ماشین و مقدار s_j برابر اندازه کار j در نظر گرفته می‌شود.

۳. زمان مورد نیاز جهت انجام عملیات کارها (زمان پردازش) بر روی ماشین معلوم است. زمان پردازش کار j برابر با p_j است.

۴. مقادیر زمان‌های پردازش و اندازه کارها از قبل مشخص و قطعی هستند.

۵. موعد تحویل کارها یکسان و برابر d است. موعد تحویل دور (فرصت‌دار)[1] است، به عبارت دیگر می‌توان فرض کرد که $\sum_{j=1}^{n} p_j \leq d$.

۶. با شروع عملیات ماشین روی یک انباشته، توقف عملیات امکان‌پذیر نبوده و پیش از اتمام عملیات ماشین، هیچ کاری نمی‌تواند به انباشته اضافه و یا از آن خارج شود.

۷. حداکثر ظرفیت ماشین برای انجام عملیات هم‌زمان روی کارها برابر B است. به عبارت دیگر مجموع اندازه کارهای متعلق به یک انباشته نمی‌بایست از مقدار B متجاوز شود، همچنین فرض می‌شود کلیه کارها دارای اندازه‌ای کوچک‌تر یا مساوی با ظرفیت ماشین هستند.

۸. زمان شروع و زمان ختم عملیات تمامی کارهای متعلق به یک انباشته با یکدیگر یکسان است. زمان پردازش انباشته توسط ماشین برابر زمان پردازش کاری است که در میان کارهای متعلق به انباشته، زمان پردازش مورد نیاز آن طولانی‌تر است.

۹. خرابی ماشین مجاز نیست.

به منظور سادگی، در ادامه از عبارت *SUMET* برای اشاره به مساله زمان‌بندی یک ماشین پردازنده انباشته تحت مفروضات فوق استفاده می‌شود. در مساله زمان‌بندی ماشین‌های پردازنده انباشته، دو دسته از تصمیمات متفاوت اما مرتبط با یکدیگر می‌بایست اتخاذ شود:

۱. تصمیم‌گیری در مورد نحوه انباشته‌سازی.

۲. تصمیم‌گیری در مورد نحوه زمان‌بندی و تعیین توالی انباشته‌های تشکیل شده.

[1] Loose due date

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

وابستگی دو دسته تصمیم فوق از آنجاست که زمان پردازش انباشته وابسته به کارهای درون انباشته است. به محض اینکه یک طرح انباشته‌سازی تعیین شد، با در نظر گرفتن هر انباشته در قالب یک کار، مسایل زمان‌بندی ماشین‌های پردازنده انباشته به مسایل زمان‌بندی کلاسیک تبدیل می‌شوند.

براکر و همکاران [۲۵] ثابت کرده‌اند که تمامی مسایل زمان‌بندی ماشین پردازنده انباشته با معیارهای عملکرد مبتنی بر موعد تحویل، NP-hard هستند. بنابراین مساله *SUMET* نیز NP-hard است.

٤ الگوریتم‌های فراابتکاری جهت یافتن جواب نزدیک بهینه

در این بخش، هدف یافتن جواب‌های نزدیک بهینه برای مساله در اندازه‌های بزرگ و صنعتی است. با توجه به NP-hard بودن مساله *SUMET*، در این بخش دو الگوریتم فراابتکاری برای مساله ارایه می‌شود. الگوریتم نخست، الگوریتم ژنتیک ترکیبی[1] (HGA) و الگوریتم دوم مبتنی بر الگوریتم جستجوی تطبیقی تصادفی حریصانه[2] (GRASP) است.

٤-١ الگوریتم ژنتیک ترکیبی

در این بخش الگوریتم ژنتیک ترکیبی برای مساله *SUMET* ارایه می‌شود. در الگوریتم ارایه شده، از رویه جستجوی محلی به منظور تقویت الگوریتم ژنتیک استفاده شده است. در ادامه، جزییات الگوریتم ترکیبی ارایه شده، که HGA نامیده می‌شود، بیان می‌گردد.

٤-١-١ الگوریتم ژنتیک ترکیبی

الگوریتم ژنتیک ترکیبی یکی از الگوریتم‌های فراابتکاری است که در حل مسایل پیچیده بهینه‌سازی ترکیبی به کار گرفته می‌شود. از این الگوریتم در حل مسایلی که فضای جواب آن بسیار بزرگ است (مانند مساله مورد بررسی در این مقاله) استفاده می‌شود. تحقیقات گذشته نشان می‌دهد که این الگوریتم کارایی خوبی در حل مسایل زمان‌بندی و مسایل تعیین جدول زمانی[3] دارد [۲۶]. این الگوریتم در واقع نوعی الگوریتم تکاملی است که الگوریتم ژنتیک معمولی را با یک رویه جستجوی محلی ترکیب می‌کند. جزییات الگوریتم ژنتیک ترکیبی توسط مسکاتو [۲۷] و کرن [۲۸] شرح داده شده است.

- **شیوه نمایش جواب**

یکی از مهم‌ترین جنبه‌های الگوریتم‌های فراابتکاری شیوه نمایش جواب است. تعریف یک شیوه نمایش جواب مناسب تاثیر بسزایی بر عملکرد عملگرهای ترکیب و جهش و همچنین رویه جستجوی محلی خواهد داشت. به هر جواب در الگوریتم ژنتیک، کروموزوم[4] گفته می‌شود. مراحل مختلف الگوریتم نظیر عملگرهای ترکیب و جهش

[1] Hybrid genetic algorithm
[2] Greedy randomized adaptive search procedure
[3] Timetabling
[4] Chromosome

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

و دیگر عملگرها روی کروموزوم‌ها اجرا می‌شوند. در الگوریتم HGA ارایه شده، هر کروموزوم به صورت توالی از کارها تعریف شده‌است. بنابراین، هر جایگشتی از ۱ تا n (تعداد کارها)، یک کروموزوم یا جواب را نمایش می‌دهد. نحوه محاسبه شایستگی هر کروموزوم که برابر با مجموع زودهنگامی و دیرهنگامی تمام کارهاست در بخش‌های بعدی شرح داده خواهد شد.

➢ ساختار جمعیت و جمعیت اولیه

جمعیت مورد استفاده برای الگوریتم ژنتیک ترکیبی در این مقاله یک جمعیت ساختاریافته است. جمعیت مورد استفاده یک جمعیت با ساختار سلسله مراتبی مطابق با یک درخت کامل سه تایی است. بر خلاف روش‌های غیر ساختاریافته، که همه کروموزوم‌ها می‌توانند با هم ترکیب شوند، جمعیت به چندین بخش تقسیم می‌شود در نتیجه تعداد جفت ترکیب‌های ممکن محدود می‌شود.

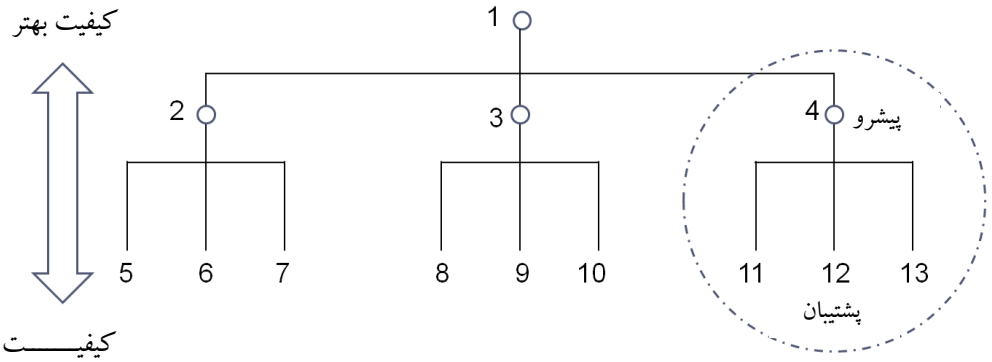
ساختار جمعیت همان‌گونه که در شکل ۱ نشان داده شده است، شامل چندین بخش است و هر بخش یک جواب پیشرو[1] و سه جواب پشتیبان[2] دارد. پیشرو هر بخش در طول مدت اجرا به صورت پویا به روز می‌شود تا همواره کیفیت بهتری نسبت به سه پشتیبان خود داشته باشد. از این رو، بخش‌های بالایی نسبت به بخش‌های پایین‌تر، متمایل به جواب‌های بهتری هستند. با تولید مداوم جواب‌ها (چه با ترکیب، جهش یا جستجوی محلی) و جایگزینی جواب‌های قدیمی، اصلاحات دوره‌ای لازم است تا این ساختار، شکل مناسب ترتیبی خود را حفظ کند.

