حل مسئله زمانبندی کمینه سازی مجموع تاخیرات وزندار کارها روی یک ماشین با استفاده از آتاماتای یادگیر و ترکیب آن با الگوریتم ژنتیکی

کیوان اصغری[۱] ؛ محمد رضا میبدی[۲]

چکیده

در این مقاله سعی نموده ایم تا مسئله زمانبندی مجموع تاخیرات وزندار تک ماشینه را که یک مسئله **NP-Hard** است، با استفاده از آتاماتای یادگیر و ترکیب آن با الگوریتم ژنتیکی حل کنیم. ابتدا به بررسی روشهای قبلی حل این مسئله مانند روشهای تکراری، حریصانه، کاهشی، الگوریتمهای ژنتیکی و ممتیکی پرداخته و سپس روشهای آتاماتای یادگیر و الگوریتم ترکیبی را برای حل این مسئله پیاده سازی کرده ایم. در روش ترکیبی، هر کروموزوم الگوریتم ژنتیکی از یک آتاماتای یادگیر تشکیل یافته است که در حین فرایند تکامل ژنتیکی، عمل یادگیری انجام داده و سعی در بهبود راه حل نهفته در خود دارد. نتایج بدست آمده از الگوریتم های مختلف برای نمونه های ۴۰، ۵۰ و ۱۰۰ کاره از این مسئله را که از کتابخانه **OR** اخذ شده اند، مورد مقایسه قرار داده و اقدام به تنظیم پارامترهای الگوریتم ترکیبی نموده ایم. نتایج بدست آمده، حاکی از برتری الگوریتم ترکیبی نسبت به تمام الگوریتمهای قبلی از لحاظ کیفیت جوابهای بدست آمده می باشند.

کلمات کلیدی

مسئله زمانبندی مجموع تاخیرات وزندار تک ماشینه، آتاماتای یادگیر، الگوریتم ژنتیکی

Solving Single Machine Total Weighted Tardiness Scheduling Problem using Learning Automata and Combination of it with Genetic Algorithm

Keyvan Asghari; Mohammad Reza Meybodi
k.asghari@yahoo.com, mmeybodi@aut.ac.ir

ABSTRACT

We have tried to solving the Single Machine Total Weighted Tardiness Scheduling Problem using Learning Automata and combination of it with Genetic Algorithm In this paper. This problem is strongly NP-Hard. We first have studied some previous methods for solving this problem, for example Iterative, Greedy, Descent, Genetic and Memetic algorithms, and then implemented the Learning Automata and combinative algorithm for this problem. In the combinative algorithm, each chromosome contains one Learning Automata that performs learning actions along the genetic evolution and try to improve the hidden solution in itself. We have compared the results obtained from various algorithms for instances of this problem with 40, 50 and 100 job that given from OR library and we have performed the parameter tuning for combinative algorithm. Obtained results show the influence of combinative algorithm to all previous algorithms from the viewpoint of quality.

KEYWORDS

Single Machine Total Weighted Tardiness Scheduling Problem, Learning Automata, Genetic Algorithm

---

[۱] گروه کامپیوتر، دانشگاه آزاد اسلامی واحد خامنه، k.asghari@yahoo.com
[۲] دانشکده مهندسی کامپیوتر و فناوری اطلاعات، دانشگاه صنعتی امیرکبیر، mmeybodi@aut.ac.ir

۱. مقدمه

در این مقاله، یکی از مشهورترین مسائل بهینه سازی بنام مسئله زمانبندی مجموع تاخیرات وزندار تک ماشینه (SMTWTSP)[۱] مورد بررسی قرار می گیرد. کمینه کردن مجموع تاخیرات، اولین بار توسط تانسل مورد توجه قرار گرفت [۱]. این مسئله، توسط شماری از الگوریتمها مانند الگوریتمهای انشعاب و تحدید [۲, ۳] و الگوریتمهای برنامه نویسی پویا [۴] ، مورد بررسی قرار گرفته است تا راه حلهای دقیق که بهینگی آنها تضمین شود، تولید گردند. اما الگوریتمهای انشعاب و تحدید دارای نیازهای محاسباتی از لحاظ زمان هستند که مکررا بصورت نمایی یا چند جمله ای با درجه بالا متناسب با اندازه مسئله n، رشد می کند و الگوریتمهای برنامه نویسی پویا نیز به واسطه نیازمندیهای حافظه کامپیوتر بخصوص وقتی تعداد کارها بیشتر از ۵۰ می شود، محدودند[۵]. پس از آن نیز این مسئله بطور گسترده ای توسط روشهای اکتشافی مورد مطالعه قرار گرفته است. این روشهای اکتشافی شامل قوانین توزیع امکان اکتشافی[۲] [۶] و روشهای اکتشافی جستجوی محلی[۳] هستند. از آنجاییکه قانون توزیع امکان واحدی برای تمام محیطهای مسئله وجود ندارد و به بیان دیگر قوانین توزیع امکان، بطور سازگاری راه حلهای با کیفیت خوب را ارائه نمی کنند ، در سالهای اخیر توجه زیادی به روشهای اکتشافی جستجوی محلی اختصاص یافته است[۷, ۸]. این روشهای اکتشافی جستجوی محلی اساسا شامل روشهای جستجوی همسایه مانند روشهای کاهشی، نرم کردن شبیه سازی شده، پذیرش آستانه، جستجوی تابو [۳, ۵, ۹] و الگوریتمهای ژنتیکی هستند[۵, ۱۰, ۱۱]. همچنین مادوریرا [۱۱] ، راه حلهای خوبی را در زمان کمی توسط یک الگوریتم ژنتیکی رمز شده با جایگشت طبیعی برای مسائل با ۴۰ و ۵۰ کار بدست آورده است. اوسی [۱۰] یک الگوریتم ژنتیکی را برای مسائل با ۲۰۰ کار پیشنهاد کرده است که از قوانین تسلط کلی و محلی برای بهبود ساختار همسایگی فضای جستجو استفاده کرده و نتایج تقریبا یکسانی را با نتایج کراولز [۵] بدست آورده است. در [۱۲, ۱۳] نیز یک الگوریتم ژنتیکی با کدبندی کروموزومها بصورت جایگشت طبیعی از کارها، ارائه شده است. آخرین کار موجود در مورد حل مسئله مجموع تاخیرات وزندار با استفاده از الگوریتمهای ژنتیکی، در [۱۴] ارائه شده است. در این مقاله مفهوم جدیدی از عملگرهای ژنتیکی برای مسایل زمانبندی ارائه شده و کارایی عملگرهای مختلف جابجایی و جهش برای مسائل زمانبندی، مورد آزمایش قرار گرفته است.

فرانکا در [۱۵]، یک الگوریتم ممتیکی برای مسئله زمانبندی مجموع تاخیرات تک ماشینه با موعدهای مقرر و زمان راه اندازی وابسته به ترتیب کارها، پیشنهاد کرده است. در [۱۶] نیز یک الگوریتم ممتیکی برای حل مسئله مجموع تاخیرات وزندار تک ماشینه، پیشنهاد شده است. در [۱۷]، یک الگوریتم تکاملی جستجوی متمرکز پیشنهاد شده که کاملا مشابه الگوریتم ممتیکی ارائه شده در مرجع قبلی است، با این تفاوت که از یک روش اکتشافی بنام هیوریستیک فاز بازگشتی به همراه روش تصادفی برای مقدار دهی اولیه راه حلها استفاده می کند و این هیوریستیک در [۸] برای بررسی روی تاثیر فاکتورهای RDD[۴] و TF[۵] بکار رفته که در قسمتهای بعدی به توضیح آنها خواهیم پرداخت. پاتس و وان واسن هو و کراولز چنین تشخیص داده اند که برای نمایش جایگشت طبیعی، همسایگی تعویضی به دیگر همسایگی ها برای مسئله زمانبندی مجموع تاخیرات وزندار ترجیح داده می شود [۵]. اما دایناسرچ[۶] نیز یک تکنیک جستجوی همسایگی است که مهمترین خصوصیت آن توانایی جستجوی همسایگی های دارای اندازه نمایی در زمان چند جمله ای با بهره برداری از ساختار مسئله است. کنگرام، پات و واندولد نیز در [۱۸]، روشی تکراری مبتنی بر همسایگی دایناسرچ برای تعویض کارها در یک دنباله از ترتیب اجرای آنها ارائه کرده اند. گراسو، دللا کروس و تادی بر اساس کار کنگرام، پات و واندولد، یک همسایگی دایناسرچ توسعه یافته را برای مسئله زمانبندی مجموع تاخیرات تک ماشینه ایجاد کرده اند که توسط عملگرهای تعویض دوبدو عمومی GPI[۷] بدست آمده است و در نتیجه همسایگی مبتنی بر تعویض موجود در کار کنگرام را گسترش داده اند[۹]. ما نیز در این مقاله از آتاماتای یادگیر و ترکیب آن با الگوریتم ژنتیکی برای حل مسئله زمانبندی مجموع تاخیرات وزندار استفاده می کنیم. نوع اولیه و ابتدایی از الگوریتم ترکیبی، قبلا برای حل مسائلی مانند تناظر گراف و بهینه سازی پرس و جو های پایگاه داده ای بکار رفته است [۱۹, ۲۰].

۲. تعریف مسئله

مسئله بهینه سازی زمانبندی مجموع تاخیرات وزندار تک ماشینه به این صورت تعریف می شود: تعداد N کار را در نظر بگیرید که قرار است بدون وقفه روی یک ماشین که بطور پیوسته قابل دسترس است، پردازش شوند و این ماشین در هر زمان فقط می تواند یک کار را پردازش کند. هر کار j، در زمان صفر برای پردازش قابل دسترس بوده و دارای یک زمان پردازش مثبت p_j ، یک وزن مثبت w_j و یک زمان مقرر d_j است. برای یک ترتیب کار مفروض، تاخیر کار j بصورت مقابل تعریف می شود. $T_j=\max\{0,C_j-d_j\}$ بطوریکه $C_j = \sum_{i=1}^{j} p_i$ زمان اتمام کار j است. هدف مسئله مجموع تاخیرات وزندار، پیدا کردن ترتیب پردازشی از تمام کارها است که این ترتیب، یک زمانبندی است که در آن مجموع تاخیرات وزندار تمام کارها $\sum_{j=1}^{n} w_j T_j$ کمینه شود. بنابراین این مسئله، زمانبندی n کار روی یک ماشین برای کمینه کردن مجموع تاخیرات وزندار تمام کارهاست. برای وزنهای اختیاری مثبت، این مسئله قویاً NP-hard است [۲۱].

