

## بهینه‌سازی زمان بندی الگوریتم‌های موازی با استفاده از الگوریتم ژنتیک

خدیجه نعمتی<sup>۱</sup>، امیر حسین رفاهی شیخانی<sup>۲\*</sup>، سهراب کردرستمی<sup>۳</sup>

۱- دانشجوی کارشناسی ارشد، دانشگاه آزاد اسلامی، واحد لاهیجان، دانشکده علوم ریاضی، گروه علوم کامپیوتر، لاهیجان، ایران

۲- استادیار، دانشگاه آزاد اسلامی، واحد لاهیجان، دانشکده علوم ریاضی، گروه علوم کامپیوتر، لاهیجان، ایران

۳- استاد، دانشگاه آزاد اسلامی، واحد لاهیجان، دانشکده علوم ریاضی، گروه علوم کامپیوتر، لاهیجان، ایران

رسید مقاله: ۹ دی ۱۳۹۴

پذیرش مقاله: ۷ خرداد ۱۳۹۵

### چکیده

زمان بندی<sup>۱</sup> مجموعه‌ای از ماشین‌های موازی که در یک محیط هستند، هم از نظر تئوری و هم از نظر کاربردی مهم است. از نظر تئوری، تعمیم مساله‌ی زمان بندی یک ماشین است و از نظر کاربردی صحت منابع موازی در جهان واقعی می‌باشد. وقتی ماشین‌ها، کامپیوتر باشند یک برنامه‌ی موازی نیاز است زیرا اعضای مجموعه به طور موازی اجرا می‌شوند و این اجرا بر اساس ارتباطات تقدمی<sup>۲</sup> آن‌ها است. مزیت اجرای زمان بندی وظایف، قدرت محاسباتی کامل را فراهم می‌کند که به وسیله‌ی سیستم چندپردازنده یا چند کامپیوتری به دست می‌آید. در این مقاله نشان می‌دهیم مساله‌ی تخصیص تعدادی وظایف ناهمسان در سیستم‌های چندپردازنده یا چند کامپیوتری چگونه است. مدل فرضی سیستم شامل تعدادی پردازنده‌ی همسان است و در یک زمان، فقط یک وظیفه، روی یک پردازنده اجرا می‌شود و نیز همه‌ی زمان بندی‌ها و وظایف، غیرانحصاری<sup>۳</sup> هستند.

**کلمات کلیدی:** الگوریتم ژنتیک، الگوریتم‌های موازی، گراف وظیفه، زمان بندی وظایف.

### ۱ مقدمه

سیستم‌های پردازش موازی امروزه کاربردهای زیادی در زمینه‌ی پردازش اطلاعات، مدل سازی آب و هوا، سیستم‌های پایگاه داده، شبیه سازی بلادرنگ سیستم‌های پویا و... دارند. بیشترین کارایی از این سیستم‌ها زمانی به دست می‌آید که وظایف بین پردازنده‌ها به صورت کارا تقسیم بندی شود [۱]. کارهای محاسباتی پیچیده نمی‌تواند در بازه زمانی قابل قبول بر روی یک پردازنده اجرا شود و به همین دلیل باید به زیرکارهای کوچک تر تقسیم گردد و با زمان بندی مناسب و اختصاص دادن آن‌ها به یک سیستم چندپردازنده، زمان انجام کلیه‌ی زیرکارها

\*عده‌دار مکاتبات

آدرس الکترونیکی: Ah\_refahi@liau.ac.ir

<sup>1</sup> scheduling

<sup>2</sup> precedence relationship

<sup>3</sup> non-preemptive

کاهش یابد یا به عبارتی زمان اجرای آخرین کار مینیمم شود [۲]. زمان بندی کارها به طور مناسب از اهمیت بالایی برخوردار است. چون زمان بندی نامناسب<sup>۱</sup> می‌تواند منجر به شکست پتانسیل یک سیستم موازی شود و یا به علت سربار ارتباطی بیش از اندازه<sup>۲</sup> یا منابع تحت بهره‌برداری<sup>۳</sup> مزایای موازی سازی را خنثی کند [۳].

راه حل‌های کلاسیک برای این مساله [۴-۶] از ترکیب روش‌های جستجو و اکتشافی (یا ابتکاری) استفاده می‌کند در حالی که این روش‌ها اغلب راه حل‌های مناسب تولید می‌کنند، پس نتایج زمان بندی معمولاً زیربینه<sup>۴</sup> است. این روش‌ها اغلب به علت فقدان ضمانت کارایی و مقیاس پذیری مورد انتقاد قرار می‌گیرند. در مقابل، الگوریتم ژنتیک [۷ و ۸] راه حل‌های اتفاقی<sup>۵</sup> برای مساله‌های زمان بندی بزرگ فراهم می‌کند.

این مقاله پیاده سازی یک الگوریتم ژنتیک را برای مینیم کردن طول زمان بندی‌های گراف وظیفه‌ای که باید روی یک سیستم چندپردازنده اجرا شود توصیف می‌کند. اهمیت مساله‌ی زمان بندی روی سیستم چندپردازنده ما را به مطالعه‌ی مقایسه‌ای<sup>۶</sup> رهنمون می‌کند. راه حل‌های زیادی برای مساله‌ی زمان بندی کارها روی سیستم چندپردازنده ارایه شده است [۳].

الگوریتم ژنتیک هولاند در زمان بندی [۷] به صورت یک استراتژی تکاملی<sup>۷</sup> ارایه شده است که کشف و جستجوی سریع فضای زمان بندی را مجاز دانسته و راه حل‌های خوبی را به سرعت پیدا می‌کند و آن‌ها را برای ایجاد زمان بندی در مسایل عمومی اعمال می‌نماید [۸].

کیم و همکاران [۹] یک مساله‌ی زمان بندی قطعی را که در آن وظایف چند گانه با تعداد  $k$  رابطه تقدمی، که روی ماشین‌های موازی چند گانه برابر، پردازش می‌شود در نظر می‌گیرد. هدف مینیم کردن زمان اجراست. ارتباط تقدمی بین دو وظیفه  $i$  و  $j$  موقعیت وظیفه  $i$  را نشان می‌دهد که تا زمانی که  $i$  پردازش نشود از پردازش شدن آن جلوگیری می‌کند. این مورد با تعریف‌های استاندارد روابط تقدمی، متفاوت است به طوری که زمانی می‌تواند شروع شود تا زمانی که  $i$  تکمیل گردد.