تعداد جواب‌های جمعیت محدود به تعداد گره‌های یک درخت کامل سه تایی یعنی $(۳^x - ۱)/۲$ می‌شود که در آن x تعداد سطوح درخت است. بنابراین ۱۳ جواب مورد نیاز است تا یک درخت کامل سه تایی با ۳ سطح ساخته شود (مطابق شکل ۱) و ۴۰ جواب برای ساخت درختی با ۴ سطح لازم است.

برای تولید جمعیت اولیه به تعداد جمعیت مورد نیاز، دو کروموزوم بر اساس قوانین توالی بزرگ‌ترین زمان پردازش[3] و کوچک‌ترین زمان پردازش[4] تولید شده و مابقی کروموزوم‌ها به صورت کاملا تصادفی تولید می‌شوند و سپس با توجه به مقدار تابع هدفشان، در ساختار جمعیتی شرح داده شده قرار می‌گیرند. جواب‌های تولید شده در مرحله بعد با توجه به ترتیب صعودی مقدار تابع هدفشان به درخت ساختار جمعیت تخصیص می‌یابند. همواره بهترین کروموزوم در بالاترین سطح درخت و بدترین کروموزوم در پایین‌ترین سطح قرار می‌گیرند. تعداد کروموزوم‌های تشکیل‌دهنده جمعیت به صورت *pop_size* نشان داده می‌شود.

[1] Leader
[2] Supporter
[3] Longest processing time (LPT)
[4] Shortest processing time (SPT)

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]



تابع هدف	کروموزوم										فرد
۱۸	۶	۱	۹	۴	۱۰	۳	۸	۷	۲	۵	۱
۲۲	۹	۲	۱	۶	۱۰	۸	۳	۷	۴	۵	۲
۲۴	۱	۵	۷	۳	۴	۶	۸	۹	۱۰	۲	۳
۲۶	۹	۲	۵	۳	۱۰	۱	۷	۶	۴	۸	۴
۲۶	۳	۷	۱	۹	۸	۴	۱۰	۵	۶	۲	۵
۲۷	۶	۱۰	۵	۳	۴	۱	۹	۸	۷	۲	۶
۲۸	۶	۸	۹	۷	۱۰	۳	۵	۴	۲	۱	۷
۲۹	۲	۱۰	۷	۹	۱	۶	۵	۸	۳	۴	۸
۳۰	۱۰	۵	۶	۳	۲	۴	۹	۱	۷	۸	۹
۳۰	۴	۵	۶	۱	۲	۱۰	۹	۷	۸	۳	۱۰
۳۵	۱۰	۶	۴	۲	۷	۱	۸	۵	۳	۹	۱۱
۳۶	۶	۴	۵	۳	۷	۸	۲	۱۰	۹	۱	۱۲
۴۲	۵	۲	۴	۸	۱۰	۹	۱	۳	۷	۶	۱۳

شکل ۱. ساختار جمعیت به همراه ماتریس جمعیت مربوطه

➢ رویه انتخاب برای ترکیب

در هر تکرار الگوریتم، تعدادی کروموزوم از جمعیت برای ترکیب و تولید کروموزوم‌های جدید انتخاب می‌شوند. در حالت جمعیت ساختاریافته، ترکیب فقط مابین یک پیشرو و یکی از پشتیبان‌های آن در همان بخش قابل انجام است. رویه انتخاب، یک پیشرو (و متعاقبا بخش) را به شکل تصادفی یکنواخت انتخاب می‌کند. سپس، یکی از سه پشتیبان موجود در آن بخش (باز هم به شکل تصادفی یکنواخت) انتخاب می‌شود. محدودیتی در تعداد ترکیب‌هایی که یک کروموزوم می‌تواند درآن شرکت کند وجود ندارد.

➢ عملگر ترکیب

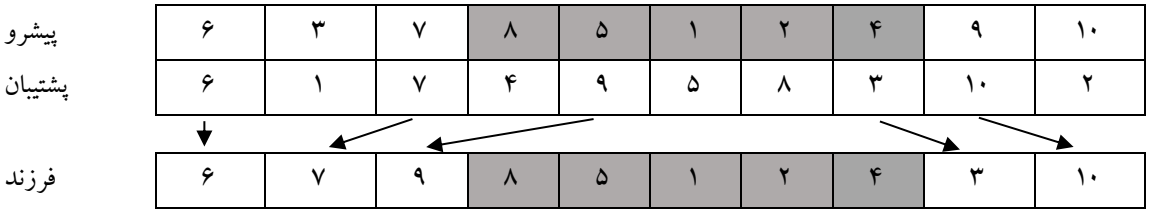
عملگر ترکیب مورد استفاده در این تحقیق، نوعی از عملگر معروف ترتیبی دونقطه‌ای[1] است. پس از آنکه دو والد (در حالت جمعیت ساختار یافته، یک پیشرو و یکی از پشتیبان‌های آن)، مطابق رویه انتخاب، انتخاب شدند ترکیب آن‌ها آغاز می‌شود. ابتدا یک قسمت از پیشرو به شکل تصادفی یکنواخت انتخاب می‌شود (در واقع ابتدا و انتهای این قسمت به شکل تصادفی مشخص می‌شود). این قسمت در فرزند و در همان مکان قرار می‌گیرد. ما بقی

[1] Two point order crossover

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

ژن‌های فرزند، طبق اطلاعات والد دوم تکمیل می‌شود. در واقع از چپ به راست رقم‌هایی که در قسمت کپی شده از والد اول ظاهر نشده‌اند، با توجه به ترتیبشان در والد دوم، در ژن‌های خالی فرزند قرار می‌گیرند. نحوه عمل عملگر ترکیب، در شکل ۲ نمایش داده شده است. در این شکل همچنین مشخص شده است که چه ژن‌هایی مستقیما از پیشرو به ارث برده شده و چگونه پشتیبان جاهای خالی باقیمانده را تکمیل کرده است.

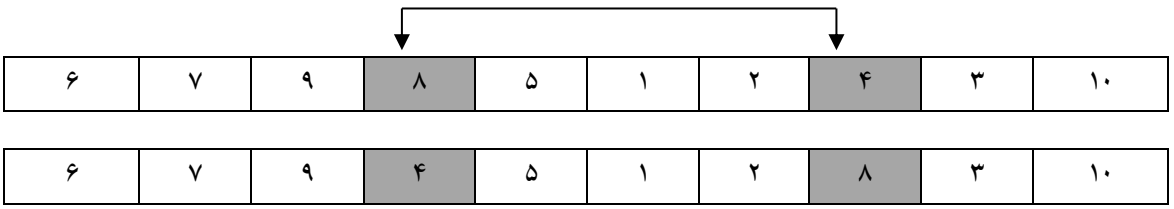
اطلاعاتی که از پیشرو به فرزند منتقل می‌شود کاملا دست نخورده باقی می‌ماند، به این معنی که ژن‌های انتخابی از آن دقیقا به همان موقعیت در فرزند منتقل می‌شوند. در مقابل پشتیبان فقط ترتیب نسبی را منتقل می‌کند. این روش تنوع بسیار کمی را ایجاد می‌کند، از این‌رو عملگر جهش بسیار با اهمیت خواهد بود تا از همگرایی پیش از موقع جلوگیری کند.



شکل ۲. مثالی از عملگر ترکیب ترتیبی دونقطه‌ای

➢ عملگر جهش

عمل جهش نقش مهمی را در تامین تنوع جمعیت بازی می‌کند مخصوصا اگر عملگرهای ترکیب بسیار سریع جلوی تنوع را بگیرند. در این تحقیق دو نوع رویه جهش اجرا شده است: یکی رویه جهش جزیی و دیگری رویه جهش شدید. جهش جزیی با توجه به نرخ جهش روی کروموزوم‌های فرزند اعمال می‌شود، اما رویه جهش شدید در پایان هر تولید نسل و فقط در صورت نیاز به کار گرفته می‌شود. رویه جهش جزیی استفاده شده، عملگر جهش تعویض جفتی است. در این روش، ابتدا دو ژن از کروموزوم به شکل تصادفی انتخاب می‌شوند و سپس مقادیر آن‌ها با یکدیگر تعویض می‌گردند. چون تغییرات روی کروموزوم در این روش بسیار کم است، احتمال این‌که مشخصه خاصی که در طول الگوریتم تکاملی به دست آمده از بین برود بسیار اندک است. شکل ۳ نحوه عملکرد این عملگر را به تصویر کشیده است.