۳. تعدادی از الگوریتمهای قبلی حل مسئله مجموع تاخیرات وزندار

۱.۳. الگوریتم حریصانه

در روش حریصانه، در هر مرحله کاری که دارای بهترین معیار حریصانه در حالت فعلی است، انتخاب می شود. روشهای مختلفی برای انتخاب این کار وجود دارند که بعنوان توابع کشف کننده یا قوانین توزیع امکان اکتشافی در جستجو مطرح می شوند و در هر مرحله بهترین کار را با در نظر گرفتن معیار بخصوص خود انتخاب می کنند. بعضی از معروفترین این قوانین در زیر آمده اند:

- EDD[۸] (زودترین موعد مقرر): در هر مرحله، کار دارای زودترین موعد مقرر انتخاب می شود.
- WSPT[۹] (کوتاهترین زمان پردازش وزندار): ابتدا نرخ Sj=pj/wj برای هر کار j ، محاسبه شده و کارها با ترتیب صعودی نرخهایشان زمانبندی می شوند.
- SPT[۱۰] (کوتاهترین زمان پردازش): کارها با ترتیب صعودی از زمانهای پردازششان زمانبندی می شوند.
- BWF[۱۱] (ابتدا بزرگترین وزن): کارها با ترتیب نزولی از وزنهایشان زمانبندی می گردند.
- AU[۱۲] (ضرورت آشکار) [۱۳]: اولویت ضرورت آشکار AUj بصورت زیر برای هر کار j مورد محاسبه قرار گرفته و کارها با ترتیب نزولی از این اولویت، مرتب می شوند. در اینجا k پارامتر نگاه به جلو (look-ahead) را نشان می دهد و مطابق فشار موعد مقرر تنظیم می شود. p بار، متوسط زمان پردازش بوده و t زمان فعلی است. این روش اکتشافی دارای پیچیدگی زمانی همسان با EDD و WSPT است.

$$AU_j = (\frac{w_j}{p_j})\exp(\frac{-\max\{0,\ d_j - t - p_j\}}{k \cdot \overline{p}})$$

۲.۳. روشهای کاهشی

این روشها راهکارهای معاوضه دوبدو شامل DES و DESO هستند[۵, ۷]. آنها با دنباله هایی بدست آمده توسط اعمال قانون توزیع امکان AU شروع می کنند و کارهای (u,v) از (۲, ۱) به (۳, ۱) , (۴, ۱), ... , و (n, n-1) را با یکدیگر تعویض می کنند که (u,v) به معنی تعویض کار موجود در موقعیت u ام با کار موجود در موقعیت v ام می باشد. روش DES ، روش کاهشی سخت گیری است که تنها معاوضه کارهای دوبدو مورد قبول واقع می شوند که منجر به کاهش در تابع هدف که در اینجا مجموع تاخیرات وزندار است، گردد. اما در روش DESO (روش کاهشی با تعویض صفر)، تعویض کارهای دوبدو که منجر به تغییر در تابع هدف نشود نیز بعلاوه تعویضاتی که باعث کاهش در تابع هدف می شود، مورد پذیرش واقع می شود.

۳.۳. جستجوی تعویضی مبتنی بر برنامه نویسی پویا (دایناسرچ)

در الگوریتمهای اکتشافی جستجوی محلی، معمولاً همسایگی یک راه حل توسط بکار بردن عملگرهای تعویض دوبدو برای دنباله کارها محاسبه می شود. فرض کنید σ=(σ(1), …, σ(n)) دنباله ای باشد که ترتیب پردازش فعلی کارها را تعریف می کند که σ(i) عبارت از کار موجود در محل i برای i=1, …, n است. همسایگی تعویضی دایناسرچ شامل تمام راه حلهایی است که می توان از جایگشت مفروض σ با مجموعه ای از حرکتهای تعویض مستقل دوبدو بدست آورد تا چند مرحله بهبود، بطور همزمان انجام گیرد. دو حرکت که کار σ(i) را با کار σ(j) و کار σ(k) را با کار σ(l) بترتیب عوض می کنند مستقل نامیده می شوند اگر max{i,j}<min{k,l} یا min {i,j}> max {k,l} . در این مقاله نیز همسایگی دایناسرچ GPI طبق [۹] برای مقایسه با الگوریتمهای دیگر بطور کامل پیاده سازی شده است. یک دنباله اولیه S را در نظر بگیرید. کارها را بصورت S=(1,2,3,…,n) شماره گذاری کنید. چهار عملگر GPI بنامهای API[۱۳](تعویض دوبدوی همجوار)، NAPI[۱۴] (تعویض دوبدوی نا همجوار)، EBSR[۱۵] (استخراج و درج دوباره با شیفت به سوی عقب) و EFSR[۱۶] (استخراج و درج دوباره با شیفت به سوی جلو) داریم که روی دنباله ها بصورت زیر عمل می کنند. عملگرهای API و NAPI نیز در قالب عملگر SWAP ادغام شده اند. با فرض جفتی از اندیسها i<j بطوریکه S=σiπjω داریم:

$\mathrm{API}(i,j)$: $\sigma ij\omega \Rightarrow \sigma ji\omega$ (requires $\pi = \emptyset$),

$\mathrm{NAPI}(i,j)$: $\sigma i\pi j\omega \Rightarrow \sigma j\pi i\omega$,

$\mathrm{EBSR}(i,j)$: $\sigma i\pi j\omega \Rightarrow \sigma ji\pi\omega$,

$\mathrm{EFSR}(i,j)$: $\sigma i\pi j\omega \Rightarrow \sigma\pi ji\omega$.

۴.۳. الگوریتم ممتیکی

الگوریتم ممتیکی ترکیبی میان جستجوی سراسری مبتنی بر جمعیت و جستجوی محلی اکتشافی است که روی هر یک از افراد صورت می گیرد. هر فرد نشان دهنده یک راه حل ممکن است و جمعیت مذکور توسط عملگرهای جابجایی و جهش همراه با بهبود محلی با استفاده از هیوریستیکها

رشد می کند. الگوریتم تکاملی جستجوی متمرکز پیشنهاد شده در [۱۷]، دنباله ای بدست آمده از الگوریتم حریصانه (با هیوریستیک فاز بازگشتی) و دنباله‌ای تولید شده بصورت تصادفی را بعنوان والدین اولیه بکار می برد. روش نمایش راه حل در الگوریتم ممتیکی، جایگشت طبیعی کارهای از شماره ۱تا n بوده و تابع برازندگی الگوریتم ممتیکی، مجموع تاخیرات وزندار $Z(\sigma)$ برای دنباله مفروض σ است و بایستی کمینه شود.

۵.۳. الگوریتم ژنتیکی

الگوریتمهای ژنتیکی با تقلید فرایند تکامل بیولوژیکی، یک فضای جستجو را کاوش می کنند. این الگوریتم ها بر روی جمعیتی از راه حلهای بالقوه یاکروموزوم ها که هر یک می‌توانند بعنوان پاسخی از مسئله تلقی شوند، با اعمال عملگرهای ژنتیکی به جستجوی راه حل نهایی می پردازند. در الگوریتم ژنتیکی برای مسئله مجموع تاخیرات وزندار تک ماشینه، از جایگشت طبیعی کارها برای نمایش ساختار کروموزومهای جمعیت استفاده شده است. بمنظور تخمین یک راه حل بهینه بصورت دقیقتر، جمعیت اولیه کروموزومها توسط قوانین توزیع امکان اکتشافی گفته شده در بخش ۳-۱ در ترکیب با روش تصادفی ایجاد می شوند[۲۲]. هنگامی که یک جمعیت ایجاد شد، هر کروموزوم ارزیابی می شود و برازندگی آن با محاسبه مجموع تاخیرات وزندار تعیین می شود. با استفاده از روشهای انتخاب، کروموزومها (والدین) از جمعیت انتخاب می شوند تا با هم ترکیب شده و کروموزومهای جدید (فرزندان) را برای بکار بردن عملگرهای ژنتیکی تولید کنند. در پیاده سازی ما عملگرهای انتخاب چرخ رولت و رتبه بندی بکار رفته اند. نقش یک عملگر جابجایی، ترکیب عناصر دو کروموزوم والد برای تولید یک یا چند کروموزوم فرزند است. در این عملگر دو جایگشت والد انتخاب، و سپس طبق یکی از روشهای جابجایی، عمل جابجایی روی والدین انجام می گیرد. با این عمل دو جایگشت جدید حاصل می شود که اصطلاحا فرزندان دو جایگشت والد خوانده می شوند. نقش یک عملگر جهش نیز فراهم کردن و نگهداری تنوع و گوناگونی در یک جمعیت به گونه ای است که دیگر عملگرها بتوانند به کارشان ادامه دهند. عملگر جایگزینی بکاررفته در الگوریتم ژنتیکی پیاده سازی شده برای این مسئله، مبتنی بر نخبه سالاری است.

۴. آتاماتای یادگیر

روش کار آتاماتای یادگیر در یادگیری عبارت است از تعیین اقدام بهینه از مجموعه محدودی از اقدامهای از پیش تعریف شده که قابل انجام در یک محیط تصادفی می باشند. یک آتاماتا در یک مدار بازخوردی با محیط تصادفی ناشناس، تراکنش انجام می دهد. آتاماتا در هر لحظه اقدامی را از مجموعه اقدامهای خود انتخاب و به محیط اعلام می نماید. محیط در پاسخ به اقدام انجام شده، یک خروجی از مجموعه خروجی های تعریف شده را تولید و به آتاماتا اعلام می نماید و آتاماتا با دریافت پاسخ محیط، شیوه تصمیم گیری خود را در انتخاب اقدام بعدی به هنگام می نماید. فرض می شود که بین هر اقدام آتاماتا و پاسخ محیط یک رابطه احتمالی وجود داشته باشد و این رابطه که در واقع مشخصات داخلی محیط است، در طول یادگیری توسط آتاماتا شناخته می شود [۲۳]. برای یک مسئله زمانبندی با n کار، !n جایگشت مختلف وجود دارد و در صورتی که از آتاماتای یادگیر برای حل مساله زمانبندی استفاده شود، آتاماتای یادگیر باید !n اقدام داشته باشد که تعداد زیاد اقدام ها باعث کاهش سرعت همگرایی می شود و به همین منظور ساختار خاصی از آتاماتای یادگیر بنام آتاماتای مهاجرت اشیاء[۱۷] توسط اومن[۱۸] و ما[۱۹] پیشنهاد شده است [۲۴].

۵. الگوریتم ترکیبی آتاماتای یادگیر و الگوریتم ژنتیکی (GA+LA)

در الگوریتم ترکیبی هر یک از اعضای جمعیت در سیر تکاملی الگوریتم ژنتیکی دارای قابلیت یادگیری خواهند بود و با توجه به این قابلیت می توانند راه حل نهفته در ساختار خود را بطور مستقل از سایر اعضای جمعیت، بهبود بخشند. بعلاوه قابلیت یادگیری مانع از بدام افتادن الگوریتم در حداقل های محلی شده و سرعت رسیدن به جواب نیز افزایش پیدا می کند. در ادامه بطور خلاصه الگوریتم ترکیبی برای حل مسئله زمانبندی را مطرح می کنیم.