وو و همکاران [۱۰] یک الگوریتم ژنتیک جدید پیشنهاد داده است که هم افراد معتبر و هم نامعتبر را در جمعیت می‌پذیرد. این GA یک تابع ارزیابی شایستگی<sup>۸</sup> را به تدریج<sup>۹</sup> افزایش می‌دهد و مقادیر حاصل از تابع را تا زمانی که یک راه حل رضایت بخش<sup>۱۰</sup> پیدا شود بررسی می‌کند. این روش قابل قیاس با مساله‌های بزرگ نیست چون زمان زیادی صرف ارزیابی افراد نامعتبر می‌شود که شاید معتبر باشند.

مساله تخصیص وظیفه توسط بسیاری از محققان [۵، ۱۱-۱۷] بررسی شده است. چندین روش اکتشافی نیز پیشنهاد شده است مانند روش اکتشاف بر مبنای کوتاه‌ترین مسیر<sup>۱۱</sup>، دوبخشی بازگشتی متعامد<sup>۱۲</sup>، الگوریتم ژنتیک

<sup>1</sup> inappropriate

<sup>2</sup> excessive

<sup>3</sup> under-utilization

<sup>4</sup> suboptimal

<sup>5</sup> stochastic solution

<sup>6</sup> comparative studies

<sup>7</sup> evolutionary

<sup>8</sup> incremental fitness function

<sup>9</sup> gradually

<sup>10</sup> satisfactory

<sup>11</sup> mincut-based heuristic

<sup>12</sup> orthogonal recursive bisection

و شبکه‌های عصبی<sup>۱</sup>. روش‌های محاسبه‌ی تکاملی مختلفی ابداع شده است که زمان‌بندی وظایف را انجام می‌دهد. البته این کار با توجه به تفاوت نمایش‌های مستقیم و غیرمستقیم پروژه انجام می‌شود.

در این مقاله یک رویکرد مستقیم با روش ترکیب احتمالی سریع<sup>۲</sup> و یک نمایش رمزگشایی غیرمستقیم<sup>۳</sup> را بررسی می‌کنیم به طوری که فقط روی نتایج نمایش مستقیم متمرکز می‌شویم. ساختار باقی‌مانده‌ی این مقاله به صورت زیر است: در بخش دوم، الگوریتم‌های موازی شرح داده شده است. در بخش سوم، خلاصه‌ای در مورد الگوریتم ژنتیک بیان شده است. در بخش چهارم مفاهیم الگوریتم ژنتیک ارائه شده است. بخش پنجم به مساله زمان‌بندی وظایف موازی و الگوریتم ژنتیک اختصاص یافته است. انواع نمایش کروموزوم‌ها برای الگوریتم ژنتیک در بخش ششم آورده شده است. بخش هفتم به الگوریتم‌های اولویت‌دهی وظایف تخصیص یافته است. در بخش هشتم نتایج حاصل از بررسی‌های انجام شده شرح داده شده است. در بخش نهم نیز نتیجه‌گیری پایانی آورده شده است.

## ۲ الگوریتم‌های موازی<sup>۴</sup>

یک برنامه‌ی موازی، مجموعه‌ای از کارهاست که بعضی از آن‌ها باید قبل از شروع دیگری کامل شود و ارتباطات تقدمی بین کارها معمولاً به صورت اجمالی در گراف بدون چرخه جهت‌دار<sup>۵</sup> نشان داده می‌شود. مانند یک گراف وظیفه<sup>۶</sup> که گره‌های این گراف همان وظایف<sup>۷</sup> و کمان جهت‌دار بین گره‌ها نشان‌دهنده‌ی ارتباطات تقدمی عامل‌هاست. تعداد کارها و محدودیت‌های تقدمی بین آن‌ها، انجام یک وظیفه را به نحو احسن، مشکل می‌سازد. مساله مورد نظر شامل یافتن یک زمان‌بندی مناسب روی تعداد  $m > 2$  پردازنده با ظرفیت برابر و مینیم کردن زمان پردازش کارهای مستقل، که به کلاس غیرچندجمله‌ای<sup>۸</sup> تعلق دارد، می‌باشد. به طور غیررسمی یک کار موازی، وظیفه‌ای است که از عملیات اصلی یا مقدماتی نتیجه می‌شود. مانند، یک روال عددی، که شرایط کافی برای اجرا شدن به وسیله‌ی بیش از یک پردازنده را دارا هستند. این دیدگاه کلی تر از موارد استاندارد است و کارهای متوالی را مانند یک مورد ویژه شامل می‌شود [۱۸].

## ۲-۱ انواع الگوریتم‌های موازی<sup>۹</sup>

چندین نسخه از کارهای موازی وجود دارد که به اجرائشان روی سیستم موازی و توزیع شده وابسته هستند.

• انعطاف‌ناپذیر<sup>۱۰</sup>: وقتی تعداد پردازنده‌ها برای اجرای PT، ثابت باشد. تعداد آن می‌تواند هم توانی از ۲ یا

هر عدد صحیح دیگری باشد [۱۹].

<sup>1</sup> neural network

<sup>2</sup> as soon as possible crossover

<sup>3</sup> indirect-decode

<sup>4</sup> Parallel Algorithm

<sup>5</sup> directed acyclic graph

<sup>6</sup> task graph

<sup>7</sup> task

<sup>8</sup> NP-complete

<sup>9</sup> Typology of Parallel Task

<sup>10</sup> rigid

- قالبی<sup>۱</sup>: در این مورد تعداد پردازنده‌ها برای اجرا کردن PT ثابت نیست اما قبل از اجرا تعیین می‌شود به طوری که این تعداد تا زمانی که PTها تکمیل شوند تغییر نمی‌کند.
  - انعطاف‌پذیر<sup>۲</sup>: در بیش‌تر موارد تعداد پردازنده‌ها ممکن است در طول اجرا به وسیله‌ی الگوریتم انحصاری و یا ساده با دوباره توزیع کردن داده‌ها تغییر کند.
- از نظر کاربردی بیش‌ترین برنامه‌های کاربردی موازی انعطاف‌پذیرند. یک توسعه‌دهنده‌ی کاربردی، تعداد دقیق پردازنده‌هایی را که قرار است در زمان اجرا به کار گیرد نمی‌داند [۱۸].