شکل ۳. مثالی از عملگر جهش تعویض جفتی

جهش شدید، q جابجایی دوتایی را در هر کروموزوم اجرا می‌کند که q، تعداد ژن‌های موجود در کروموزوم است. در نتیجه وقتی این جهش شدید روی یک کروموزوم اعمال می‌شود، احتمالا تمام خصوصیاتش

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

را از دست می‌دهد و در انتهای فرآیند، یک جواب تصادفی را ایجاد می‌کند و همه مشخصه‌های تکاملی کروموزوم از دست خواهد رفت.

٤-١-٢ رویه جستجوی محلی

در الگوریتم ژنتیک ترکیبی، رویه جستجوی محلی روی فرزندهای جدید به منظور یافتن جواب جدید با میزان شایستگی بهتر به کار گرفته می‌شود. رویه‌های جستجوی محلی متکی به تعریف همسایگی هستند. اکثر الگوریتم‌های جستجوی محلی تنها از یک ساختار همسایگی استفاده می‌کنند. هر چند، جواب بهینه محلی یک ساختار همسایگی لزوما جواب بهینه محلی ساختار دیگر نخواهد بود. الگوریتم جستجوی همسایگی متغیر از ساختارهای همسایگی متفاوت و جابجایی بین آن‌ها برای جستجوی جواب بهتر استفاده می‌کند. در این پژوهش تغییر سیستماتیکی از همسایگی‌ها برای یافتن بهترین جواب بهینه محلی به کار گرفته می‌شود. به عبارت دیگر، از استراتژی تغییر پویای ساختار همسایگی برای جستجوی کارا و موثر استفاده می‌شود. در الگوریتم ژنتیک ترکیبی ارایه شده، از سه ساختار همسایگی که در شکل ۴ نمایش داده شده است، استفاده شده است: ساختار همسایگی تعویض همجوار[1] ($\mathcal{N}_1$)، تعویض جفتی[2] ($\mathcal{N}_2$) و درج[3] ($\mathcal{N}_3$).

در همسایگی اول، مقدار دو ژن مجاور جابه‌جا می‌شود. در ساختار تعویض جفتی، مقدار دو ژن که لزوما مجاور نیستند، تعویض می‌شود و در ساختار درج، یک ژن از محل مربوطه حذف شده و در مکان دیگری قرار می‌گیرد.

نخست، جستجو روی ساختار همسایگی اول برای یافتن جواب بهتر انجام می‌شود. اگر بهبودی در شایستگی جواب حاصل شد، جواب به‌روز می‌شود و جستجو روی جواب بهبود یافته ادامه می‌یابد. در غیر این‌صورت، جستجو در ساختار همسایگی بعدی ادامه می‌یابد. ساختارهای همسایگی‌ها یک‌به‌یک برای یافتن جواب با شایستگی بهتر مورد بررسی قرار می‌گیرند. جابجایی بین ساختارهای همسایگی از افتادن در نقطه بهینه محلی، جلوگیری می‌کند. وقتی هیچ جواب بهتری در یک ساختار همسایگی پیدا نشود، جواب بهینه محلی حاصل شده است. با تغییر در ساختار همسایگی و جستجو در ساختار همسایگی جدید، ممکن است جواب بهتری پیدا شود. در نهایت، جستجو بعد از تعداد ثابتی تکرار متوقف می‌شود. پیاده‌سازی الگوریتم جستجو به‌صورت زیر شرح داده شده است.

گام ۱: قرار دهید: جواب اولیه: x، تعداد تکرار $t=0$ و ساختار همسایگی $l=1$

گام ۲: از طریق ساختار $\mathcal{N}_l$، جواب همسایگی x_1 مربوط به جواب x را تولید کنید.

گام ۳: اگر میزان شایستگی x_1 از میزان شایستگی x بهتر است، x_1 را جایگزین x کنید، در غیر این‌صورت $l=l+1$.

[1] Adjacent interchange
[2] Swap
[3] Insertion

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

گام ۴: $t = t+1$، اگر حداکثر تعداد تکرار حاصل شده است، توقف کنید. در غیر این‌صورت به مرحله ۵ بروید.

گام ۵: اگر $l = 4$، قرار دهید $l = 1$ و به مرحله ۲ بروید، درغیراین‌صورت به مرحله ۲ بروید.



شکل ٤. ساختار همسایگی‌ها در الگوریتم جستجوی محلی

➢ به‌روزرسانی جمعیت

در این پژوهش از یک سیاست سختگیرانه که فقط افراد جدیدی را می‌پذیرد که میانگین شایستگی جمعیت را بهبود بخشند استفاده شده است. در عمل این بدان معناست که فرزند جدیدی با تابع هدف بیشتر از هر دو والدش یا مساوی با یکی از والدینش، خود به خود رد می‌شود. پذیرش تنها افرادی که متوسط کیفیت را افزایش می‌دهند، باعث از دست رفتن تنوع می‌شود اما باعث نرخ ثابت سریع‌تری در بهبود بهترین جواب تاکنون می‌شود. برای الگوریتم ژنتیک ترکیبی این سیاست منجر به جواب‌های عالی می‌شود و نرخ بهبود، بسیار بهتر از سیاستی خواهد بود که همیشه افراد جدید پذیرفته شوند. به علاوه برای مقابله با کاهش تنوع جمعیت از رویه جهش شدید شرح داده شده در بخش قبل استفاده می‌شود. هر موقع هیچ جواب جدیدی نتواند در طول یک تولید نسل کامل، وارد جمعیت شود از رویه جهش شدید استفاده می‌شود. وقتی هیچ جوابی نتواند وارد جمعیت شود نشان می‌دهد که جمعیت به سوی جواب‌های خیلی نزدیک به هم در فضای جواب همگرا شده است. در این شرایط ایجاد یک بهبود جدید به سختی به دست می‌آید و در نتیجه یک جهش شدید اجرا می‌شود تا فرآیند تکاملی از نو آغاز شود. فرآیند جهش شدید روی همه جواب‌ها به جز بهترین آن‌ها که در حالت جمعیت ساختار یافته در گره ریشه واقع است اجرا می‌شود.

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

در پایان هر تولید نسل، جمعیت دوباره ساختارمند می‌شود تا رابطه سلسله مراتبی بین افراد حفظ شود. از آن جا که جمعیت به شکل سلسله مراتبی ساختار یافته و مرتب شده است تابع هدف هر فرد باید بهتر از تابع هدف همه افرادی باشد که در زیردرخت‌های پایین آن قرار دارند. زیر بخش‌های بالاتر افراد بهتری نسبت به بخش‌های پایین‌تر خواهند داشت و بهترین جواب، پیشرو بالاترین بخش خواهد بود. اصلاح، با مقایسه هر پیشرو با پشتیبان‌هایش انجام می‌شود. یعنی اگر پشتیبان بهتر از آن باشد جایشان عوض می‌شود. یک الگوریتم ساده مرتب‌سازی درخت به راحتی می‌تواند این کار را انجام دهد.