۱.۵. کد گذاری راه حل

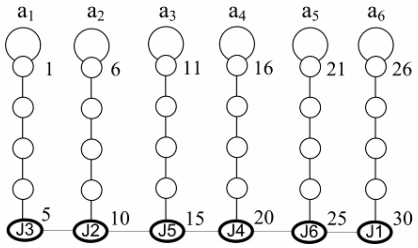
در الگوریتم پیشنهادی هر کروموزوم توسط یک آتاماتای یادگیر از نوع مهاجرت اشیاء نشان داده می شود بطوریکه هر کدام از محتوی ژن های کروموزوم بصورت یک شیء، به یکی از اقدام های آتاماتا نسبت داده می شود و در عمق مشخصی از آن اقدام قرار می گیرند. در این آتاماتا $\underline{\alpha}=\{\alpha_1,\ldots,\alpha_k\}$ مجموعه اقدام های مجاز برای آتاماتای یادگیر است. این آتاماتا k اقدام دارد (تعداد اقدام های این آتاماتا برابر با تعداد کارهایی است که باید زمانبندی شوند). اگر کار s از مجموعه کارها در اقدام m قرار گرفته باشد، در اینصورت کار s در ترتیب زمانبندی کارها، m امین کاری خواهد بود که انجام می شود. $\underline{\phi}=\{\phi_1, \phi_2, \ldots, \phi_{KN}\}$ مجموعه وضعیت ها و N عمق حافظه برای آتاماتا می باشد. مجموعه وضعیت های این آتاماتا به K زیر مجموعه $\{\phi_1, \phi_2, \ldots, \phi_N\}$ و $\{\phi_{N+1}, \phi_{N+2}, \ldots, \phi_{2N}\}$ و ... و $\{\phi_{(K-1)N+1}, \phi_{(K-1)N+2}, \ldots, \phi_{KN}\}$ افراز می شود و کارها بر اساس این که در

کدام وضعیت قرار داشته باشند دسته بندی می گردند. اگر کار s از مجموعه کارها در مجموعه وضعیت های $\{\phi_{(j-1)N+1}, \phi_{(j-1)N+2}, \dots, \phi_{jN}\}$ قرار داشته باشد در اینصورت کار s در ترتیب زمانبندی کارها، j امین کاری خواهد بود که انجام می شود. در مجموعه وضعیت های اقدام j، به وضعیت $\phi_{(j-1)N+1}$ وضعیت داخلی و به وضعیت ϕ_{jN}، وضعیت مرزی گفته می شود. کاری که در وضعیت $\phi_{(j-1)N+1}$ قرار دارد، کار با اهمیت بیشتر وکار در وضعیت ϕ_{jN}، کار با اهمیت کمتر نامیده می شود. در اثر پاداش دادن یا جریمه کردن یک اقدام، وضعیت کار وابسته به آن اقدام تغییر می کند. اگر کاری در وضعیت مرزی یک اقدام قرار داشته باشد، جریمه شدن اقدام آن باعث تغییر اقدامی که کار به آن وابسته است، می شود و در نتیجه باعث ایجاد جایگشت جدیدی می گردد. با توجه به اینکه هر یک از کروموزومها در الگوریتم ما از یک آتاماتای یادگیر مهاجرت اشیاء تشکیل شده است، لذا اتصالات مختلفی برای آتاماتاهای موجود در جمعیت، در نظر گرفته شده است. ما چهار نوع اتصال آتاماتای ستلین، کرایلو، کرینیسکی [۲۳] ، اومن و همچنین ترکیبی از آنها را در قالب یک جمعیت ناهمگن دارای آتاماتاهای با اتصالات مختلف، مورد توجه قرار داده ایم. حال بعنوان مثالی از یک مسئله زمانبندی دارای ۶ کار، مجموعه کارهای J_i برای i از ۱ تا ۶ را در نظر می گیریم. هر کار دارای یک زمان پردازش p_i یک موعد مقرر d_i و یک وزن w_i می باشد. اطلاعات مربوط به کارهای این مسئله در جدول ۱ آمده اند.

جدول ۱- اطلاعات مربوط به یک مسئله دارای ۶ کار برای زمانبندی

I	p_i	d_i	w_i
1	8	26	4
2	12	28	1
3	6	32	6
4	10	35	5
5	3	38	1
6	11	48	4

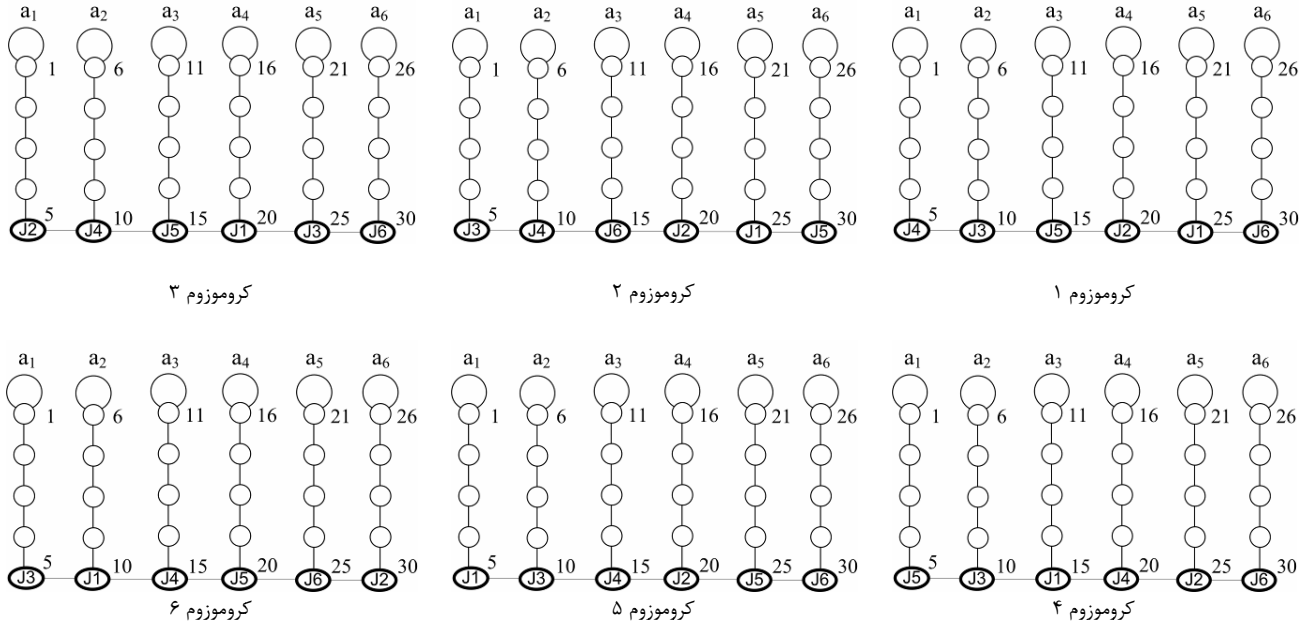
جایگشت (۱ ، ۶ ، ۴ ، ۵ ، ۲ ، ۳) را بعنوان یک جایگشت اولیه برای زمانبندی ترتیب اجرای شش کار فوق در نظر بگیرید. نحوه نمایش این زمانبندی با آتاماتای مهاجرت اشیای مبتنی بر اتصالات آتاماتای ستلین بصورت شکل ۱ است. هر آتاماتا دارای ۶ اقدام $\{\alpha_1,\alpha_2,\alpha_3,\alpha_4,\alpha_5,\alpha_6\}$ (به تعداد کارها) و عمق ۵ می باشد. مجموعه وضعیت های $\{1,6,11,16,21,26\}$ وضعیت های داخلی و مجموعه وضعیت های $\{5,10,15,20,25,30\}$ وضعیت های مرزی آتاماتا هستند. در ابتدا هر یک از کارها در وضعیت مرزی اقدام مربوطه قرار دارند. در الگوریتم ژنتیکی جدید، هر ژن از کروموزوم معادل یک اقدام آتاماتا می باشد و لذا می توان در ادامه این دو واژه را به جای یکدیگر بکار برد.



شکل ۱-نمایش جایگشت (۱, ۶, ۴, ۵, ۲, ۳) بوسیله آتاماتای یادگیر مهاجرت اشیای مبتنی بر اتصالات آتاماتای ستلین

۲.۵. جمعیت اولیه

با فرض اینکه تعداد اعضای جمعیت p باشد، p-1 عضو جمعیت با ایجاد p-1 جایگشت تصادفی تولید می شوند. برای تولید آخرین عضو جمعیت، از یکی از قوانین توزیع امکان اکتشافی مانندEDD, SPT, WSPT, BWF یا AU استفاده می کنیم. به عنوان مثال نحوه تشکیل جمعیت اولیه برای مسئله دارای ۶ کار مطرح شده با فرض p=6 برای تعداد اعضای جمعیت در ادامه توضیح داده شده است. پنج عضو اول جمعیت به وسیله پنج جایگشت تصادفی (۶ ، ۱ ، ۲ ، ۵ ، ۳ ، ۴) ، (۵ ، ۱ ، ۲ ، ۶ ، ۴ ، ۳) ، (۶ ، ۳ ، ۱ ، ۵ ، ۴ ، ۲) ، (۶ ، ۲ ، ۴ ، ۱ ، ۳ ، ۵) ، (۶ ، ۵ ، ۲ ، ۴ ، ۳ ، ۱) تولید می شود. برای ایجاد جایگشت ششم از روش اکتشافی اولویت آشکار AU استفاده می کنیم که در بخش های قبلی توضیح داده شده است. حاصل استفاده از این روش جایگشت (۲ ، ۶ ، ۵ ، ۴ ، ۱ ، ۳) است. جمعیت اولیه تولید شده برای مسئله مطرح شده بصورت شکل ۲ می باشد. در ابتدا هر کار در وضعیت مرزی اقدام خود قرار دارد.



شکل ۲- جمعیت اولیه برای مسئله زمانبندی دارای ۶ کار

۳.۵. تابع برازندگی

در الگوریتم های ژنتیکی، تابع برازندگی شاخص زنده ماندن کروموزوم ها است. در مسئله زمانبندی کارها، هدف یافتن جایگشتی (زمانبندی) از کارها مثل σ است که مجموع تاخیرهای وزندار بدست آمده از اجرای تمام کارها کمینه باشد. لذا تابع برازندگی $f(\sigma)$ درمساله زمانبندی کارها به صورت زیر تعریف می شود.

$$f(\sigma) = \frac{1}{Total\ \ Weighted\ \ Tardiness\ (\sigma)}$$

۴.۵. عملگرهای الگوریتم ترکیبی

الف) عملگر انتخاب: برای انتخاب آتاماتاهای یادگیر (کروموزوم) برای عملگرهای جهش و جابجایی، دو روش انتخاب چرخ رولت و رتبه بندی در الگوریتم ترکیبی، پیاده سازی شده است.