### ۳ خلاصه‌ای در مورد الگوریتم ژنتیک

ایده‌ی اصلی الگوریتم‌های ژنتیک بر پایه‌ی نظریه‌ی داروین می‌باشد. براساس نظریه‌ی داروین نسل‌هایی که از ویژگی‌ها و خصوصیات برتری نسبت به نسل‌های دیگر برخوردارند شانس بیش‌تری نیز برای بقا و تکثیر خواهند داشت و ویژگی‌ها و خصوصیات برتر آن‌ها به نسل‌های بعدی آنان نیز منتقل خواهد شد. جواب مساله‌ای که از طریق الگوریتم ژنتیک حل می‌شود مرتباً بهبود می‌یابد. الگوریتم ژنتیک با مجموعه‌ای از جواب‌ها که از طریق کروموزوم‌ها نشان داده می‌شود شروع می‌گردد. در این الگوریتم جواب‌های حاصل از یک جمعیت<sup>۳</sup> برای تولید جمعیت بعدی استفاده می‌شود که عمل تولیدنسل جدید با ترکیب و یا جهش<sup>۴</sup> همراه خواهد بود. در این فرایند امید است که جمعیت جدید نسبت به جمعیت قبلی بهتر باشد. انتخاب بعضی از جواب‌ها (کروموزوم‌ها)<sup>۵</sup> از میان کل جواب‌ها والدین<sup>۶</sup> به منظور ایجاد جواب‌های جدید فرزندان<sup>۷</sup> براساس میزان مطلوبیت آن‌ها می‌باشد که این کار با استفاده از تابع برازش<sup>۸</sup> صورت می‌گیرد. طبیعی است که جواب‌های مناسب‌تر شانس بیش‌تری برای تولید مجدد داشته باشند. این فرایند تا برقراری شرطی که از پیش تعیین شده است (مانند تعداد جمعیت‌ها یا میزان بهبود جواب) ادامه می‌یابد [۲۰].

### ۴ مفاهیم الگوریتم ژنتیک

به طور کلی، الگوریتم‌های ژنتیکی از اجزای زیر تشکیل می‌شوند.

#### ۴-۱ کروموزوم<sup>۹</sup>

در الگوریتم‌های ژنتیکی، هر کروموزوم نشان‌دهنده‌ی یک نقطه در فضای جستجو و یک راه‌حل ممکن برای مساله مورد نظر است. خود کروموزوم‌ها (راه‌حل‌ها) از تعداد ثابتی ژن (متغیر) تشکیل می‌شوند.

<sup>1</sup> moldable  
<sup>2</sup> malleable  
<sup>3</sup> Population  
<sup>4</sup> mutation  
<sup>5</sup> chromosome  
<sup>6</sup> Parent  
<sup>7</sup> Offspring  
<sup>8</sup> fitness  
<sup>9</sup> Chromosome

#### ۴-۲ جمعیت<sup>۱</sup>

مجموعه‌ای از کروموزوم‌ها یک جمعیت را تشکیل می‌دهند. با تاثیر عملگرهای ژنتیکی بر روی هر جمعیت، جمعیت جدیدی با همان تعداد کروموزوم تشکیل می‌شود. یکی از ویژگی‌های ژنتیک این است که به جای تمرکز بر روی یک نقطه از فضای جستجو یا یک کروموزوم، بر روی جمعیتی از کروموزوم‌ها کار می‌کند [۲۰].

#### ۴-۳ تابع برازندگی<sup>۲</sup>

به منظور حل هر مساله با استفاده از الگوریتم‌های ژنتیکی، ابتدا باید یک تابع برازندگی برای آن مساله ابداع شود. برای هر کروموزوم، این تابع عددی غیر منفی را برمی‌گرداند که نشان‌دهنده‌ی شایستگی یا توانایی فردی آن کروموزوم است [۲۰].

#### ۴-۴ عملگرهای ژنتیکی

در الگوریتم‌های ژنتیکی، در طی مرحله‌ی تولیدمثل<sup>۳</sup> از عملگرهای ژنتیکی استفاده می‌شود. با تاثیر این عملگرها بر روی یک جمعیت، نسل<sup>۴</sup> بعدی آن جمعیت تولید می‌شود. عملگرهای انتخاب<sup>۵</sup>، ترکیب و جهش معمولاً بیش‌ترین کاربرد را در الگوریتم‌های ژنتیکی دارند. حال هر یک از عملگرهای فوق به صورت جداگانه معرفی می‌شوند.

##### ۴-۴-۱ عملگر انتخاب

این عملگر از بین کروموزوم‌های موجود در یک جمعیت، تعدادی کروموزوم را برای تولیدمثل انتخاب می‌کند. کروموزوم‌های برازنده‌تر شانس بیشتری دارند تا برای تولیدمثل انتخاب شوند. برخی از روش‌های متداول انتخاب عبارت است از: انتخاب از طریق چرخ رولت، انتخاب از طریق نخبه‌سالاری، انتخاب از طریق رقابتی، انتخاب از طریق مسابقه.

##### ۴-۴-۲ عملگر ترکیب

عملگر ترکیب بر روی یک زوج کروموزوم از نسل مولد عمل کرده و یک زوج کروموزوم جدید تولید می‌کند. عملگرهای ترکیب متعددی از قبیل، ترکیب تک‌نقطه‌ای<sup>۶</sup> و ترکیب دونقطه‌ای<sup>۷</sup> وجود دارد.

##### ۴-۴-۳ عملگر جهش

پس از اتمام عمل ترکیب، عملگر جهش بر روی کروموزوم‌ها اثر داده می‌شود. این عملگر یک ژن از یک کروموزوم را به طور تصادفی انتخاب نموده و سپس محتوای آن ژن را تغییر می‌دهد [۲۱].

<sup>1</sup> Population  
<sup>2</sup> Fitness Function  
<sup>3</sup> Reproduction  
<sup>4</sup> Generation  
<sup>5</sup> Selection  
<sup>6</sup> One-point Crossover  
<sup>7</sup> Two-point Crossover