➢ محاسبه شایستگی کروموزوم‌ها

میزان شایستگی هر کروموزوم برابر با مجموع زودهنگامی و دیرهنگامی تمام کارها تعریف می‌شود. بنابراین برای محاسبه میزان شایستگی یک کروموزوم، ابتدا کارها بر اساس توالی تعریف شده در کروموزوم و با استفاده از رویه *First-Fit* به انباشته‌ها تخصیص می‌یابند و سپس انباشته‌ها با هدف کمینه کردن مجموع زودهنگامی و دیرهنگامی روی ماشین پردازش می‌شوند. در رویه *First-Fit* اولین کار در ابتدای لیست انتخاب شده و در اولین انباشته با ظرفیت خالی کافی قرار می‌گیرد. اگر کار مربوطه در هیچ انباشته‌ای جای نگرفت، به یک انباشته جدید اختصاص داده می‌شود. این رویه تا تخصیص تمام کارها به انباشته‌ها تکرار می‌شود. به منظور محاسبه زمان‌بندی بهینه با کمترین مقدار زودهنگامی و دیرهنگامی برای انباشته‌های تشکیل شده، از الگوریتم برنامه‌ریزی پویا ارایه شده توسط رفیعی پارسا و همکاران [۲۴] استفاده شده است. الگوریتم برنامه‌ریزی پویا ارایه شده، بر اساس یک رابطه بازگشتی حداقل هزینه زودهنگامی و دیرهنگامی را برای انباشته‌ها تعیین می‌کند. از آنجا که موعد تحویل دور فرض شده است، زمان‌بندی بهینه V-شکل است و یکی از انباشته‌ها دقیقا در موعد تحویل تکمیل می‌شود. مجموعه انباشته‌های تشکیل شده به صورت $\varphi=\{B_1,B_2,...,B_m\}$ را در نظر بگیرید، که در آن m تعداد انباشته‌ها و B_b به ازای $b=1,2,...,m$ انباشته‌ای شامل زیر مجموعه‌ای از کارها است. P_b و n_b را به ترتیب زمان پردازش انباشته b و تعداد کارهای قرار گرفته در انباشته b در نظر بگیرید. فرض کنید انباشته‌های مجموعه φ به ترتیب صعودی زمان پردازش وزنی انباشته‌ها[1] (BWSPT) مرتب شده‌اند، یعنی $\frac{P_1}{n_1}\le\frac{P_2}{n_2}\le\cdots\le\frac{P_m}{n_m}$.

هدف الگوریتم برنامه‌ریزی پویای ارایه شده، یافتن توالی و زمان‌بندی انباشته‌های مذکور است به نحوی که مجموع هزینه‌های زودهنگامی و دیرهنگامی کارها حداقل شود.

این الگوریتم، به کمک یک رابطه بازگشتی حداقل هزینه زودهنگامی و دیرهنگامی را برای انباشته‌های مجموعه φ تعیین می‌کند. با فرض اینکه انباشته‌های $1,2,...,b-1$ زمان‌بندی شده‌اند، $ET_b(s)$ را حداقل هزینه زمان‌بندی انباشته‌های $b,b+1,...,m$ قراردهید. مجموع زمان پردازش انباشته‌های زمان‌بندی شده دارای زودهنگامی را نیز s در نظر بگیرید. از آنجا که با دانستن s، مجموع زمان پردازش انباشته‌های زمان‌بندی شده دارای دیرهنگامی قابل تعیین است، متغیر حالت تنها با s تعریف می‌شود. مجموع زمان پردازش تمام انباشته‌های از قبل زمان‌بندی شده برابر است با $\sum_{a=1}^{b-1}P_a$، بنابراین مجموع زمان پردازش انباشته‌های دارای دیرهنگامی برابر

[1] Batch weighted shortest processing time

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

با $\sum_{a=1}^{b-1} P_a - s$ خواهد بود. رابطه بازگشتی در الگوریتم برنامه‌ریزی پویای ارایه شده به صورت زیر تعریف می‌شود:

$$ET_b(s) = \min\left\{n_b s + ET_{b+1}(s+P_b),\ n_b\left(\sum_{a=1}^{b} P_a - s\right) + ET_{b+1}(s)\right\} \quad (1)$$

شرایط حدی نیز به صورت زیر در نظر گرفته شده است:

$$ET_{m+1}(s) = 0, \qquad s = 0,1,\ldots,\sum_{a=1}^{m} P_a \quad (2)$$

زمان‌بندی بهینه از طریق یافتن $ET_1(0)$ به دست می‌آید و زمان‌بندی متناظر با آن با برگشت به عقب تعیین خواهد شد. لازم به ذکر است در رابطه بازگشتی (۱)، عبارت اول و دوم به ترتیب هزینه قراردادن انباشته bام قبل از موعد تحویل (در ابتدای توالی) و بعد از موعد تحویل (در انتهای توالی) را نشان می‌دهد. اگر انباشته b را در ابتدای توالی قرار دهیم، این انباشته به اندازه s دارای زودهنگامی خواهد بود و متغیر حالت از s به $s+P_b$ تغییر خواهد کرد (عبارت اول). در غیر این‌صورت اگر انباشته b را در انتهای توالی قرار دهیم، این انباشته به اندازه $\sum_{a=1}^{b} P_a - s$ دارای دیرهنگامی خواهد بود و متغیر حالت تغییر نخواهد کرد (عبارت دوم).

٤-۲ الگوریتم جستجوی تطبیقی تصادفی حریصانه

در این بخش به ارایه الگوریتم جستجوی تطبیقی تصادفی حریصانه (GRASP) برای مساله *SUMET* پرداخته می‌شود. الگوریتم GRASP یک روش فراابتکاری است که توسط فئو و رسنده [۲۹] معرفی شده است. این الگوریتم ترکیبی از سه رویکرد کلاسیک کاربردی در بهینه‌سازی ترکیبی شامل الگوریتم ابتکاری حریصانه[1]، تصادفی کردن[2]، و رویه جستجوی محلی است. الگوریتم GRASP از یک جواب آغازین که توسط الگوریتم حریصانه تصادفی ایجاد شده است، شروع کرده و از طریق جستجوی محلی سعی بر بهبود آن دارد. جستجوی محلی تا رسیدن به شرط توقف ادامه می یابد. این رویه به منظور یافتن بهترین جواب چند بار تکرار می‌شود. بنابراین ساختار این الگوریتم، رویه‌ای تکراری شامل دو مرحله است: ساخت جواب و بهبود جواب. مکانیزم ساخت جواب با دو جزء اصلی مشخص می گردد: الگوریتم حریصانه سازنده و نحوه تصادفی کردن. فرض کنید جواب شامل زیرمجموعه‌ای از مجموعه عناصر (اجزای جواب) است. جواب با اضافه کردن مرحله به مرحله یک عنصر جدید، ساخته می‌شود. انتخاب عنصر بعدی با برداشتن تصادفی عنصر به صورت یکنواخت از لیست کاندیداها انجام می‌شود.

مرحله دوم الگوریتم، فرآیند جستجوی محلی است که هدف آن بهبود جواب به دست آمده در مرحله اول است. انتخاب توابع ابتکاری موثر و طول لیست کاندید مناسب بر عملکرد الگوریتم GRASP تاثیر بسزایی دارد.

[1] Greedy heuristic
[2] Randomization

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

این الگوریتم از حافظه جستجو استفاده نمی‌کند ولی به دلیل سادگی، عمدتا بسیار سریع است و در زمان کوتاه جواب‌های نسبتا خوبی ارایه می‌کند. همچنین قابلیت ادغام در سایر تکنیک‌های جستجو، را نیز دارد.

٤-٢-١ الگوریتم حریصانه تصادفی

هر تکرار الگوریتم GRASP، از یک جواب شدنی که توسط الگوریتم حریصانه تصادفی تولید می‌شود، آغاز می‌شود. این جواب، به‌عنوان جواب اولیه رویه جستجوی محلی در نظر گرفته می‌شود و جستجو تا رسیدن به شرط توقف ادامه می‌یابد. الگوریتم حریصانه تصادفی به دلیل ماهیت تصادفی آن از تولید یک جواب مشخص در هر تکرار جلوگیری می‌کند.

در الگوریتم GRASP ارایه شده، الگوریتم حریصانه تصادفی بر اساس روش ابتکاری *ETFF-LPT* که توسط رفیعی پارسا و همکاران [۲۴] ارایه شده، یک جواب اولیه تولید می‌کند. در الگوریتم *ETFF-LPT*، نخست کارها به ترتیب نزولی زمان‌های پردازش‌شان مرتب می‌شوند. اولین کار تخصیص نیافته در ابتدای لیست در اولین انباشته با ظرفیت خالی کافی قرار می‌گیرد. اگر کار مربوطه در هیچ انباشته‌ای جای نگرفت، انباشته جدیدی ایجاد شده و کار به انباشته جدید تخصیص می‌یابد. این رویه تا تخصیص تمام کارها به انباشته‌ها تکرار می‌شود.