از آنجا که در الگوریتم ترکیبی، هر کروموزوم به صورت یک آتاماتای یادگیر نمایش داده می شود عملگرهای جابجایی و جهش مشابه عملگرهای سنتی الگوریتم ژنتیک نیستند. عملگر جایگزینی بکار رفته در الگوریتم های پیاده سازی شده برای این مسئله، مبتنی بر نخبه سالاری است.

ب) عملگر ترکیب یا جابجایی: برای الگوریتم ترکیبی و همچنین الگوریتمهای ژنتیکی و ممتیکی، چهار عملگر جابجایی Ordered [۲۲] ، Cycle، Partially Mapped و Reverse Ordered که برای کار با جایگشت ها مناسب می باشند، پیاده سازی شده اند که به توضیح مختصر آنها می پردازیم.

- **Ordered**: این روش جابجایی ترتیب نسبی ژنها در کروموزوم را تا حد ممکن حفظ می کند. با در نظر گرفتن جایگشت های والد اول و دوم، دو جایگشت فرزند به وسیله انتخاب دو نقطه برش در جایگشت های والدین و کپی کردن المان های بین دو نقطه برش در فرزندان، و پر کردن باقیمانده جایگشت های فرزندان با المان های استفاده نشده از والد دیگر با شروع از نقطه برش دوم به بعد، ایجاد می شوند.
- **Reverse Ordered**: این نوع عملگر جابجایی پیشنهاد شده شبیه روش Ordered است با این تفاوت که در این روش ژنهای خارج از دو نقطه برش در فرزندان کپی می شوند بر خلاف روش Ordered که ژنهای بین دو نقطه برش در فرزندان کپی می شوند.
- **Partially Mapped**: در این نوع عملگر جابجایی, دو جایگشت فرزند به وسیله انتخاب دو نقطه برش در جایگشت های والدین و کپی کردن المان های بین دو نقطه برش در فرزندان و پر کردن باقیمانده جایگشت های فرزندان با المان های متناظر آنها از والدین، ایجاد می شوند. توجه کنید که اگر یکی از المانهای خارج از ناحیه جابجایی فرزندی، با یکی از المان های ناحیه جابجایی خود یکسان باشد آن المان به عنوان حفره در

نظر گرفته می شود و باید مقدار آن عوض شود. نحوه پر کردن یک حفره به این صورت است که مثلا اگر حفره در فرزند اول باشد، ابتدا موقعیت ژنی از والد دوم را که دارای مقدار مساوی با مقدار حفره است پیدا می کنیم. سپس از والد اول مقدار ژنی را که در موقعیت یکسانی با ژن پیدا شده قرار دارد، به عنوان مقدار حفره در نظر می گیریم.

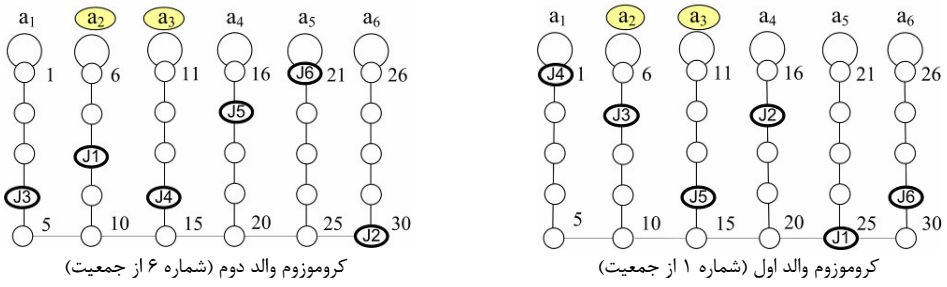
- **Cycle** : در این عملگر جابجایی، دو جایگشت فرزند بوسیله شکل دادن یک سیکل در بین والدین ایجاد می شوند. به عنوان مثال برای ایجاد فرزند اول، با شروع از اولین المان والد ۱، آن را به اولین ژن در والد دوم نگاشت می کنیم. سپس اولین ژن والد دوم را در فرزند اول پیدا کرده و به ژن هم مکان خود در والد دوم نگاشت می دهیم و این کار تا کامل شدن سیکل کامل ادامه می یابد. المان هایی از سیکل را که در والد ۱ هستند در فرزند اول قرار می دهیم. سپس مکان های خالی فرزند اول را با المان های متناظر در والد ۲ پر می کنیم. فرزند دوم نیز به همین ترتیب ایجاد می شود.

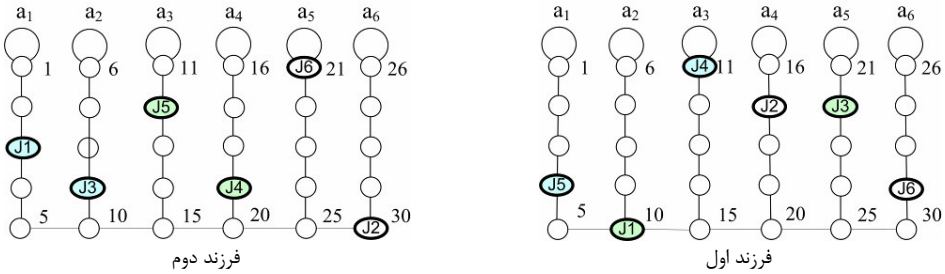
همچنین در این مقاله، دو عملگر جابجایی Exchange و Smart Exchange را نیز معرفی می کنیم که عملکرد بهتری در الگوریتم ترکیبی دارند. در عملگر جابجایی Smart Exchange که در شکل ۳ شبه کد آن نشان داده شده است، ، دو کروموزوم والد انتخاب می شوند و به صورت تصادفی دو ژن i و j در یکی از دو کروموزوم والد انتخاب می شوند. سپس همین دو ژن در کروموزوم دیگر انتخاب می گردند. مجموعه ژنهای با شماره های بین i و j را مجموعه جابجایی می نامیم. سپس ژن های هم شماره در دو مجموعه جابجایی با یکدیگر جابجا می شوند. با این عمل دو کروموزوم جدید حاصل می شوند که اصطلاحاً فرزندان دو آتاماتای والد خوانده می شوند. از آنجا که در این الگوریتم از n کروموزوم (آتاماتا) استفاده می‌شود و هر آتاماتا دارای مشخصه‌های اختصاصی مربوط به خود (وضعیت، اقدام و شیئ متناظر هر اقدام) می‌باشند، جهت خوانایی بیشتر شبه کد، این مشخصه ها را با پیشوند نام آتاماتا و جداساز نقطه نشان می دهیم. مثلا برای نشان دادن وضعیت کار روی اقدام u از آتاماتای i از نمایش Ai.Action(u).State استفاده شده است.

```
Procedure Crossover ( LA1, LA2 )
  Generate two random numbers r1 and r2 between 1 to n
  r1 = Random *n;   r2 = Random *n;
  r1 = Min (r1, r2);   r2 = Max (r1, r2);
  For i = r1 to r2 do
    If (TWT_i( LA1) < TWT_i( LA2) ) Then
      j = Action of LA2 Where
      LA2.Action (j).Object == LA1.Action (i).Object;
      Swap (LA2.Action (i).Object, LA2.Action (j).Object);
    Else
      j = Action of LA1 where
      LA1.Action ( j ) .Object == LA2.Action( i ).Object;
      Swap (LA1.Action (i).Object, LA1.Action (j).Object);
    End if   End For
End Procedure
//TWT_i(LA_k)=Total Weighted Tardiness of Action i in LA Number k
```

شکل ۳- شبه کد عملگر جابجایی Smart Exchange Crossover

عملگر جابجایی Exchange نیز مانند عملگر Smart Exchange است با این تفاوت که شرط If(TWTi(LA1)<TWTi(LA2)) در عملگر جابجایی Exchange وجود ندارد. در این شرط کارهای متعلق به اقدام های i ام از دو آتاماتای LA1 و LA2 با هم مقایسه می شوند که مشخص شود کدامیک از آنها با اجرا شدن در اقدام i ام، مجموع تاخیرات وزندار کمتری تولید می کند. در شکل ۴ مثالی از این عملگرها نشان داده شده است. به عنوان مثال دو آتاماتای LA1 و LA6 از جمعیت تشکیل شده قبل به صورت تصادفی انتخاب شده و با انتخاب تصادفی دو محل a2 وa3 مجموعه جابجایی $\{\alpha_2,\alpha_3\}$ حاصل می شود. و در نهایت با جابجایی کارهای موجود روی اقدام های متناظر در فاصله جابجایی، دو کروموزوم جدید حاصل می شود.





$$Crossover\,Set:\{a_2,a_3\}\Rightarrow\begin{cases}J1\leftrightarrow J3\\J4\leftrightarrow J5\end{cases}$$

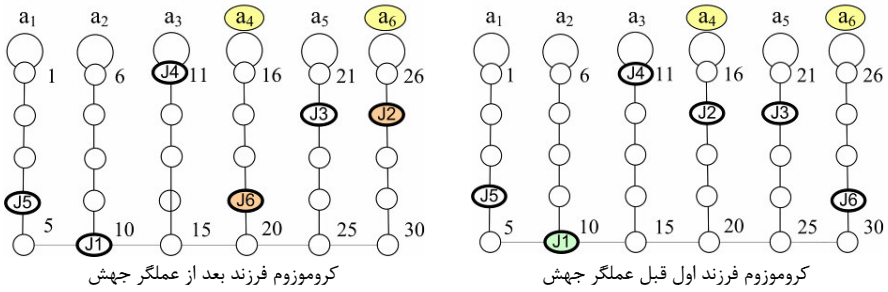
شکل ۴- نحوه انجام عملگر جابجایی Smart Exchange Crossover

ج) عملگر جهش: چهار عملگر جهش Swap، Insertion، Inversion و Scramble که برای کار با جایگشت ها مناسب می باشند، برای حل مسئله با الگوریتم ژنتیک معمولی و الگوریتم ترکیبی، پیاده سازی شده اند. برای مثال در عملگر جهش Swap، دو اقدام (ژن) از یک آتاماتا (کروموزوم) به صورت تصادفی انتخاب شده و کارهای متعلق به این اقدامها با یکدیگر جابجا می شوند. شکل های ۵ و ۶ شبه کد و مثالی از این عملگر را نشان می دهد. نحوه کار دیگر عملگرهای جهش بصورت زیر است.