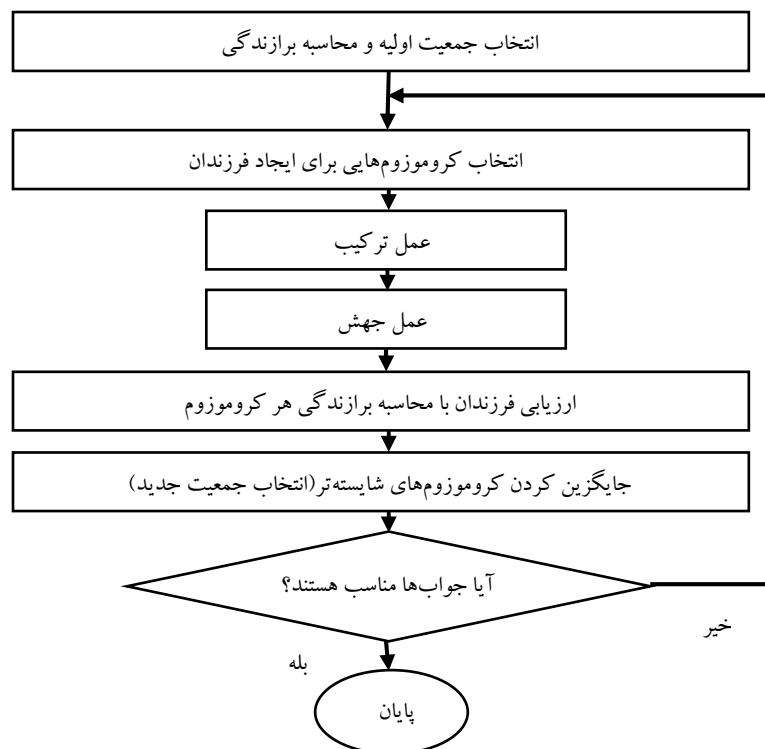
#### ۴-۵ تعیین جمعیت اولیه

الگوریتم ژنتیک کار خود را با تولید جمعیت اولیه‌ای از کروموزوم‌ها آغاز می‌کند و سپس در یک حلقه به طور مکرر تعدادی از کروموزوم‌های برتر نسل فعلی را انتخاب کرده و سپس نسل جدیدی را از این کروموزوم‌ها تولید می‌کند. منظور از تولید جمعیت اولیه، تولید تعدادی جواب برای مساله خواهد بود. تولید جواب‌های اولیه نیز به دو صورت تصادفی و اکتشافی می‌تواند انجام پذیرد.

#### ۴-۶ شرایط توقف الگوریتم ژنتیک

- به تعداد ثابتی نسل برسیم.
- زمان محاسبه‌ی اختصاص داده شده تمام شود.
- بیش‌ترین درجه‌ی برازش برای فرزندان حاصل شود یا دیگر نتایج بهتری حاصل نشود.
- ترکیبی از شرایط بالا.

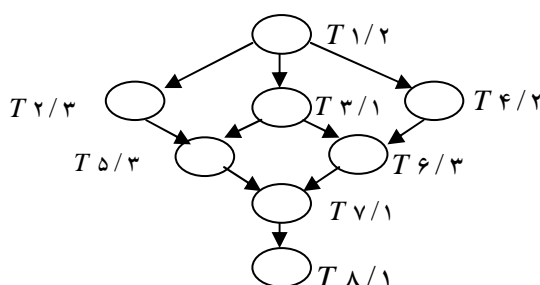
#### ۴-۷ تشریح کلی الگوریتم ژنتیک



شکل ۱. نمودار مراحل الگوریتم ژنتیک

### ۵ مساله زمان بندی وظایف موازی و الگوریتم ژنتیک

مساله زمان بندی وظایف در یک گراف وظیفه به صورت بهینه با به کار بردن  $m$  پردازنده، عبارت است از تخصیص وظایف به مجموعه پردازنده‌ها به طوری که اولویت بین آن‌ها رعایت شود و همچنین پردازش وظایف در کوتاه‌ترین زمان ممکن انجام شود. یک گراف وظیفه، یک نمایش ساده از اجرای برنامه‌های موازی است. البته با نادیده گرفتن سربارها با توجه به توقف‌ها برای دسترسی به منابع و غیره. با وجود این، مبنایی برای تخصیص پردازنده به صورت ایستا ارائه می‌دهد. یک زمان بندی<sup>۱</sup>، به صورت تخصیص کارها یا وظایف به پردازنده‌ها است که معمولاً با نمودار گانت<sup>۲</sup> به تصویر کشیده می‌شود. نمودار گانت به زمان شروع و پایان هر کار در پردازنده‌های در دسترس، اشاره دارد.



شکل ۲. مدل گراف وظیفه

متغیرهای عملکردی دیگری مثل استفاده از پردازنده‌های انفرادی<sup>۳</sup> یا توزیع بار یکنواخت را نیز می‌توان در نظر گرفت. یک برنامه‌ی زمان بندی ساده می‌تواند به صورت بهینه با زمان چند جمله‌ای<sup>۴</sup> حل شود [۲۲].

$P_r$		$T_r$	$T_r$	$T_r$	$T_r$					
$P_1$	$T_1$		$T_r$		$T_r$		$T_r$		$T_r$	$T_r$
Time Slot	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
Time	۰	۱	۲	۳	۴	۵	۶	۷	۸	۹

شکل ۳. زمان بندی ۸ وظیفه روی ۲ پردازنده با روش LSA

### ۶ انواع نمایش کروموزوم‌ها برای الگوریتم ژنتیک

#### ۶-۱ نمایش مستقیم راه حل‌ها

در اینجا یک زمان بندی را به صورت نمایش یک کروموزوم پیشنهاد می‌دهیم. فرض می‌کنیم دو زمان بندی مختلف  $a$  و  $b$  را داریم

شکل ۴ برای مدل گراف وظیفه شکل ۲ نشان داده شده است که به وسیله نمودار گانت ارائه می‌شود.

<sup>1</sup> schedule  
<sup>2</sup> Gant Chart  
<sup>3</sup> individual processor  
<sup>4</sup> P-complete

$P_r$		$T_r$	$T_f$	$T_p$						
$P_1$	$T_1$	$T_r$			$T_o$	$T_v$	$T_a$			
TS	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰

Schedule (a)

$P_r$		$T_r$		$T_o$						$T_a$		
$P_1$	$T_1$	$T_r$			$T_f$	$T_p$		$T_v$				
TS	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲

Schedule (b)

شکل ۴: نمودار گانت زمان‌بندی احتمالی برای مدل گراف وظیفه شکل ۲

ارتباطات تقدمی توصیف شده در گراف وظیفه را می‌توان در ماتریس تقدمی  $A$  نشان داد به طوری که در آن وقتی عضو  $a_{ij}$  در این ماتریس برابر ۱ باشد، یعنی وظیفه‌ی  $i$  اُم مقدم بر وظیفه‌ی  $j$  اُم است و در غیر این صورت برابر صفر می‌باشد. یک ژن<sup>۱</sup> در کروموزوم را به صورت یک چهارتایی زیر نمایش می‌دهیم:

$(task\_id \quad proc\_id \quad inite\_time \quad end\_time)$

$task\_id$ : شناسه‌ای است که به یک وظیفه اختصاص می‌یابد.

$proc\_id$ : شناسه‌ای که به پردازنده‌ای که وظیفه‌ی  $task\_id$  را انجام می‌دهد اختصاص می‌یابد.

$inite\_time$ : زمان شروع وظیفه‌ی  $task\_id$  در پردازنده‌ی  $proc\_id$  می‌باشد.