ماهیت تصادفی بودن با تعریف لیست کاندید محدود شده[1] (RCL) در الگوریتم ابتکاری *ETFF-LPT* در نظر گرفته می‌شود. بنابراین در الگوریتم ابتکاری حریصانه تصادفی، به منظور تخصیص کارها به انباشته‌ها، به جای انتخاب کار تخصیص نیافته با بیشترین زمان پردازش، کار به صورت تصادفی از اولین k کار تخصیص نیافته با بیشترین زمان پردازش انتخاب می‌شود. اولین k کار تخصیص نیافته با بیشترین زمان پردازش، لیست کاندید محدود شده را در الگوریتم GRASP تشکیل می‌دهند که در آن k اندازه RCL است. بعد از تشکیل انباشته‌ها، با استفاده از الگوریتم برنامه‌ریزی پویا که در بخش قبل شرح داده شد، زمان‌بندی بهینه با حداقل هزینه زودهنگامی و دیرهنگامی انباشته‌های تشکیل شده به دست می‌آید.

٤-٢-٢ رویه جستجوی محلی

در رویه جستجوی محلی از دو ساختار همسایگی استفاده شده است. ساختار همسایگی تعویض دو کار[2] و درج کار[3]. در ساختار تعویض، دو کار زمان‌بندی شده در انباشته‌های متفاوت به صورت تصادفی انتخاب شده و جابه‌جا می‌شوند. در ساختار همسایگی درج، یک کار تصادفی از یک انباشته انتخاب شده و به صورت تصادفی در انباشته دیگر یا یک انباشته جدید قرار می‌گیرد. هر دو ساختار همسایگی با درنظر گرفتن محدودیت ظرفیت ماشین صورت می‌گیرد. همسایگی‌ها به صورت تصادفی انتخاب می‌شوند و احتمال انتخاب همسایگی اول برابر θ است. بعد از یافتن یک جواب طبق ساختار همسایگی، حداقل هزینه زودهنگامی و دیرهنگامی انباشته‌های تشکیل شده به منظور به‌روزرسانی جواب تعیین می‌شود.

[1] Restricted candidate list
[2] Job interchange
[3] Job insertion

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

۵ نتایج محاسباتی

هدف این بخش تنظیم پارامترها و سپس ارزیابی عملکرد الگوریتم‌های ارایه شده برای مساله *SUMET* است. تمامی الگوریتم‌های ارایه شده، به کمک نرم‌افزار MATLAB 10.0 برنامه‌نویسی و اجرا شده است. آزمایشات محاسباتی روی یک کامپیوتر شخصی با پردازنده 3.1 GHz و حافظه 4 GB اجرا شده است. برای انجام آزمایشات محاسباتی، نمونه مسایلی مشابه جولای و دوپونت [۴] در نظر گرفته شده است. پارامترهای متعددی بر نتایج محاسباتی تاثیرگذار هستند، نظیر تعداد کارها، اندازه کارها، و زمان پردازش کارها. مشخصات نمونه مسایل استفاده شده به شرح زیر است:

- تعداد کارها (n) در محدوده ۱ تا ۲۰۰ و در ابعاد ۱۰، ۱۲، ۱۵، ۲۰، ۴۰، ۶۰، ۸۰، ۱۰۰، و ۲۰۰ در نظر گرفته شده است.
- اندازه کارها (s_j) اعداد صحیح تصادفی از توزیع‌های یکنواخت U[۱,۱۰]، U[۳,۸]، U[۴,۱۰] و U[۱,۵] است.
- زمان پردازش کارها روی ماشین‌ها، اعداد صحیح از توزیع یکنواخت بین ۱ تا ۱۰۰ است.
- ظرفیت ماشین در تمام آزمایشات محاسباتی برابر ۱۰ در نظر گرفته شده است.

۵-۱ تنظیم پارامترهای الگوریتم

در این بخش، نحوه تنظیم پارامترهای الگوریتم ارایه شده در این مقاله شرح داده می‌شود.

۵-۱-۱ تنظیم پارامترهای الگوریتم HGA

عملکرد الگوریتم HGA به مقادیر پارامترهای موثر بر فرآیند جستجو حساس است. به منظور دستیابی به نتایج با کیفیت بالا، انتخاب مقادیر مناسب برای پارامترهای الگوریتم ضروری است. آزمایشات اولیه نشان داد که پارامترهای موثر بر عملکرد الگوریتم عبارتند از: اندازه جمعیت (*pop_size*)، نرخ ترکیب ($\mathcal{C}$)، نرخ جهش ($\mathcal{M}$)، و حداکثر تعداد تکرار الگوریتم جستجوی محلی ($t_{\max}$). برای یافتن مقادیر مناسب برای این پارامترها، طراحی آزمایش‌ها بر اساس روش تاگوچی[1] [۳۰] انجام شده است. روش تاگوچی با تعداد آزمایشات کمتر سعی بر افزایش پایداری[2] سیستم با حداقل کردن تغییرات در نتایج دارد. برای یافتن مقادیر مناسب پارامترها، تجزیه و تحلیل مقدماتی گسترده‌ای با تنظیمات مختلف اجرا شده است. از نرم‌افزار آماری Minitab 17 برای برنامه‌نویسی و اجرای آزمایشات آماری استفاده شده و بر اساس نتایج به دست آمده، ۴ عامل و ۳ سطح برای هر یک از مقادیر پارامترها درنظر گرفته شده است. جزییات عامل‌ها و سطوح مختلف آن‌ها در جدول ۱ نشان داده می‌شود.

[1] Taguchi method
[2] Robustness

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

جدول ۱. سطوح عامل‌ها در الگوریتم HGA

A: اندازه جمعیت (pop_size)	B: نرخ ترکیب ($\mathcal{C}$)	C: نرخ جهش ($\mathcal{M}$)	D: حداکثرتعداد تکرار جستجوی محلی ($t_{\max}$)
A(۱): ۱۳	B(۱): ۰/۲	C(۱): ۰/۰۱	D(۱): ۱۰
A(۲): ۴۰	B(۲): ۰/۴	C(۲): ۰/۱	D(۲): ۲۰
A(۳): ۱۲۱	B(۳): ۰/۸	C(۳): ۰/۵	D(۳): ۳۰

آرایه متعامد $L_۹(۳^۴)$ بهترین طراحی برای سطوح عامل‌ها است. تمامی ۹ سناریو تعریف شده برای ۲۰ نمونه مساله اجرا شده است. بنابراین، در مجموع ۱۸۰ اجرا برای این طراحی مورد نیاز است. تابع هدف الگوریتم HGA به عنوان متغیر پاسخ طراحی در نظر گرفته شده است.

پایداری الگوریتم با معیار اثر-به-اختلال[1] (S/N) اندازه‌گیری می‌شود. نسبت S/N، به صورت $-۱۰\log(ET)^۲$ محاسبه می‌شود. برای مسایل حداقل‌سازی، مقدار بیشتر نسبت S/N تغییرات کمتر نتایج خروجی را به همراه خواهد داشت. از طرفی، با توجه به این‌که مسایل نمونه از اندازه‌های مختلف هستند، به طبع آن مقیاس مقادیر توابع هدف متفاوت خواهد بود، لذا می‌بایست اثر اندازه مسایل حذف شود. به این منظور، به جای مقدار تابع هدف هر مساله نمونه از درصد انحراف نسبی[2] (RPD) استفاده شده است. RPD به صورت $(ET - ET_{best}) / (ET_{worst} - ET_{best})$ محاسبه می‌شود که در آن ET مقدار تابع هدف مساله نمونه به دست آمده از الگوریتم HGA، و ET_{best} و ET_{worst} به ترتیب بهترین و بدترین مقدار تابع هدف به دست آمده مساله نمونه در سناریوهای مختلف است. بنابراین، نسبت S/N برای سناریو r به کمک رابطه زیر محاسبه می‌شود.

$$(S/N)_r = -۱۰\log(\frac{۱}{۲۰}\sum_{i=۱}^{۲۰} RPD_{ir}^۲) \qquad r = ۱,۲,...,۹ \qquad (۳)$$

برای هر عامل، بهترین سطح، سطحی است که بیشترین نسبت S/N (حداکثر پایدرای) و کم‌ترین مقدار RPD (حداقل هزینه) را دارد. مقادیر نهایی پارامترهای الگوریتم در جدول ۲ بیان شده است.