- Inversion: دو ژن از کروموزوم را بطور تصادفی انتخاب کرده و ترتیب ژنها در بین این دو ژن را معکوس می کنیم.
- Insertion: یک ژن یا یک بلوک از ژنها را در کروموزوم بطور تصادفی انتخاب کرده و آنها را در یک نقطه تصادفی دیگر درج می کنیم و
- Scramble: یک بلوک از ژنها را بطور تصادفی انتخاب کرده و آنها را بصورت تصادفی دوباره مرتب می کنیم.

```
Procedure Mutation ( LA )
    i = Random *n;   j = Random *n;
    Swap( LA.Action( i ).Object , LA.Action( j ).Object );
End Procedure
```

شکل ۵- شبه کد عملگر جهش



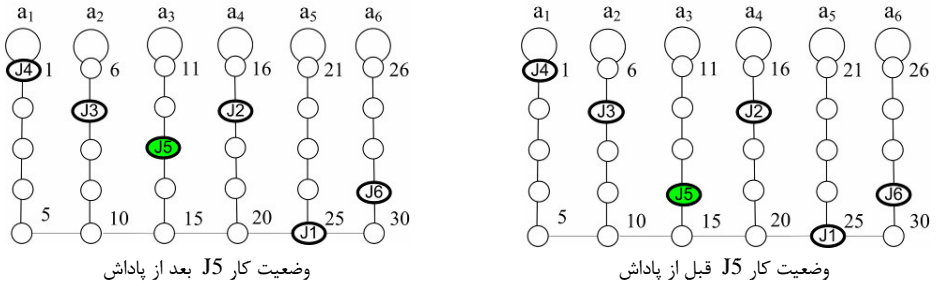
شکل ۶- نحوه اعمال عملگر جهش Swap

د) عملگر جریمه و پاداش[۲۰]: از آنجا که هر کروموزوم به صورت یک آتاماتای یادگیر نشان داده شده است در هر یک از آتاماتاها پس از بررسی میزان برازندگی کار متعلق به یک اقدام آتاماتا (ژن کروموزوم) که به صورت تصادفی انتخاب می شود، آن اقدام (ژن) پاداش یا جریمه می شود. در اثر پاداش دادن یا جریمه کردن یک اقدام، وضعیت کار متعلق به آن اقدام در مجموعه وضعیت های اقدام، تغییر می کند. اگر یک کار در وضعیت مرزی یک اقدام قرار داشته باشد، جریمه شدن اقدام آن باعث تغییر وضعیت آن کار از روی این اقدام به روی یک اقدام دیگر می شود و در نتیجه باعث ایجاد جایگشت جدیدی می گردد. نرخ این عملگر باید پایین باشد زیرا این عملگر، یک عملگر جستجوی تصادفی است و اگر با نرخ بالا اعمال شود باعث کاهش در کارایی الگوریتم می شود. عملگر جریمه و پاداش با توجه به نوع آتاماتای یادگیر متفاوت خواهد بود. شکل ۷ شبه کد عملگر پاداش اقدام u از آتاماتای LA با اتصالات مشابه آتاماتای ستلین را نشان می دهد.

```
Procedure Reward( LA, u )
    If (LA.Action(u).State-1) mod N <> 0 then
        Dec (LA.Action(u).State);
    End if
End Procedure
```

شکل ۷- عملگر پاداش اقدام u از آتاماتای LA با اتصالات آتاماتای ستلین

به عنوان مثال با توجه به شکل ۸ در آتاماتای با اتصالات مشابه آتاماتای ستلین اگر کار J_5 در مجموعه وضعیت های {۱۱,۱۲,۱۳,۱۴,۱۵} قرار داشته باشد و تاخیر وزندار حاصل از این کار در صورت تمام شدن فرجه اینکار از مقدارآستانه (مقدار آستانه در هر لحظه برابر است با کل تاخیرات وزندار حاصل از این زمانبندی تقسیم بر تعداد کارها) کوچکتر باشد به اقدام a_3 که این کار رویش قرار دارد، پاداش داده می شود یعنی کار روی اقدام مربوطه به سمت وضعیت های داخلی تر و بالاتر این اقدام حرکت می کند. اگر کار J_5 در وضعیت داخلی (وضعیت شماره ۱۱) قرار داشته باشد و اقدام مربوط به آن پاداش بگیرد در همان وضعیت باقی می ماند.



شکل ۸- نحوه پاداش دادن به کار J5 روی اقدام a3

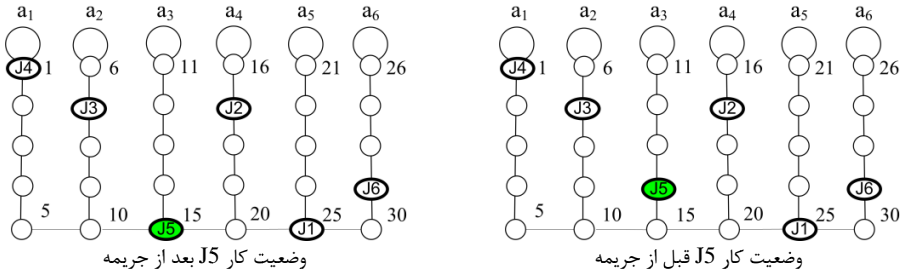
همچنین اگر تاخیر وزندار حاصل از کار موجود روی اقدام u از مجموعه کارها، در صورت تمام شدن فرجه اینکار از مقدارآستانه، بزرگتر باشد در این صورت زمانبندی در این قسمت مناسب نبوده و اقدام u جریمه می شود. شبه کد عملگر جریمه اقدام u از آتاماتای LA با اتصالات مشابه آتاماتای ستلین در شکل ۹ نشان داده شده است.

```
Procedure Penalize( LA , u )
    If (LA.Action(u).State) Mod N <> 0  then
        Inc(LA.Action(u).State);
    Else
        LowestWeightedTardiness= ∞ ;
        For U = 1 To n Do
            Create new Automata LA from LA by
            swapping  Objects on u and U;
            If TWT(Specified by LA') <
                LowestWeightedTardiness Then
                LowestWeightedTardiness=TWT(Specified by LA');
                BestJobAction = U;
            End If
        End For
        LA.Action(BestJobAction).State = LA.Action(BestJobAction).ActionNumber*N;
        Swap(LA.Action(u).State, LA.Action (BestJobAction).State);
    End Else
End Procedure
```

شکل ۹- شبه کد عملگر جریمه اقدام u از آتاماتای LA با اتصالات آتاماتای ستلین

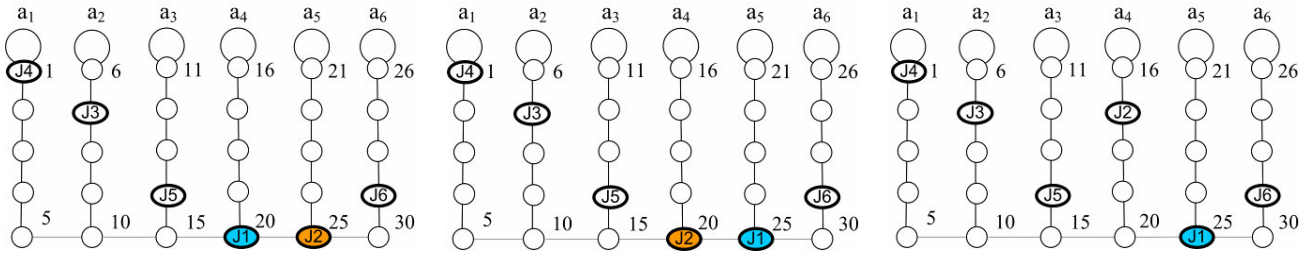
نحوه حرکت چنین کاری برای دو حالت مختلف در زیر آمده است.

الف) کار مورد نظر در وضعیتی غیر از وضعیت مرزی اقدام قرار داشته باشد: جریمه نمودن این اقدام سبب کم اهمیت و ناپایدار شدن آن می شود. برای مثال نحوه حرکت یک کار مانند J5 در چنین وضعیتی هنگام جریمه شدن اقدام مربوط به آن، در شکل ۱۰ نشان داده شده است.



شکل ۱۰- نحوه جریمه کردن کاری واقع در وضعیت غیر مرزی

ب) کار در وضعیت مرزی اقدام قرار داشته باشد: در این حالت کار دیگری از مجموعه کارهای مستقر روی دیگر اقدامها را پیدا می کنیم بطوریکه با ایجاد یک آتاماتای موقت و در نتیجه یک جایگشت جدید، اگر در این جایگشت جای کار فعلی و کار پیدا شده، عوض شوند بیشترین کاهش در مجموع تاخیرات وزندار بوجود آمده توسط آتاماتای جدید، حاصل گردد. در اینصورت اگر کار پیدا شده در وضعیت مرزی اقدام قرار داشته باشد جای دو کار عوض می شود و در غیر اینصورت ابتدا کار پیدا شده به وضعیت مرزی اقدام خود منتقل و سپس جابجایی صورت می پذیرد. نحوه حرکت چنین کاری در شکل ۱۱ نشان داده شده است.



شکل ۱۱- نحوه جریمه شدن یک کار موجود در وضعیت مرزی

در شکل ۱۲ نیز ساختار و حلقه اصلی الگوریتم ژنتیکی جدید برای حل مساله زمانبندی مجموع تاخیرات وزندار تک ماشینه ارائه شده است.

```
Function SMTWTSP_Solver(Set of Job Information):JobSchedulingSequence
    P = Size of Population;   Create the Initial Population LA1 … LAP;
    Copy Initial Population to OldPop;   OldPop.EvaluateFitness();
    While( Not Terminate Condition ) Do
        For i = 1 To P-1 Do
            Parent1=Select (OldPop); Parent2=Select(OldPop);
            If (Random < CrossoverRate) Then
                Crossover ( Parent1, Parent2 );//Crossover Operator Creates Child1 & Child2
            End If
            If (Random < MutationRate) Then
                Mutation ( Child1 ); Mutation ( Child2 );
                Child1.EvaluateFitness(); Child2.EvaluateFitness();
                NewPop.LAi =child1; NewPop.LAi+1=Child2;  i=i+2;
            End If
        End For
        For i = 1 To P Do
            u = Random(1 to Job Count) ;
            If ( TWTu( NewPop.LAi) < Threshold(NewPop.LAi )) Then
                Reward(NewPop.LAi , u );
            Else
                Penalize(NewPop.LAi , u );
            End If
        End For
        NewPop.EvaluateFitness();
        Copy NewPop to OldPop with Applying Elitism;
    End While
End Function
//Threshold(LAi)=(Total Weighted Tardiness of Specified Schedule by LAi ) / Job Count;
                //TWTu(NewPop.LAi)=Total Weighted Tardiness of Job in u Action inside i'th LA in the New population;
```

شکل ۱۲- شبه کد الگوریتم تکاملی جدید

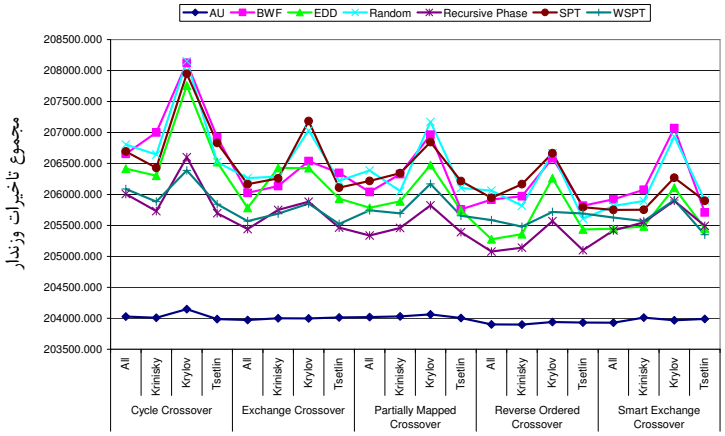
۶. نتایج آزمایشها

۱.۶. مسائل استاندارد مورد آزمایش

نمونه های آزمایشی مسئله از کتابخانه OR[۲۱] گرفته شده اند که توسط Beasley [۲۵] نگهداری می شود و توسط بسیاری از محققان مورد استفاده قرار می گیرند. بهترین راه حلهای شناخته شده نیز در کتابخانه OR در دسترس هستند که توسط یک الگوریتم جستجوی برنامه ریزی پویای تکراری [۱۸] حل شده اند.