$end\_time$ : زمان پایان وظیفه‌ی  $task\_id$  در پردازنده‌ی  $proc\_id$  می‌باشد.

$C_a$ :	۱،۱،۰،۰،۲	۲،۱،۰،۲،۵	۳،۲،۰،۲،۳	۴،۲،۰،۳،۵	۵،۱،۵،۵،۸	۶،۲،۵،۵،۸	۷،۱،۸،۹	۸،۱،۹،۱،۰
$C_b$ :	۱،۱،۰،۰،۲	۲،۱،۰،۲،۵	۳،۲،۰،۲،۳	۴،۱،۵،۵،۷	۵،۲،۵،۵،۸	۶،۱،۷،۱،۰	۷،۱،۱۰،۱۱	۸،۲،۱۱،۱،۱۲

شکل ۵: نمایش مستقیم کروموزوم‌ها شکل ۲

نمایش شکل ۵ یک مشکل دارد اگر یک مسیر قراردادی<sup>۲</sup> مثل یک ترکیب تک‌نقطه‌ای نامعتبر به کار گرفته شود، فرزندان نامعتبر (زمان‌بندی‌های غیرمحمول<sup>۳</sup>) ایجاد می‌شوند.

به عنوان مثال اگر بخواهیم عملگر ترکیب را در شکل ۵ بعد از موقعیت پنجم اعمال کنیم دو کروموزوم نامعتبر به دست می‌آید (شکل ۶). هر دو کروموزوم این محدودیت که در یک زمان یک پردازنده یک وظیفه را انجام دهد نقض می‌کنند. به طور مثال ژن ۵ و ۶ در کروموزوم‌های  $C_a$  و  $C_b$  زمان‌بندی نامعتبری را توصیف می‌کنند چون پردازنده‌ی  $P_1$  برای کروموزوم  $C_a$  و  $P_r$  برای کروموزوم  $C_b$  دو وظیفه را در مدت زمان مشابه پردازش می‌کنند.

<sup>1</sup> gene

<sup>2</sup> conventional

<sup>3</sup> feasible scheduling



$C_{a'} :$	۱،۱،۰،۰،۲	۲،۱،۰،۲،۵	۳،۲،۰،۲،۳	۴،۲،۳،۵	۵،۱،۵،۸	۶،۱،۷،۱۰	۷،۱،۱۰،۱۱	۸،۲،۱۱،۱۲
$C_{b'} :$	۱،۱،۰،۰،۲	۲،۱،۰،۲،۵	۳،۲،۰،۲،۳	۴،۱،۵،۷	۵،۲،۵،۸	۶،۲،۵،۸	۷،۱،۸،۹	۸،۱،۹،۱۰

شکل ۶. فرزندان (کروموزوم‌های) نامعتبر مربوط به شکل ۲

الگوریتم ترمیم سعی می‌کند تا راه‌حل‌های معتبر را از میان نامعتبرها پیدا کند این روش از عمل ترکیب پیشنهادی برانز [۲۳] استخراج شده است اگر یک عمل (پردازش وظیفه<sup>۱</sup>) از یکی از والدین‌ها به ارث برسد، نمی‌تواند در آن مدت زمان مشخصی که به پردازنده نسبت داده شده، زمان‌بندی شود در این مورد پردازنده نسبت داده شده به وظیفه، تغییر نمی‌کند، ولی با تأخیر، در پردازش بعدی در دسترس قرار می‌گیرد (شکل ۷).

$C_{a''} :$	۱،۱،۰،۰،۲	۲،۱،۰،۲،۵	۳،۲،۰،۲،۳	۴،۲،۳،۵	۵،۱،۵،۸	۶،۱،۸،۱۱	۷،۱،۱۱،۱۲	۸،۲،۱۲،۱۳
$C_{b''} :$	۱،۱،۰،۰،۲	۲،۱،۰،۲،۵	۳،۲،۰،۲،۳	۴،۱،۵،۷	۵،۲،۵،۸	۶،۲،۸،۱۱	۷،۱،۱۱،۱۲	۸،۱،۱۲،۱۳

شکل ۷. فرزندان با زمان‌بندی محتمل برای مدل گراف وظیفه شکل ۲

$P_1$			$T_1$	$T_2$									$T_8$
$P_1$	$T_1$		$T_2$		$T_3$		$T_4$		$T_5$		$T_6$	$T_7$	
TS	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳

Schedule (a'')

$P_1$			$T_1$			$T_3$							$T_8$
$P_1$	$T_1$		$T_2$		$T_3$		$T_4$		$T_5$		$T_6$	$T_7$	$T_8$
TS	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳

Schedule (b'')

شکل ۸. نمودار گانت فرزندان با زمان‌بندی محتمل برای مدل گراف وظیفه شکل ۲

## ۶-۲ نمایش غیرمستقیم راه‌حل‌ها

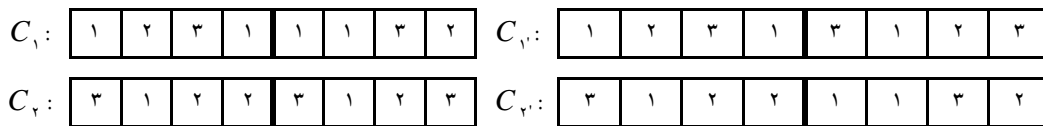
در این روش، یک زمان‌بندی به صورت یک کروموزوم رمزگذاری<sup>۲</sup> می‌شود به طوری که شماره‌ی وظیفه مورد نظر متناظر با موقعیت ژن و به پردازنده‌های تخصیص یافته به هر وظیفه با محتوای آن ژن نشان داده می‌شود. مانند شکل ۹.

Processor	۱	۲	۳	۲	۱	۳	۱	۲
Task	۱	۲	۳	۴	۵	۶	۷	۸

شکل ۹. ساختار کروموزوم برای مساله تخصیص وظایف

<sup>1</sup> task processing  
<sup>2</sup> encoding

این یک ایده برای رمزگشایی<sup>۱</sup> است. یک رمزگشا<sup>۲</sup> یک مسیره‌ی<sup>۳</sup> از فضای نمایش که فضای راه‌حل احتمالی استنتاج شده را ارزیابی می‌کند. در اینجا کروموزوم با دستور دادن به رمزگشا چگونگی ساخت زمان‌بندی احتمالی را تعیین می‌کند. در مورد مساله تخصیص وظیفه، ایجاد زمان‌بندی احتمالی، رمزگشا به روش بعدی دستورهایی را به وسیله‌ی لیست تقدمی بعدی ایجاد می‌کند طی کردن کروموزوم و نسبت دادن وظایف متناظر با پردازنده‌ی مورد نظر به محض این که ارتباط تقدمی‌شان انجام شده باشد. در این روش، محدوده‌ی ممنوعه برای رمزگشا توانایی ایجاد کردن فرزند معتبر است، حتی با روش‌های عملکردی متداول ساده است. از معایب آن هم ارزیابی‌کننده این روش است. برای مدل گراف وظیفه‌ی شکل ۲ با سه پردازنده، والدین کروموزوم  $C_1$  و  $C_2$  و با ترکیب تک نقطه‌ای بعد از موقعیت چهارم، فرزند معتبر  $C_3$  و  $C_4$  به دست می‌آیند.