جدول ۲. سطوح انتخاب شده عامل‌ها در الگوریتم HGA

A: اندازه جمعیت (pop_size)	B: نرخ ترکیب ($\mathcal{C}$)	C: نرخ جهش ($\mathcal{M}$)	D: حداکثرتعداد تکرار الگوریتم جستجوی محلی ($t_{\max}$)
A(۲): ۴۰	B(۳): ۰/۸	C(۲): ۰/۱	D(۳): ۳۰

۵-۱-۲ تنظیم پارامترهای الگوریتم GRASP

پارامترهای تاثیرگذار بر الگوریتم GRASP ارایه شده، عبارتند از: اندازه RCL (k)، احتمال استفاده از ساختارهای همسایگی و حداکثر تکرار برای توقف الگوریتم. آزمایشات اولیه برای انتخاب مقادیر مناسب برای این پارامترها انجام شده است. برای یافتن مقادیر مناسب برای این پارامترها، مشابه الگوریتم HGA، از روش

[1] Signal-to-Noise
[2] Relative percentage deviation

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

تاگوچی [۳۰] استفاده شده است. بر اساس نتایج به دست آمده، ۳ عامل و ۲ سطح برای هر یک از مقادیر پارامترها درنظر گرفته شده است. جزییات عامل‌ها و سطوح مختلف آن‌ها در جدول ۳ نشان داده شده است.

جدول ۳. سطوح عامل‌ها در الگوریتم GRASP

A: اندازه لیست (k)	B: احتمال انتخاب همسایگی (θ)	C: حداکثر تعداد تکرار الگوریتم ($t_{\max}$)
A(۱): ۲	B(۱): ۰/۵	C(۱): ۱۰۰
A(۲): ۴	B(۲): ۰/۹	C(۲): ۵۰۰

آرایه متعامد $L_8(2^3)$ بهترین طراحی برای سطوح عامل‌ها است. تمامی ۸ سناریو تعریف شده برای ۲۰ نمونه مساله اجرا شده است. بنابراین، در مجموع ۱۶۰ اجرا برای این طراحی مورد نیاز است. مقادیر نهایی پارامترهای الگوریتم در جدول ۴ بیان شده است. طبق نتایج حاصل از طراحی تاگوچی، بهترین سطوح A(۲)، B(۲) و C(۲) است.

جدول ٤. سطوح انتخابی پارامترهای الگوریتم GRASP

A: اندازه لیست (k)	B: احتمال انتخاب همسایگی (θ)	C: حداکثر تعداد تکرار الگوریتم ($t_{\max}$)
A(۲): ۴	B(۲): ۰/۹	C(۲): ۵۰۰

۵-۲ ارزیابی کارایی الگوریتم‌های ارایه‌شده

هدف این بخش اعتبارسنجی الگوریتم‌های HGA و GRASP ارایه‌شده در این مقاله است. ابتدا به ارزیابی الگوریتم‌های ارایه‌شده برای نمونه مسایل کوچک پرداخته شده است. بنابراین، میانگین (Avg) و حداکثر (Max) درصد اختلاف جواب الگوریتم‌های فراابتکاری با جواب بهینه به‌دست آمده از الگوریتم شاخ و کران ارایه شده توسط رفیعی پارسا و همکاران [۲۴] به تفکیک اندازه و تعداد کارها در جدول ۵ نشان داده شده است.

بر اساس نتایج جدول ۵، الگوریتم‌های HGA و GRASP قادر به یافتن جواب بهینه برای تمام مسایل با ۱۰ کار هستند. اختلاف جواب الگوریتم HGA از جواب بهینه در نمونه مسایل کوچک به طور متوسط ۰/۶ درصد و این مقدار برای الگوریتم GRASP در حدود ۱/۵ درصد گزارش شده‌است. نتایج حاکی از معتبر و همچنین کارا بودن این الگوریتم‌ها است.

از آنجا که مساله *SUMET*، یک مساله NP-hard است، یافتن جواب بهینه مساله در اندازه‌های بزرگ در زمان معقول امکان‌پذیر نیست. برای سنجش میزان کارایی الگوریتم‌های فراابتکاری، از حد پایین ارایه شده توسط رفیعی پارسا و همکاران [۲۴] استفاده شده است. بنابراین، در این بخش میزان کارایی الگوریتم‌های فراابتکاری بر اساس فاصله آن‌ها تا مقدار حد پایین ارزیابی می‌شود.

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

جدول ۵. مقایسه نتایج الگوریتم‌های فراابتکاری و جواب بهینه برای نمونه مسایل کوچک

s_j	n	HGA		GRASP	
		Avg	Max	Avg	Max
[۱, ۱۰]	۱۰	۰/۰۰	۰/۰۰	۰/۰۰	۰/۰۰
	۱۲	۰/۰۰	۰/۰۰	۱/۶۱	۷/۶۱
	۱۵	۱/۴۱	۲/۱۳	۲/۷۳	۴/۱۲
	۲۰	۲/۰۶	۳/۳۷	۳/۰۴	۵/۷۴
	میانگین	**۰/۸۷**	**۱/۳۸**	**۱/۸۵**	**٤/۳۷**
[۳, ۸]	۱۰	۰/۰۰	۰/۰۰	۰/۰۰	۰/۰۰
	۱۲	۰/۰۰	۰/۰۰	۱/۱۷	۵/۸۴
	۱۵	۰/۰۰	۰/۰۰	۲/۰۶	۶/۲۳
	۲۰	۲/۶۴	۳/۷۷	۴/۵۹	۷/۲۴
	میانگین	**۰/٦٦**	**۰/۹٤**	**۱/۹٦**	**٤/۸۳**
[۴, ۱۰]	۱۰	۰/۰۰	۰/۰۰	۰/۰۰	۰/۰۰
	۱۲	۰/۰۰	۰/۰۰	۰/۰۰	۰/۰۰
	۱۵	۰/۰۰	۰/۰۰	۰/۰۰	۰/۰۰
	۲۰	۰/۰۰	۰/۰۰	۰/۸۴	۱/۷۳
	میانگین	**۰/۰۰**	**۰/۰۰**	**۰/۲۱**	**۰/٤۳**
[۱, ۵]	۱۰	۰/۰۰	۰/۰۰	۰/۰۰	۰/۰۰
	۱۲	۰/۰۰	۰/۰۰	۱/۴۹	۶/۱۸
	۱۵	۱/۵۷	۳/۶۳	۲/۴۷	۴/۷۶
	۲۰	۲/۸۴	۳/۳۴	۳/۱۹	۴/۹۲
	میانگین	**۱/۱۰**	**۱/۷٤**	**۱/۷۹**	**۳/۹۷**

تمامی نمونه مسایل به کمک الگوریتم‌های HGA و GRASP برای تولید جواب نزدیک بهینه حل شده‌اند. سپس میزان کارایی الگوریتم‌ها بر اساس درصد اختلاف جواب الگوریتم از حد پایین طبق رابطه

$$\frac{\text{حد پایین} - \text{جواب الگوریتم}}{\text{حد پایین}} \times ۱۰۰$$

سنجیده می‌شود. متوسط درصد اختلاف الگوریتم‌ها از حد پایین در جدول ۶ گزارش شده است. شرط توقف برای هر دو الگوریتم HGA و GRASP مدت زمان ۶۰ ثانیه در نظر گرفته شده‌است. متوسط خطای الگوریتم HGA برای همه نمونه مسایل برابر ۶/۸۲٪ و این مقدار برای الگوریتم GRASP برابر ۱۱/۶۴٪ است. توزیع درصد خطای الگوریتم‌های HGA و GRASP به ترتیب در جداول ۷ و ۸ ارایه شده است.