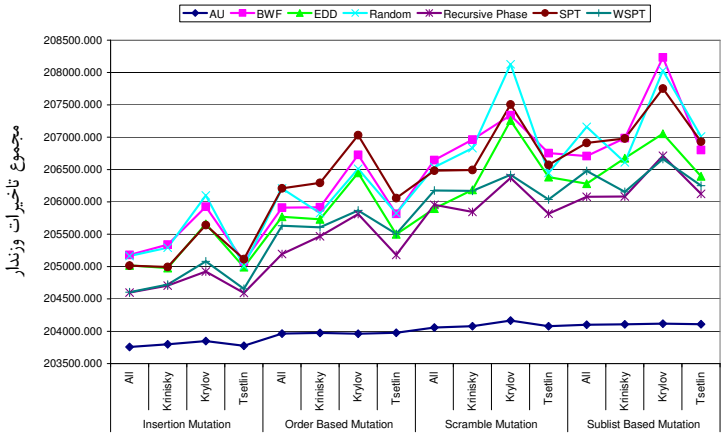
۲.۶. مقایسه نتایج الگوریتمهای حل مسئله SMTWTSP

قبل از مقایسه روشهای مختلف، عملیات تنظیم پارامترهای الگوریتم ترکیبی توسط ابزار حل مسئله زمانبندی با اجرای الگوریتم برای ۹ مسئله نمونه موجود در کتابخانه OR و سه مسئله تصادفی تولید شده، انجام گرفته است. در بررسی های انجام شده برای تنظیم پارامترهای الگوریتم ترکیبی، تعداد جمعیت الگوریتم ژنتیکی را برابر ۳۰ ، تعداد نسلهای فرایند تکامل را ۲۰۰ نسل و عمق آتاماتای یادگیر را ۵ در نظر گرفته ایم و از الگوریتم مقداردهی اولیه تصادفی، استفاده کرده ایم. در بررسی های انجام شده چنین استنباط شده است که عمگر انتخاب رتبه بندی عملکرد بهتری نسبت به چرخ رولت دارد. همچنین سه عملگر جابجایی Partially Mapped، Smart Exchange وReverse Ordered با نرخ بالای ۰.۸ نیز عملکرد بهتری نسبت به دیگر عملگرها، نشان می دهند. عملگر جهش Insertion همیشه بهتر از دیگر عملگرها بوده و ۰.۳ تا ۰.۵ نیز مناسب ترین نرخ اجرای این عملگر در کنار عملگر انتخاب مبتنی بر رتبه بندی می باشد. در اجرای الگوریتمها برای مقایسه آنها از این بهترین مقادیر استفاده شده است. شکل ۱۳ و ۱۴ نیز نشان می دهند که الگوریتم مقدار دهی اولیه AU در کنار عملگر جهش Insertion و عملگر جابجایی Smart Exchange و Reverse Ordered و اتصالات آتاماتای ستلین، کرینیسکی و ترکیب آنها(All)، بهترین روش مقدار دهی الگوریتم ترکیبی است و در آزمایشات نیز این پارامترها بکار رفته اند. عبارت ALL در این شکل حاکی از این مسئله است که انواع مختلفی از اتصالات آتاماتا در جمعیت ژنتیکی بصورت تصادفی بکار رفته است. شکل ۱۵ نیز میانگین مجموع تاخیرات وزندار بدست آمده برای بهترین کروموزوم جمعیت به ازای افزایش تعداد نسلها در طول فرآیند تکامل از ۲۰ نسل تا ۱۰۰۰ نسل برای مسئله شماره ۱۸ از مسائل استاندارد دارای ۵۰ کار را نشان می دهد. همچنان که دیده می شود تا نسل ۶۴۰ ام، بهبود محسوسی در نتایج وجود داشته است و پس از آن با نزدیک شدن به مقدار بهینه میزان بهبود کاهش یافته است. شکل ۱۶، مقایسه نتایج بدست آمده حاصل از اجرای الگوریتم ژنتیکی یادگیر برای تعداد مختلف افراد جمعیت از ۵ تا ۱۰۰ کروموزوم را نشان می دهد. شکل ۱۷ حاکی از مقدار مجموع تاخیرات بدست آمده از اجرای الگوریتم ژنتیک یادگیر برای عمقهای مختلف اقدامها برای آتاماتای یادگیر مهاجرت اشیا بکار رفته بعنوان کروموزومهای جمعیت از عمق ۲ تا ۴۰ حالت برای هر اقدام می باشد. همچنان که دیده می شود افزایش یا کاهش عمق تاثیر محسوسی روی جوابها ندارد و قانون خاصی را نمی توان در نظر گرفت. شکل ۱۸، زمان لازم برای اجرای الگوریتمهای مختلف جهت حل کردن سه مسئله زمانبندی دارای ۴۰ کار ، ۵۰ کار و ۱۰۰ کار را بر حسب میلی ثانیه نشان می دهد. مقدار موعد مقرر نسبی کارها (RDD) در هر سه مسئله ۰.۴ و فاکتور تاخیر (TF) آنها ۰.۸ می باشد. همچنان که دیده می شود، زمان اجرای الگوریتم ترکیبی نسبت به دیگر الگوریتمها بدلیل پیچیدگی ساختار آن بیشتر است اما کیفیت نتایج بدست آمده بسیار بهتر از دیگر الگوریتمهاست. در جدول های ۲، ۳ و ۴ درصد خطای[۲۲] (PRD) نتایج حاصل از مقایسه الگوریتمهای مختلف با مقادیر بهینه برای مسائل دارای ۴۰، ۵۰ و ۱۰۰ کار آمده اند. نحوه بدست آوردن درصد خطا بصورت PRD=100×(TWT-OPT)/OPT است که TWT مجموع تاخیرات وزندار بدست آمده از اجرای هر الگوریتم و OPT مقدار بهینه موجود برای حل مسائل با شرایط گفته شده می باشد. در هر جدول، نتایج اجرای الگوریتمها برای مقادیر موعد مقرر و فاکتور تاخیر متفاوت، خلاصه شده اند. برای تمام الگوریتم ها تعداد تکرار ۵۰۰ در نظر گرفته شده و عمق آتاماتای بکار رفته در الگوریتم جدید ۵ و تعداد اعضای جمعیت ۵۰ بوده است. همچنین مقایسه میانگین کل نتایج حاصل از اجرای الگوریتم ترکیبی روی تمام مسائل موجود برای اتصالات مختلف آتاماتا نشان می دهد که زمانی که انواع مختلفی از آتاماتا با اتصالات مختلف و با قابلیتهای یادگیری مختلف در جمعیت حضور داشته باشند، میانگین کیفیت راه حلهای بدست آمده بهبود می یابد.



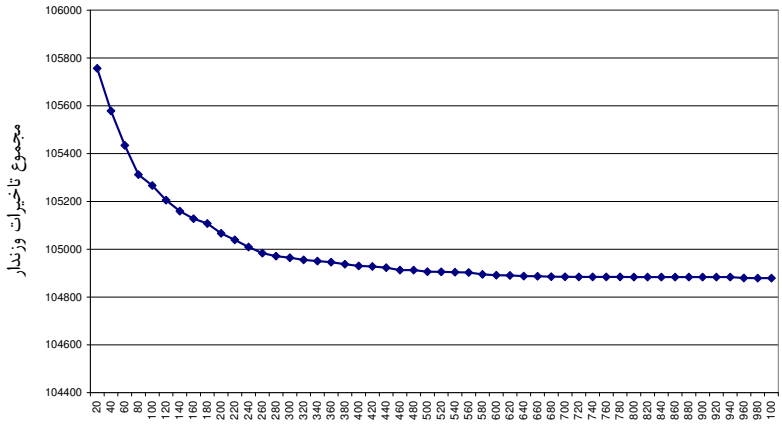
نوع عملگر جابجایی و نوع اتصالات آتاماتا

شکل۱۳-تعامل عملگرهای جابجایی و اتصالات آتاماتا و الگوریتمهای مقدار دهی متفاوت



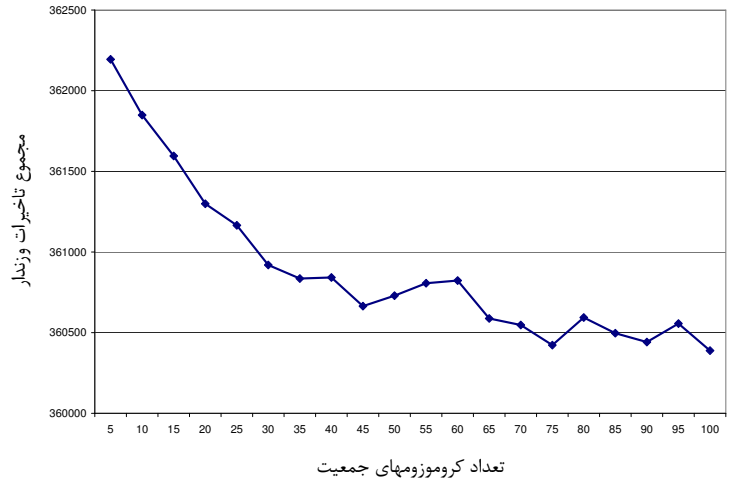
نوع اتصالات آتاماتا و نوع عملگر مقداردهی اولیه