شکل ۱۰. نمایش غیرمستقیم کروموزوم‌ها

## ۷ الگوریتم‌های اولویت‌دهی وظایف

الگوریتم‌های متنوعی برای اولویت‌دهی وظایف موازی استفاده می‌شود که چند نمونه را در ادامه به طور مختصر بیان می‌کنیم.

### ۷-۱ الگوریتم زمان‌بندی لیست LSA<sup>۵</sup>

روشی که در این مقاله مورد بررسی قرار می‌گیرد این الگوریتم می‌باشد که نتایج حاصل از این روش را با دو روش ذکر شده در بخش‌های ۷-۲ و ۷-۳ مقایسه می‌کنیم. حال به شرح این الگوریتم می‌پردازیم.

برای به دست آوردن لیستی از وظایف مرتب شده براساس تقدم، کارهایی را به پردازنده‌ها نسبت می‌دهیم. با اختصاص دادن هر پردازنده در دسترس (پردازنده‌هایی که وظایف‌شان قبلاً تمام شده باشد) به اولین وظیفه‌ای که اختصاص داده نشده است این کار صورت می‌گیرد. پس داریم:

- $T = \{T_1, \dots, T_n\}$ : یک مجموعه از وظیفه‌هاست.
- $\mu: T \rightarrow (0, \infty)$ : یک تابع است که زمان اجرایی را به هر وظیفه نسبت می‌دهد که بزرگ‌تر مساوی یک ترتیب جزئی ( $\geq$ ) در  $T$  می‌باشد.
- $L$ : لیست تقدمی از وظایف مجموعه‌ی  $T$ .

<sup>1</sup> decoding  
<sup>2</sup> decoder  
<sup>3</sup> mapping  
<sup>4</sup> valid offspring  
<sup>5</sup> List Scheduling Algorithm

در هر زمان یک پردازنده بیکار<sup>۱</sup>، فوراً از لیست  $L$  اولین وظیفه آماده را حذف می کند که همان وظیفه ی زمان بندی نشده می باشد که اجداد آن کوچک تر مساوی (ترتیب جزئی  $\leq$ ) همه وظایف تمام (کامل) شده است. در این مورد دو تا یا بیش تر از پردازنده ها سعی می کنند تا وظایف مشابه را انجام دهند (همان وظایف را انجام دهند). پردازنده ای موفق می شود که دارای کم ترین شناسه و پردازنده های باقی مانده هم به دنبال وظیفه ی مناسب می گردند. به کار بردن این روش اکتشافی LSA در مقابل شواهد<sup>۲</sup> غیر مستدل<sup>۳</sup> می تواند اتفاق بیفتد به عنوان مثال در شکل ۱۱ افزایش تعداد پردازنده ها و کاهش زمان اجرای یک یا تعداد بیش تری از وظایف یا با حذف بعضی محدودیت های تقدیمی که می تواند  $makespan^4$  را افزایش دهد [۱۴].

## ۲-۷ زمان بندی و اولویت دهی وظایف بر حسب ارتفاع

هدف از زمان بندی در یک سیستم چند پردازنده، انتساب  $n$  وظیفه به  $m$  پردازنده در یک سیستم چند پردازنده است تا زمان اتمام اجرای آخرین وظیفه در این سیستم مینیمم شود. برای سادگی، اگر دو وظیفه بر روی دو پردازنده ی مختلف زمان بندی شود هزینه ی ارتباطی برای ارسال داده ها بین دو وظیفه صفر در نظر گرفته می شود. از روش های پر کاربرد در زمان بندی، استفاده از اولویت دهی اجرای وظایف بر اساس ارتفاع آن ها می باشد. الگوریتم تولید زمان بندی کارها بر حسب ارتفاع به صورت زیر می باشد:

۱. وظایف را به ترتیب افزایشی بر حسب ارتفاع آن ها در یک صف مرتب نمایید.
۲. مراحل ۳ و ۴ را به تعداد وظایف تکرار کنید.
۳. یک عدد تصادفی  $r$  به طوری که  $1 \leq r \leq m$  تولید کنید ( $m$  تعداد پردازنده می باشد). اولین کار را از صف وظایف مرتب شده انتخاب کرده و به پردازنده ی  $r$  ام تخصیص دهید و سپس آن را حذف کنید.
۴. با تکرار به کار بردن الگوریتم تولید زمان بندی، یک جمعیت اولیه از گره های گراف مورد جستجو مورد نیاز است.

جدول ۱. شماره و ارتفاع مربوط به هر وظیفه شکل ۱۲

$T_0$	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$	$T_7$	$T_8$	$T_9$	$T_{10}$	$T_{11}$	$T_{12}$	$T_{13}$	$T_{14}$	$T_{15}$
۰	۱	۱	۲	۲	۲	۲	۳	۳	۳	۳	۳	۴	۴	۴	۵

جدول ۲. شماره وظایف و زمان مربوط به هر وظیفه شکل ۱۲

$T_0$	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$	$T_7$	$T_8$	$T_9$	$T_{10}$	$T_{11}$	$T_{12}$	$T_{13}$	$T_{14}$	$T_{15}$
۳	۲	۴	۱	۱۰	۳	۶	۹	۷	۱۱	۵	۵	۸	۱۰	۱۵	۲

<sup>1</sup> idle  
<sup>2</sup> intuition  
<sup>3</sup> anomalies

<sup>4</sup> کل زمان اجرای زمان اجرای برنامه موازی



یکسان برگ باشند و یا این که به برگ‌ها نزدیک باشند و تعدادی از آن‌ها زیرشاخه‌های متعددی داشته باشد و از برگ‌ها دور باشد که در روش اولویت‌دهی براساس ارتفاع، زمان‌بندی بدون توجه به این نکته انجام می‌شود در حالی که در روش اولویت‌دهی براساس تعداد فرزندان و نودگان به این نکته توجه می‌شود و وظایف با زیرشاخه‌ها و نودگان بیش‌تر، اولویت بالاتری دارند و بالطبع شاخه‌های شلوغ‌تر یا وظایف با تعداد نوده بیش‌تر، زودتر زمان‌بندی شوند و با اتمام آن‌ها تعداد وظایف بیش‌تری (فرزندان و نودگان آن‌ها) امکان اجرا می‌یابند و زمان‌بندی بهتری به دست می‌آید. الگوریتم تولید زمان‌بندی کارها وظایف براساس تعداد نودگان:

۱. وظایف را به ترتیب کاهشی برحسب تعداد نودگان آن‌ها در یک صف مرتب کنید.
۲. وظایف با تعداد offspringها را در یک گروه مجزا قرار دهید و مراحل ۳ و ۴ را تا خالی شدن هر گروه انجام دهید.
۳. یک وظیفه از بین وظایف گروه به طور تصادفی انتخاب و از گروه حذف کنید.
۴. سپس وظیفه را براساس روش EST<sup>۱</sup> به یکی از پردازنده‌های  $P_1$  تا  $P_m$  اختصاص دهید به طوری که زمان شروع آن وظیفه بر روی آن پردازنده از پردازنده‌های دیگر کم‌تر باشد.

جدول ۳. شماره و OFFSPRINGهای مربوط به هر وظیفه شکل ۱۲

$T_0$	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$	$T_7$	$T_8$	$T_9$	$T_{10}$	$T_{11}$	$T_{12}$	$T_{13}$	$T_{14}$	$T_{15}$
۱۵	۶	۱۰	۱	۴	۳	۵	۰	۲	۰	۲	۰	۰	۰	۱	۰

جدول ۴. مرتب کردن وظایف براساس تعداد فرزندان مربوط به شکل ۱۲

$T_0$	$T_2$	$T_1$	$T_6$	$T_4$	$T_5$	$T_8$	$T_{10}$	$T_3$	$T_{14}$	$T_7$	$T_9$	$T_{11}$	$T_{12}$	$T_{13}$	$T_{15}$
۱۵	۱۰	۶	۵	۴	۳	۲	۲	۱	۱	۰	۰	۰	۰	۰	۰

## ۸ نتایج بررسی

به منظور ارزیابی الگوریتم پیشنهادی از نرم افزار MATLAB R2014a استفاده شده است. تعداد جمعیت اولیه برای الگوریتم پیشنهادی برابر تعداد وظایف و تعداد نسل‌های الگوریتم کم‌تر از ۱۰۰۰ و آلفا احتمال راه‌حل بهینه با مقدار ۰/۵ و بتا احتمال کل راه‌حل‌های بهینه با مقدار ۰/۹ و نیز زمان اجرای وظایف بین ۱ تا ۱۰۰ واحد زمانی در نظر گرفته شده است. ارزیابی الگوریتم ژنتیک با اولویت‌دهی براساس لیست تقدمی پیشنهادی و الگوریتم‌های ژنتیک با اولویت‌دهی براساس تعداد فرزندان و اولویت‌دهی براساس ارتفاع در جدول‌های ۵ تا ۸ آورده شده است.

<sup>1</sup> Earliest Start Time

**جدول ۵.** نتایج بررسی سه الگوریتم با ۳ پردازنده

تعداد وظایف	۳۰	۵۰	۷۰	۹۰
زمان اتمام الگوریتم فرزندان	۱۰۰۲	۱۴۴۸	۱۹۱۵	۲۵۴۹
زمان اتمام الگوریتم ارتفاع	۱۱۵۸	۱۴۶۵	۱۹۶۹	۲۸۰۶
زمان اتمام الگوریتم لیست	۳۰۱	۴۵۲	۱۵۸۰	۱۹۵۴

**جدول ۶.** نتایج بررسی سه الگوریتم با ۵ پردازنده

تعداد وظایف	۳۰	۵۰	۷۰	۹۰
زمان اتمام الگوریتم فرزندان	۹۱۸	۱۶۳۰	۲۳۰۰	۳۱۹۲
زمان اتمام الگوریتم ارتفاع	۱۰۰۳	۱۶۸۳	۲۳۹۵	۳۲۴۲
زمان اتمام الگوریتم لیست	۲۶۹	۳۹۱	۶۰۳	۱۷۴۶

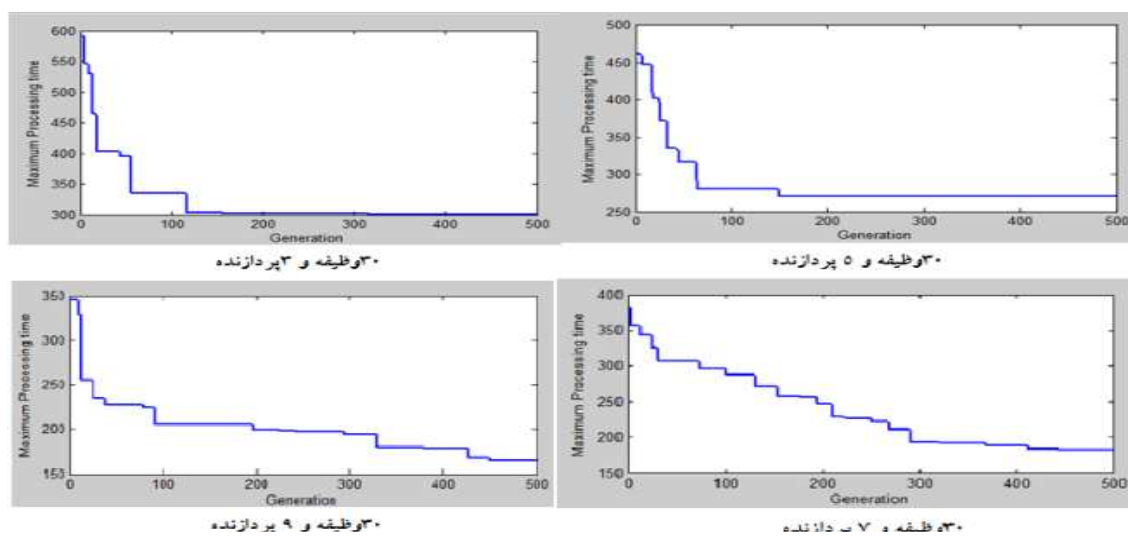
**جدول ۷.** نتایج بررسی سه الگوریتم با ۷ پردازنده

تعداد وظایف	۳۰	۵۰	۷۰	۹۰
زمان اتمام الگوریتم فرزندان	۱۲۲۲	۱۷۴۷	۱۹۸۱	۳۱۹۷
زمان اتمام الگوریتم ارتفاع	۱۲۶۰	۱۷۹۵	۱۹۹۰	۲۳۰۳
زمان اتمام الگوریتم لیست	۱۸۲	۳۰۹	۵۰۶	۶۳۷