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

جدول ٦. متوسط خطای الگوریتم‌های فراابتکاری نسبت به حد پایین

n	s_j	HGA	GRASP	s_j	HGA	GRASP
۲۰		۴/۳۲	۷/۶۷		۳/۴۲	۵/۵۷
۴۰		۵/۱۴	۸/۷۶		۳/۹۲	۶/۰۷
۶۰	[۱, ۱۰]	۶/۷۴	۹/۱۸	[۴, ۱۰]	۴/۲۱	۷/۶۹
۸۰		۷/۸۹	۱۱/۷۷		۴/۷۹	۸/۹۸
۱۰۰		۹/۳۱	۱۵/۰۶		۵/۵۷	۱۲/۸۷
۲۰۰		۱۱/۲۹	۲۰/۵۹		۶/۸۳	۱۹/۶۹
۲۰		۴/۵۴	۶/۰۹		۴/۶۶	۶/۰۹
۴۰		۴/۸۶	۷/۴۵		۵/۱۹	۷/۴۵
۶۰	[۳, ۸]	۵/۱۲	۸/۱۳	[۱, ۵]	۷/۱۸	۸/۱۳
۸۰		۵/۴۸	۹/۸۸		۹/۷۴	۱۱/۸۸
۱۰۰		۶/۳۷	۱۴/۰۷		۱۴/۰۳	۱۹/۰۷
۲۰۰		۶/۸۴	۲۱/۶		۱۶/۲۶	۲۵/۶

جدول ۷. توزیع درصد خطای الگوریتم HGA

خطا (٪)	[۰-۱]	(۱-۵]	(۵-۱۰]	(۱۰-۱۵]	(۱۵-۱۰۰]
فراوانی نسبی (٪)	۷/۷۶	۴۲/۵۳	۳۰/۴۷	۱۳/۴۱	۵/۸۳
فراوانی نسبی تجمعی (٪)	۷/۷۶	۵۰/۲۹	۸۰/۷۶	۹۴/۱۷	۱۰۰/۰۰

جدول ۸. توزیع درصد خطای الگوریتم GRASP

خطا (٪)	[۰-۱]	(۱-۵]	(۵-۱۰]	(۱۰-۱۵]	(۱۵-۱۰۰]
فراوانی نسبی (٪)	۵/۳۴	۲۴/۱۷	۳۰/۴۹	۲۸/۳۹	۱۱/۶۱
فراوانی نسبی تجمعی (٪)	۵/۳۴	۲۹/۵۱	۶۰/۰۰	۸۸/۳۹	۱۰۰/۰۰

برای مقایسه کارایی الگوریتم‌ها از آزمون تی زوجی با سطح اطمینان ۵٪ استفاده شده است. نتایج مقایسه در جدول ۹ ارایه شده است. همان‌طور که مشاهده می‌شود، مقدار P-value مربوط به آزمون مقایسه کارایی الگوریتم‌ها برابر ۰/۰۱۵ است. بنابراین می‌توان نتیجه گرفت تفاوت معناداری بین کارایی دو الگوریتم وجود دارد. با توجه به اینکه متوسط درصد اختلاف نسبت به حد پایین در الگوریتم HGA، کم‌تر از مقدار این متوسط در الگوریتم GRASP است، می‌توان نتیجه گرفت الگوریتم HGA جواب‌های مناسب‌تری برای مساله تولید می‌کند.

جدول ۹. آزمون تی زوجی برای مقایسه الگوریتم‌های HGA و GRASP

	HGA	GRASP
Mean	۶/۸۲	۱۱/۶۴
Observations	۴۸۰	۴۸۰
Hypothesized Mean Difference	۰	
Df	۴۷۹	
t Stat	۲/۱۶۷	
P(T<=t) one-tail	۰/۰۱۵	
t Critical one-tail	۱/۶۴۸	

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

در آزمایشی دیگر، برای ارزیابی کارایی الگوریتم‌های فراابتکاری، عملکرد الگوریتم HGA، به عنوان بهترین الگوریتم فراابتکاری ارایه شده، با الگوریتم توسعه داده شده توسط لی و همکاران [۲۳] به نام الگوریتم GAMARB مقایسه می‌شود. تمام مسایل نمونه با هر دو الگوریتم حل شده‌اند و شرط توقف برای دو الگوریتم ۱۰۰۰ تکرار درنظر گرفته شده است. نتایج مقایسه عملکرد این دو الگوریتم در جدول ۱۰ ارایه شده است.

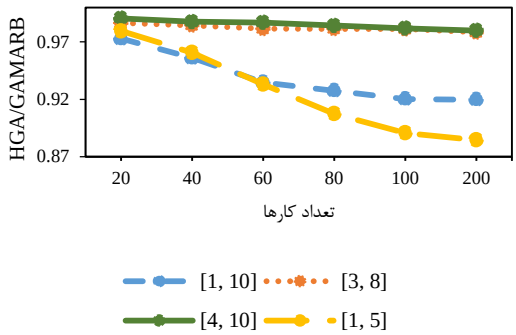
جدول ۱۰. نتایج مقایسه الگوریتم‌های HGA و GAMARB

s_j	n	HGA	GAMARB	HGA/ GAMARB
[۱, ۱۰]	۲۰	۶۵۶/۶	۶۷۵/۰	۰/۹۷۲۸
	۴۰	۲۷۱۶/۴	۲۸۴۱/۲	۰/۹۵۶۱
	۶۰	۵۸۱۳/۰	۶۲۱۹/۳	۰/۹۳۴۷
	۸۰	۱۰۰۷۲/۴	۱۰۸۵۹/۸	۰/۹۲۷۵
	۱۰۰	۱۵۲۰۲/۲	۱۶۵۲۰/۴	۰/۹۲۰۲
	۲۰۰	۶۱۸۷۳/۳	۶۷۲۷۵/۵	۰/۹۱۹۷
			میانگین	**۰/۹۳۸۵**
[۳, ۸]	۲۰	۲۹۶/۲	۳۰۰/۳	۰/۹۸۶۴
	۴۰	۱۲۰۱/۵	۱۲۲۰/۶	۰/۹۸۴۴
	۶۰	۲۵۲۵/۶	۲۵۷۳/۲	۰/۹۸۱۵
	۸۰	۴۲۹۴/۶	۴۳۷۶/۴	۰/۹۸۱۳
	۱۰۰	۷۰۵۸/۳	۷۱۹۳/۴	۰/۹۸۱۲
	۲۰۰	۲۷۱۵۹/۷	۲۷۷۶۲/۱	۰/۹۷۸۳
			میانگین	**۰/۹۸۲۲**
[۴, ۱۰]	۲۰	۱۳۰۳/۶	۱۳۱۶/۱	۰/۹۹۰۵
	۴۰	۵۴۰۹/۳	۵۴۷۷/۰	۰/۹۸۷۶
	۶۰	۱۱۸۶۵/۹	۱۲۰۲۳/۴	۰/۹۸۶۹
	۸۰	۱۸۸۱۲/۰	۱۹۱۱۴/۳	۰/۹۸۴۲
	۱۰۰	۳۱۹۷۲/۳	۳۲۵۶۵/۰	۰/۹۸۱۸
	۲۰۰	۱۲۷۳۵۵/۶	۱۲۹۹۶۸/۰	۰/۹۷۹۹
			میانگین	**۰/۹۸۵۲**
[۱, ۵]	۲۰	۱۶۰/۵	۱۶۳/۹	۰/۹۷۹۵
	۴۰	۵۶۱/۸	۵۸۴/۸	۰/۹۶۰۷
	۶۰	۱۲۰۵/۳	۱۲۹۱/۷	۰/۹۳۳۱
	۸۰	۲۱۲۴/۰	۲۳۴۰/۹	۰/۹۰۷۴
	۱۰۰	۳۵۳۴/۶	۳۹۶۸/۳	۰/۸۹۰۷
	۲۰۰	۱۳۷۳۰/۶	۱۵۵۲۳/۶	۰/۸۸۴۵
			میانگین	**۰/۹۲۶۰**

همان‌طور که مشاهده می‌شود الگوریتم HGA عملکرد بهتری نسبت به الگوریتم GAMARB دارد. کارایی الگوریتم HGA برای مسایل با کارهای با اندازه کوچک قابل توجه‌تر از مسایل با کارهای دارای اندازه بزرگ است. برای مسایل با کارهای دارای اندازه بزرگ دو الگوریتم تقریبا عملکرد یکسانی دارند و جواب‌هایی با

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

کیفیت یکسان حاصل شده است. وقتی اندازه کارها بزرگ است امکان تشکیل انباشته‌های متنوع از کارها کمتر می‌شود و به همین جهت کارایی دو الگوریتم به هم نزدیک‌تر است. همچنین مشاهده می‌شود که تفاوت عملکرد الگوریتم‌ها با افزایش تعداد کارها افزایش می‌یابد. نتایج بر اساس اندازه کارهای مختلف در شکل ۵ نمایش داده شده است.