شکل۱۴-تعامل عملگرهای جهش و اتصالات آتاماتا و الگوریتمهای مقدار دهی متفاوت

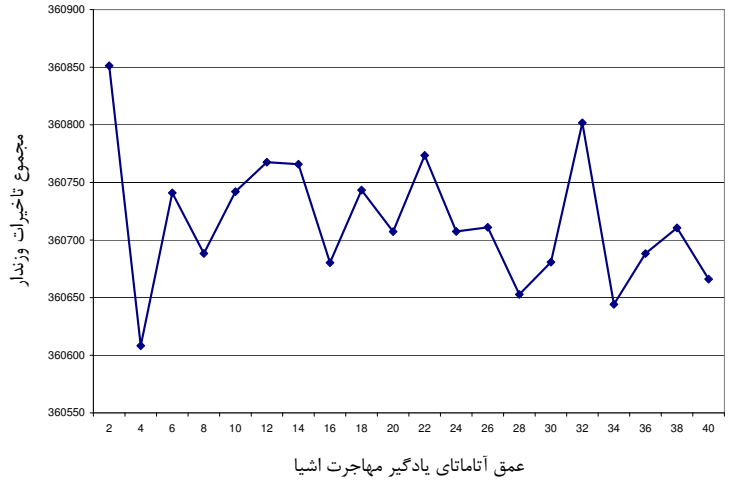


تعداد نسلهای فرایند تکامل

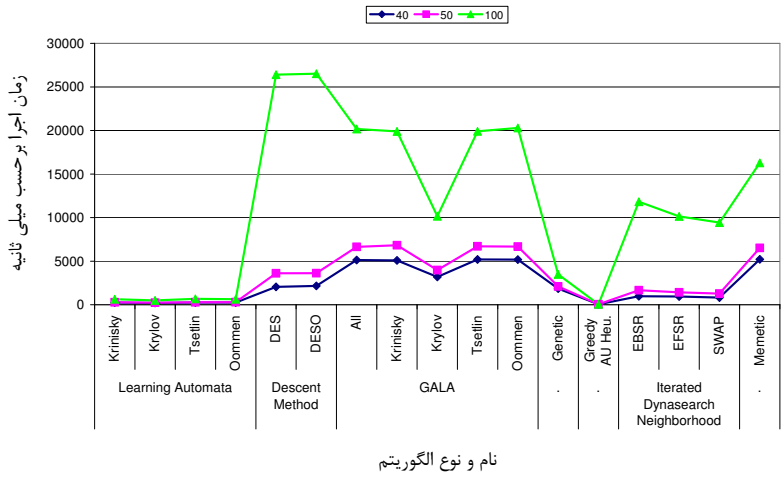
شکل۱۵- مقایسه مجموع تاخیرات وزندار بهترین کروموزوم جمعیت بعد از تعداد نسلها



شکل۱۶- مقایسه مجموع تاخیرات وزندار برای تعداد مختلف کروموزوم ها در جمعیت



شکل۱۷-مقایسه مجموع تاخیرات وزندار برای عمقهای مختلف آتاماتا



شکل ۱۸- مقایسه زمان اجرای الگوریتمهای مختلف برای مسائل دارای ۴۰، ۵۰ و ۱۰۰ کار

جدول۲-مقایسه درصد خطای نتایج حاصل از اجرای الگوریتمهای مختلف برای مسائل ۴۰ کاره

Memetic Algorithm	Iterative Dynasearch Neighborhood	Greedy Algorithm	Genetic Algorithm	GA+LA	Descent Method	Learning Automata	TF	RDD
۲.۴۳	۴.۱	۲۰.۶	۲.۷۳	۱.۱۳	۲.۴۶	۴.۸۱	۰.۲	۰.۲
۱.۳۱	۵.۸۵	۱۴.۸	۰.۴۱	۰	۰.۴۱	۲.۴۹	۰.۴	
۰.۴۴	۳	۸.۸۱	۰.۲۴	۰.۰۵	۰.۱۹	۱.۷	۰.۶	
۰.۲۹	۰.۳	۱.۰۴	۰.۲۴	۰.۰۸	۰.۱۵	۰.۶۸	۰.۸	
۰.۰۱	۰.۰۳	۰.۲۲	۰.۰۱	۰	۰.۰۱	۰.۲۲	۱.۰	
۹۹.۹	۴۳.۹	۱۹۳	۸.۷۴	۰.۲۳	۱۱.۶	۴۶.۸	۰.۲	۰.۴
۳.۲۷	۱۴.۶	۳۹.۸	۱.۴۶	۰.۰۵	۰.۰۸	۴.۸۵	۰.۴	
۱.۳۶	۱.۹۶	۶.۲۹	۱.۲۴	۰.۰۷	۰.۵۶	۲.۳۷	۰.۶	
۰.۱۶	۰.۵۳	۱.۷۸	۰.۷۳	۰.۰۶	۰.۱	۱.۰۵	۰.۸	
۰.۰۳	۰.۰۳	۰.۲۷	۰.۰۲	۰.۰۱	۰.۰۱	۰.۲۷	۱.۰	
۰	۰	۰	۰	۰	۰	۰	۰.۲	۰.۶
۸.۲۷	۱۲.۵	۴۱.۹	۳.۹	۰.۵۱	۲.۵۹	۸.۴۶	۰.۴	
۲.۸۸	۳.۰۹	۱۰.۱	۱.۱۵	۰.۱۲	۰.۸۵	۳.۱	۰.۶	
۰.۲	۰.۱۹	۱.۴۴	۰.۱۳	۰.۰۱	۰.۱۲	۰.۸۲	۰.۸	
۰.۰۲	۰.۰۸	۰.۶۸	۰.۰۸	۰.۰۲	۰.۰۱	۰.۵۵	۱.۰	
۰	۰	۰	۰	۰	۰	۰	۰.۲	۰.۸
۳۴.۶	۴.۴۳	۸۶.۳	۱.۳۹	۰.۸۸	۱.۵۳	۱۳.۷	۰.۴	
۲.۲۲	۱.۳۴	۷.۷	۰.۷۸	۰.۰۴	۰.۴۹	۳.۱۸	۰.۶	
۰.۳۷	۰.۲۲	۲.۰۸	۰.۰۸	۰.۰۱	۰.۱۲	۱.۳۷	۰.۸	
۰.۰۱	۰.۰۱	۰.۳۵	۰.۰۱	۰.۰۱	۰.۰۱	۰.۲۴	۱.۰	
۰	۰	۰	۰	۰	۰	۰	۰.۲	۱.۰
۱۴.۵	۴.۰۵	۱۸.۹	۳.۰۸	۱.۰۶	۱.۵۹	۴۰	۰.۴	
۲.۴۱	۱.۰۷	۵.۷۶	۰.۳۳	۰.۰۶	۰.۵	۳.۱۴	۰.۶	
۰.۳۷	۰.۵۵	۲.۲۹	۰.۳۶	۰.۰۴	۰.۲۶	۱.۲	۰.۸	
۰.۱	۰.۱۴	۰.۶۹	۰.۱۴	۰.۰۱	۰.۰۸	۰.۵۱	۱.۰	
۷	۴.۰۸	۱۸.۶	۱.۰۹	۰.۱۸	۰.۹۵	۵.۶۶	میانگین	

جدول۳-مقایسه درصد خطای نتایج حاصل از اجرای الگوریتمهای مختلف برای مسائل ۵۰ کاره

Memetic Algorithm	Iterative Dynasearch Neighborhood	Greedy Algorithm	Genetic Algorithm	GA+LA	Descent Method	Learning Automata	TF	RDD
۱.۷۸	۵.۵۲	۱۵.۲	۲.۳۱	۱.۰۸	۱.۲۱	۲.۳۸	۰.۲	۰.۲
۱.۲۳	۵.۷۲	۱۶.۴	۱.۴۳	۰.۰۸	۰.۲۹	۳.۱۹	۰.۴	
۰.۵۳	۵.۲۷	۱۴.۵	۰.۹۹	۰.۰۶	۰.۱۳	۱.۷۷	۰.۶	
۰.۲۶	۰.۸۸	۲.۳۳	۰.۲۸	۰.۰۳	۰.۲۲	۰.۷۵	۰.۸	
۰	۰.۰۳	۰.۲۱	۰.۰۲	۰.۰۱	۰.۰۱	۰.۲۱	۱.۰	
۱۰۷	۳۷	۱۶۹	۲۱.۸	۰.۱۵	۱.۲۸	۱۷.۷	۰.۲	۰.۴
۳.۲۳	۷.۸۱	۲۰.۸	۴.۳	۰.۲۶	۲.۰۱	۴.۵۹	۰.۴	
۰.۶۹	۴.۷۶	۱۲.۴	۱.۶۲	۰.۰۴	۰.۹۶	۲.۹۳	۰.۶	
۰.۲۸	۰.۱۳	۱.۰۷	۰.۱۴	۰.۰۶	۰.۱۴	۰.۹۴	۰.۸	
۰.۰۱	۰.۰۲	۰.۱۷	۰.۰۱	۰.۰۱	۰.۰۲	۰.۱۷	۱.۰	
۰	۰	۰	۰	۰	۰	۰	۰.۲	۰.۶
۷.۳۳	۱۴.۹	۳۵	۶.۱۴	۰.۲	۱.۷	۹.۲۱	۰.۴	
۴.۹۵	۳.۱۳	۶.۹۷	۳.۱۳	۰.۱۴	۱.۱	۲.۹۲	۰.۶	
۰.۳۵	۰.۳۸	۱.۲۱	۰.۳۹	۰.۰۷	۰.۳۵	۱	۰.۸	
۰.۰۸	۰.۰۶	۰.۵۹	۰.۰۴	۰.۰۳	۰.۰۳	۰.۳۵	۱.۰	
۰	۰	۰	۰	۰	۰	۰	۰.۲	۰.۸
۱۷	۹.۰۲	۴۲.۹	۶.۲۳	۰.۰۵	۵.۳۲	۱۲.۵	۰.۴	
۲.۱۸	۳.۳۷	۸.۰۷	۳.۰۲	۰.۳	۰.۹۶	۳.۴۸	۰.۶	
۰.۵۹	۰.۳۸	۲.۱۷	۰.۳۳	۰.۰۷	۰.۳۵	۱.۲۵	۰.۸	
۰.۰۴	۰.۱	۰.۵۱	۰.۰۶	۰.۰۱	۰.۰۷	۰.۳۱	۱.۰	
۰	۰	۰	۰	۰	۰	۰	۰.۲	۱.۰
۱۰.۳	۲.۰۹	۱۵.۷	۰.۶۸	۰.۰۵	۱.۳۱	۱.۸۴	۰.۴	
۳.۱۱	۲.۷۱	۷.۹۱	۱.۴	۰.۲	۲.۸۵	۴.۹۹	۰.۶	
۰.۶۲	۰.۱۵	۱.۴۲	۰.۱۳	۰.۰۱	۰.۱۱	۰.۶۸	۰.۸	
۰.۱۹	۰.۰۹	۰.۷۵	۰.۱	۰.۰۱	۰.۰۶	۰.۴۱	۱.۰	
۶.۴۷	۴.۱۴	۱۵	۲.۱۸	۰.۱۲	۰.۸۲	۲.۹۴	میانگین	