شکل ۵. مقایسه عملکرد الگوریتم‌های HGA و GAMARB برای نمونه مسایل بزرگ

٦ جمع‌بندی و پیشنهاد مطالعات آتی

در این مقاله، مساله زمان‌بندی تولید به‌هنگام در یک سیستم پردازنده انباشته تک ماشینه با هدف کمینه کردن مجموع هزینه‌های زودهنگامی و دیرهنگامی کارها مورد بررسی قرار گرفت. دو الگوریتم HGA و GRASP برای تولید جواب‌های نزدیک به بهینه ارایه شد. در این الگوریتم‌ها، یک رویکرد برنامه‌ریزی پویا برای زمان‌بندی انباشته‌ها به کار گرفته شد. در نهایت از آزمایشات محاسباتی برای بررسی کارایی الگوریتم‌های توسعه داده شده استفاده شد. طبق نتایج به دست آمده، متوسط خطای الگوریتم HGA برابر ۶/۸۲٪ و این مقدار برای الگوریتم GRASP برابر ۱۱/۶۴٪ است. محدودیتی که در انجام این پژوهش به صورت جدی وجود داشت، عدم دسترسی به داده‌های واقعی در صنعت نیمه هادی بود. با توجه به اهمیت مسایل زمان‌بندی تولید انباشته با در نظر گرفتن استراتژی‌های تولید به هنگام، همچنان خلأهای مطالعاتی زیادی در این زمینه وجود دارد. در نظر گرفتن شرایط پویایی مانند زمان دسترسی به کارها و خرابی ماشین‌آلات می‌تواند در جهت نزدیک‌تر شدن به شرایط واقعی کمک کننده باشد. همچنان ارایه الگوریتم‌های دقیق و روش‌های تولید حد پایین برای مساله دارای جذابیت است. استفاده از روش‌های تجزیه مانند روش تجزیه دنتزیگ-ولف برای حل مساله می‌تواند یکی از گزینه‌های مناسب برای تحقیقات آتی باشد. در نظر گرفتن استراتژی‌های تولید به‌هنگام در دیگر محیط‌های پردازش انباشته نیز می‌تواند یکی دیگر از موضوعات تحقیقات آتی باشد.

منابع

[1] Dobson, G., Nambimadom, R. S. (2001). The batch loading and scheduling problem. Operations research, 49(1), 52-65.

[2] Uzsoy, R. (1994). Scheduling a single batch processing machine with non-identical job sizes. International Journal of Production Research, 32(7), 1615-1635.

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

[3] Dupont, L., Jolai, F. (1998). Minimizing makespan on a single batch processing machine with non-identical job sizes. European Journal of Automation Systems, 32(1), 31-40.
[4] Jolai, F., Dupont, L. (1998). Minimizing mean flow times criteria on a single batch processing machine with non-identical jobs sizes. International Journal of Production Economics, 55(3), 273-280.
[5] Melouk, S., Damodaran, P., Chang, P. Y. (2004). Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing. International Journal of Production Economics, 87(2), 141-147.
[6] Damodaran, P., Kumar Manjeshwar, P., Srihari, K. (2006). Minimizing makespan on a batch-processing machine with non-identical job sizes using genetic algorithms. International Journal of Production Economics, 103(2), 882-891.
[7] Husseinzadeh Kashan, A., Karimi, B., Jolai, F. (2006). Effective hybrid genetic algorithm for minimizing makespan on a single-batch-processing machine with non-identical job sizes. International Journal of Production Research, 44(12), 2337-2360.
[8] Rafiee Parsa, N., Karimi, B., Husseinzadeh Kashan, A. (2010). A branch and price algorithm to minimize makespan on a single batch processing machine with non-identical job sizes. Computers & Operations Research, 37(10), 1720-1730.
[9] Cheng, B., Li, K., Chen, B. (2010). Scheduling a single batch-processing machine with non-identical job sizes in fuzzy environment using an improved ant colony optimization. Journal of Manufacturing Systems, 29(1), 29-34.
[10] Chen, H., Du, B., Huang, G. Q. (2011). Scheduling a batch processing machine with non-identical job sizes: a clustering perspective. International Journal of Production Research, 49(19), 5755-5778.
[11] Mathirajan, M., Bhargav, V., Ramachandran, V. (2010). Minimizing total weighted tardiness on a batch-processing machine with non-agreeable release times and due dates. The International Journal of Advanced Manufacturing Technology, 48(9), 1133-1148.
[12] Wang, H. M. (2011). Solving single batch-processing machine problems using an iterated heuristic. International Journal of Production Research, 49(14), 4245-4261.
[13] Malapert, A., Gueret, C., Rousseau, L. M. (2012). A constraint programming approach for a batch processing problem with non-identical job sizes. European Journal of Operational Research, 221(3), 533-545.
[14] Queiroga, E., Pinheiro, R. G. S., Christ, Q., Subramanian, A., Pessoa, A. (2020). Iterated local search for single machine total weighted tardiness batch scheduling. Journal of Heuristics, 27(1), 353-438.
[15] Damodaran, P., Ghrayeb, O., Guttikonda, M. C. (2013). GRASP to minimize makespan for a capacitated batch-processing machine. The International Journal of Advanced Manufacturing Technology, 68(1), 407-414.
[16] Jia, Z. h., Leung, J. Y. T. (2014). An improved meta-heuristic for makespan minimization of a single batch machine with non-identical job sizes. Computers & Operations Research, 46, 49-58.
[17] M. Al-Salamah, M. (2015). Constrained binary artificial bee colony to minimize the makespan for single machine batch processing with non-identical job sizes. Applied Soft Computing, 29, 379-385.
[18] Cabo, M., Possani, E., Potts, C. N., Song, X. (2015). Split–merge: Using exponential neighborhood search for scheduling a batching machine. Computers & Operations Research, 63, 125-135.
[19] Rafiee Parsa, N., Karimi, B., Keshavarz, T. (2016). Minimizing total earliness and tardiness on a batch processing machine using a neural network approach. 12th International Conference on Industrial Engineering.
[20] Beldar, P., Framinan, J. M., Ardakani, A. (2019). Minimization of total completion time on a batch processing machine with arbitrary release dates: an effectual teaching–learning based optimization approach. Production Engineering, 13(5), 557-566.
[21] Mousavi, M., Hajiaghaei–Keshteli, M., Tavakoli Moghaddam, R. (2018). Integrated scheduling of production on single machine environment and air transportation considering charter flights and time windows for due date with two efficient meta-heuristics algorithm. Journal of Operational Research and Its Applications, 15(2), 19-40.
[22] Beheshtinia, M., Hassani Bidgoli, M. (2017). Scheduling in Flexible Jobshop Environment Considering Assembling and Sequence Dependent Processing Times. Journal of Operational Research and Its Applications, 13(4), 21-37.
[23] Li, Z., Chen, H., Xu, R., Li, X. (2015). Earliness–tardiness minimization on scheduling a batch processing machine with non-identical job sizes. Computers & Industrial Engineering, 87(1), 590-599.

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]

[24] Rafiee Parsa, N., Karimi, B., Husseini, S. M. (2017). Exact and heuristic algorithms for the just-in-time scheduling problem in a batch processing system. Computers & Operations Research, 80(1), 173-183.
[25] Brucker, P., et al. (1998). Scheduling a batch machine. Journal of Scheduling, l(1), 31–54.
[26] Hart, W. E., Krasnogor, N., Smith, J. E. (2005). Memetic evolutionary algorithms. Recent advances in memetic algorithms, 3-27: Springer.
[27] Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech concurrent computation program, C3P Report, 826.
[28] Corne, D. (1999). New ideas in optimization. UK: McGraw-Hill Ltd.
[29] Feo, T. A., Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters, 8(2), 67-71.
[30] Taguchi, G. (1986). Introduction to quality engineering: designing quality into products and processes: Asian productivity organization.

[Downloaded from jamlu.lahijan.iau.ir on 2026-11-02]
[DOR: 20.1001.1.22517286.2021.18.2.7.2]
[DOI: 10.52547/jamlu.18.2.1]