جدول۴-مقایسه درصد خطای نتایج حاصل از اجرای الگوریتمهای مختلف برای مسائل ۱۰۰ کاره

Memetic Algorithm	Iterative Dynasearch Neighborhood	Greedy Algorithm	Genetic Algorithm	GA+LA	Descent Method	Learning Automata	TF	RDD
۱.۴۵	۱۲.۹	۳۳.۹	۲.۷۶	۰.۰۶	۱.۱۵	۳.۶۲	۰.۲	۰.۲
۰.۵۵	۱۷	۴۷.۸	۰.۸۲	۰.۰۶	۰.۸	۴.۸۹	۰.۴	
۰.۴۴	۸.۱۸	۲۳.۴	۰.۴	۰.۰۹	۰.۲۱	۱.۶۷	۰.۶	
۰.۲	۰.۷۹	۱.۶۶	۰.۴۹	۰.۱۳	۰.۱۴	۰.۸۷	۰.۸	
۰.۰۱	۰.۰۲	۰.۱۶	۰.۰۳	۰.۰۲	۰.۰۱	۰.۱۶	۱.۰	
۳۸.۱	۴۰.۱	۱۷۴	۶۳.۵	۱۸	۵۵.۴	۱۱۳	۰.۲	۰.۴
۱.۸۸	۱۶.۵	۴۵.۱	۴.۲۵	۰.۲۳	۱.۶۳	۵.۷۲	۰.۴	
۱.۱۳	۴.۵۷	۱۱.۶	۱.۲۴	۰.۲۳	۰.۱۸	۲.۵۴	۰.۶	
۰.۲۹	۰.۱۷	۰.۸۴	۰.۲۸	۰.۱	۰.۰۳	۰.۶۸	۰.۸	
۰.۰۲	۰.۰۴	۰.۱۸	۰.۰۴	۰.۰۲	۰.۰۱	۰.۱۵	۱.۰	
۰	۰	۰	۰	۰	۰	۰	۰.۲	۰.۶
۳.۶۹	۱۶.۷	۴۴.۷	۷.۸۷	۰.۵۱	۲.۰۴	۵.۴۲	۰.۴	
۱.۷۲	۶.۸۹	۱۶.۶	۱.۵۶	۰.۲۷	۱.۰۷	۴.۵۹	۰.۶	
۰.۲۳	۰.۱۶	۰.۸۶	۰.۱۹	۰.۰۹	۰.۱۲	۰.۵۸	۰.۸	
۰.۰۱	۰.۰۲	۰.۱۷	۰.۰۳	۰.۰۲	۰.۰۱	۰.۱۱	۱.۰	
۰	۰	۰	۰	۰	۰	۰	۰.۲	۰.۸
۹۷.۵	۵.۶	۱۱۴	۷.۶۶	۳.۶۴	۵.۵۶	۴۱.۱	۰.۴	
۲.۸۳	۳.۸	۱۱	۲.۰۵	۰.۳۹	۰.۵۵	۳.۳۳	۰.۶	
۰.۳۲	۰.۱۸	۰.۸۹	۰.۲۵	۰.۰۷	۰.۱۴	۰.۵۸	۰.۸	
۰.۰۵	۰.۰۴	۰.۱۸	۰.۰۴	۰.۰۲	۰.۰۳	۰.۱۲	۱.۰	
۰	۰	۰	۰	۰	۰	۰	۰.۲	۱.۰
۶۸.۲	۰.۹۱	۱۱.۲	۰.۷۳	۰.۱۸	۰.۲۳	۴.۱۱	۰.۴	
۱.۱۳	۰.۶۱	۲.۶۷	۰.۶۵	۰.۱۸	۰.۲۸	۱.۲۹	۰.۶	
۰.۲۹	۰.۱۹	۰.۹۹	۰.۲۲	۰.۰۵	۰.۱۱	۰.۵۱	۰.۸	
۰.۰۸	۰.۰۹	۰.۳۸	۰.۱	۰.۰۳	۰.۰۶	۰.۲۱	۱.۰	
۸.۸۱	۵.۴۲	۲۱.۷	۳.۸	۰.۹۸	۲.۷۹	۷.۸۱	میانگین	

۷. نتیجه گیری و پیشنهاد

در این مقاله، حل مساله زمانبندی مجموع تاخیرات وزندار تک ماشینه توسط الگوریتمهای تصادفی مختلف و یک الگوریتم ترکیبی جدید مورد بررسی قرار گرفت. الگوریتم ترکیبی از دو روش الگوریتم های ژنتیکی و آتاماتاهای یادگیر بطور همزمان برای جستجو در فضای حالات جواب مسئله، استفاده می نماید. در این مقاله، نشان داده شد که با استفاده همزمان آتاماتای یادگیر و الگوریتم ژنتیکی در فرایند جستجو، کیفیت نتایج حل مسئله مطرح شده، افزایش مناسبی پیدا می نماید. نتایج آزمایش ها، برتری نتایج حاصل از روش ترکیبی را نسبت به روش‌های دیگر نشان می دهد. هدف ما در کارهای آینده، استفاده از الگوریتم جدید برای دیگر انواع مسائل زمانبندی و پیشنهاد الگوریتم مشابهی با ساختاری ساده تر و زمان اجرای کمتر خواهد بود.

۸. مراجع

[1] Barbaros, T.C., Kara, B.Y., Sabuncuoglu, I., "*An efficient algorithm for the single machine total tardiness problem*", IIE Transaction, 33, pp.661–674, 2001.

[2] Jouglet, A., Baptiste, P., and Carlier, J., "*Exact Procedure for Single Machine Total Cost Scheduling*", IEEE International Conference on Systems, Man and Cybernetics, Oct 6-9, 2002.

[3] Potts, C.N., and Wassenhove, L.N.V., "*A branch and bound algorithm for the total weighted tardiness problems*", Operations Research, 33(2), pp. 363-377, March-April 1985.

[4] Schrage, L., and Baker, K. R., "*Dynamic programming solution of sequencing problem with precedence constraints*", Operations Research, 26, pp. 444-449, 1978.

[5] Crauwels, H. A. J., Potts, C. N., and Wassenhove L. N.V., "*Local search heuristics for the single machine total weighted tardiness scheduling problem*", INFORMS Journal on Computing, 10(3), pp. 341-350, summer 1998.

[6] Akturk, M.S., "*A new dominance rule for the total weighted tardiness problem*", Production Planning and Control, 10(2), pp. 138-149, 1999.

[7] Huegler, P.A. and Vasko, F.J., "*A performance comparison of heuristics for the total weighted tardiness problem*", Computers & Industrial Engineering, 32(4), pp. 753-767, 1997.

[8] Maheswaran, R., and Ponnambalam, S.G., "*An Investigation on Single Machine Total Weighted Tardiness Scheduling Problems*", International Journal Of Advanced Manufacturing Technology, 22(3-4), pp. 243-248, 2003.

[9] Grosso, A., Croce, F.D, and Tadei, A., "*An Enhanced Dynasearch Neighborhood for the Single Machine Total Weighted Tardiness Scheduling Problem*", Operations Research Letters, 32(1), pp. 68-72, 2004.
[10] Avci S., Akturk, M.S., and Storer, R.H., "*A Problem Space Algorithm for Single Machine Weighted Tardiness Problems*", IIE Transactions, 35, pp. 479-486, 2003.
[11] Madureira, A., Ramos, C., and Silva, S.D.C., "*A GA based scheduling system for dynamic single machine problem*", Proceedings of the forth IEEE International Symposium on Assembly and Task Planning, Soft Research Park, Fukuoka, Japan, pp. 262-267, May 2001.
[12] Liu, N., Abdelrahman, M.A., Ramaswamy, S., "*A genetic algorithm for the single machine total weighted tardiness problem*", Proceedings of the 35th Southeastern Symposium on System Theory, pp. 34 – 38,16-18 March 2003.
[13] Liu N., Abdelrahman, M., Ramaswamy, S., "*A Genetic Algorithm for Single Machine Total Weighted Tardiness Scheduling Problem*", International Journal of Intelligent Control and Systems, 10(3), pp. 218-225, Sept 2005.
[14] Ferrolho, A., Crisostomo, M., "*Single Machine Total Weighted Tardiness Problem with Genetic Algorithms*", 2007 IEEE/ACS International Conference on Computer Systems and Applications, pp. 1-8, 2007.
[15] Franca, P.M., Mendes, A., Moscato, P., "*A memetic algorithm for the total tardiness single machine scheduling problem*", Eur J Operations Research, 132(1), pp 224–242, 2001.
[16] Maheswaran, R., Ponnambalam, S.G., Aravindan, C.,"*A meta-heuristic approach to single machine scheduling problems*", International Journal Of Advanced Manufacturing Technology, 25, pp. 772–776, Springer-Verlag London, 2005.
[17] Maheswaran, R., Ponnambalam, S.G., "*An intensive search evolutionary algorithm for single-machine total-weighted-tardiness scheduling problems*", International Journal of Advanced Manufacturing Technology, 26, pp. 1150–1156, Springer-Verlag London, 2005.
[18] Congram, R.K., Potts, C.N., Van de Velde, S., "*An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem*", INFORMS Journal on Computing, 14, pp 52–67, 2002.
[19] RezapourSaleh, M. and Meybodi, M. R., "*A Hybrid Algorithm for Solving Graph Isomorphism Problem*", Proceedings of the Second International Conference on Information and Knowledge Technology (IKT2005), Tehran, Iran, May 24-26, 2005.
[20] Asghari, K., Safari Mamaghani, A. and Meybodi, M. R., "*An Evolutionary Approach for Query Optimization Problem in Database*", Proceedings of International Joint Conferences on Computers, Information and System Sciences, and Engineering (CISSE2007), University of Bridgeport, England, December 2007.
[21] Lenstra, J. K., Rinnooy Kan, A. H. G., and Brucker, P., "*Complexity of machine scheduling problems*", Annals of Discrete Mathematics, 1, pp. 343-362, 1977.
[22] Goldberg, D.E., "*Genetic Algorithms in Search, Optimization and Machine Learning*", Addison Wesley, Reading MA, 1989.
[23] Narendra, K.S. and Thathachar, M.A.L., *"Learning Automata: An Introduction",* Prentice-hall, Englewood cliffs, 1989.
[24] Oommen, B. J., and Ma, D. C. Y., "*Deterministic Learning Automata Solution to the Keyboard Optimization Problem*", IEEE Transaction on Computers, Vol. 37, No. 1 , 2-3,1988.
[25] Beasley JE (1990) O.R. library. http://people.brunel.ac.uk /~mastjjb/jeb/orlib/wtinfo.html

زیر نویس ها

[1] Single Machine Total Weighted Tardiness Scheduling Problem
[2] Heuristic Dispatching Rules
[3] Local Search Heuristics
[4] Relative Due Date
[5] Tardiness Factor
[6] Dynasearch
[7] Generalized Pairwise Interchanges
[8] Earliest Due Date
[9] Weighted Shortest Processing Time
[10] Shortest Processing Time
[11] Biggest Weight First
[12] Apparent Urgency
[13] Adjacent Pairwise Interchange
[14] NonAdjacent pairwise Interchange
[15] Extraction and Backward-Shifted Reinsertion
[16] Extraction and Forward-Shifted Reinsertion
[17] Object Migrating Automata (OMA)
[18] Oommen
[19] Ma
[20] Penalty and Reward
[21] Operation Research Library
[22] Percentage Relative Deviation