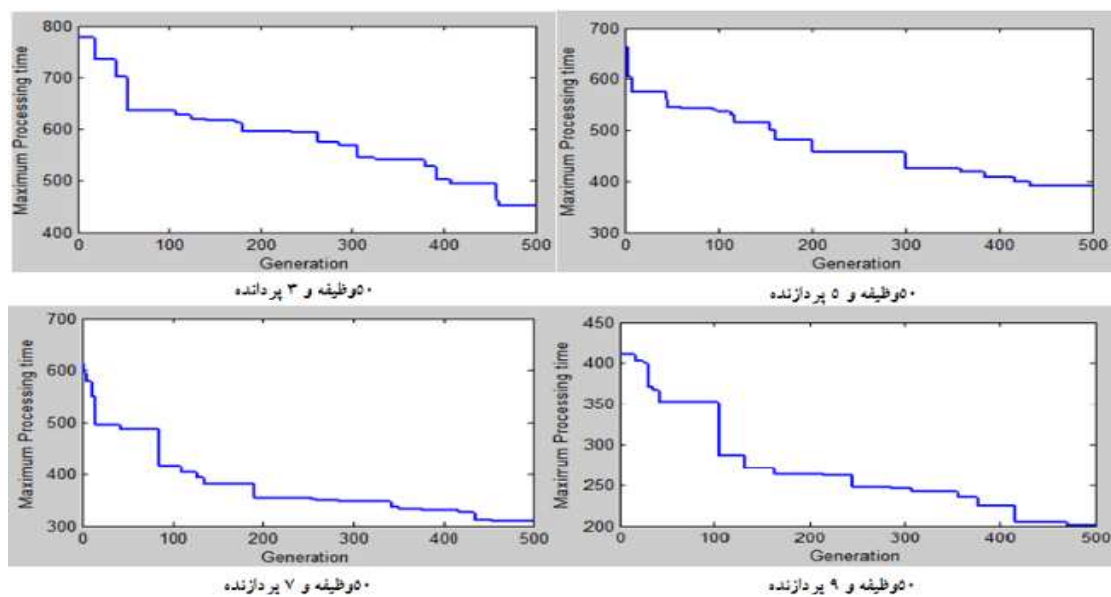
**جدول ۸.** نتایج بررسی سه الگوریتم با ۹ پردازنده

تعداد وظایف	۳۰	۵۰	۷۰	۹۰
زمان اتمام الگوریتم فرزندان	۸۰۹	۱۷۲۰	۲۰۶۵	۲۷۴۲
زمان اتمام الگوریتم ارتفاع	۸۵۶	۱۷۵۳	۲۱۹۱	۲۷۶۰
زمان اتمام الگوریتم لیست	۱۶۵	۲۰۱	۴۶۱	۵۲۸

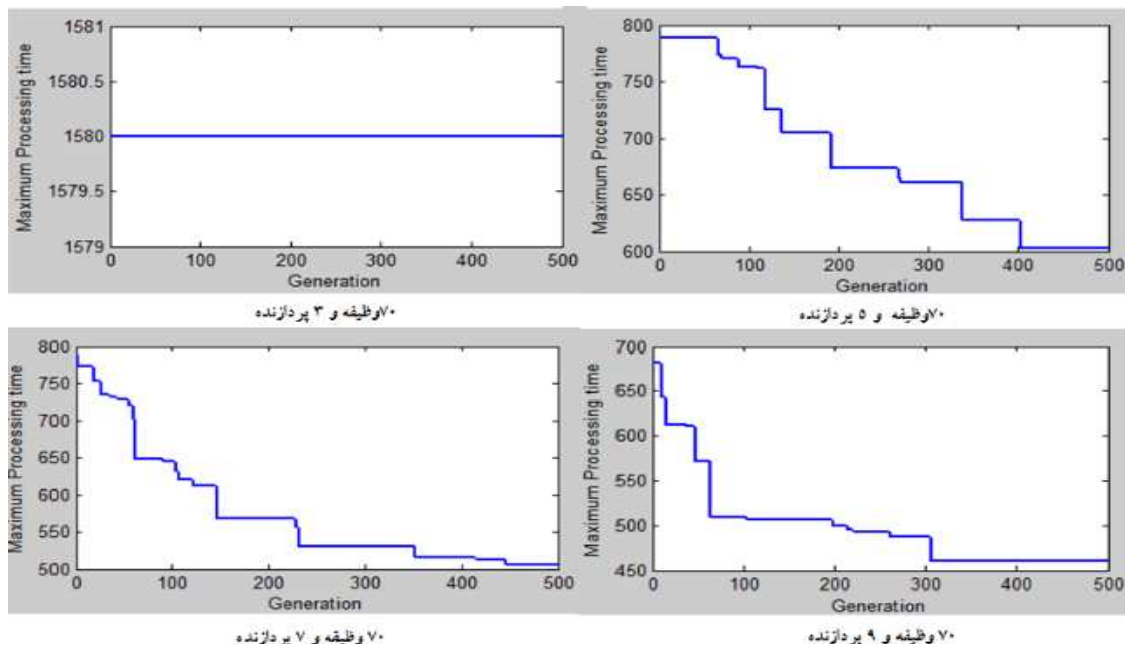
الگوریتم پیشنهادی را برای نمونه گراف‌های ۳۰، ۵۰، ۷۰، ۹۰ وظیفه‌ای اجرا کرده‌ایم نتایج به دست آمده نشان می‌دهد که، زمان اتمام این الگوریتم نسبت به الگوریتم‌های ارتفاع و فرزندان بهبود قابل توجهی یافته است و نیز با افزایش تعداد پردازنده‌ها، تأثیر مناسبی در زمان پایان اجرا برای هر یک از این گراف‌ها دارد. طبق مشاهده نتایج در جداول فوق، با افزایش تعداد وظایف تعداد پردازنده‌ها نیز باید به نسبت افزایش یابد و گرنه کاهش زمان اتمام اجرا اتفاق نمی‌افتد که این مساله چندان مطلوب نمی‌باشد. به طور مثال برای گراف ۹۰ وظیفه‌ای با تعداد ۳ و ۵ پردازنده و برای گراف ۷۰ وظیفه‌ای با تعداد ۳ پردازنده زمان اتمام اجرا زیاد بوده که با افزودن تعداد پردازنده‌ها نتیجه‌ای بهتر حاصل شده است. روند بهبود زمان اتمام اجرا در شکل‌های ۱۳ تا ۱۶ نشان داده شده است.



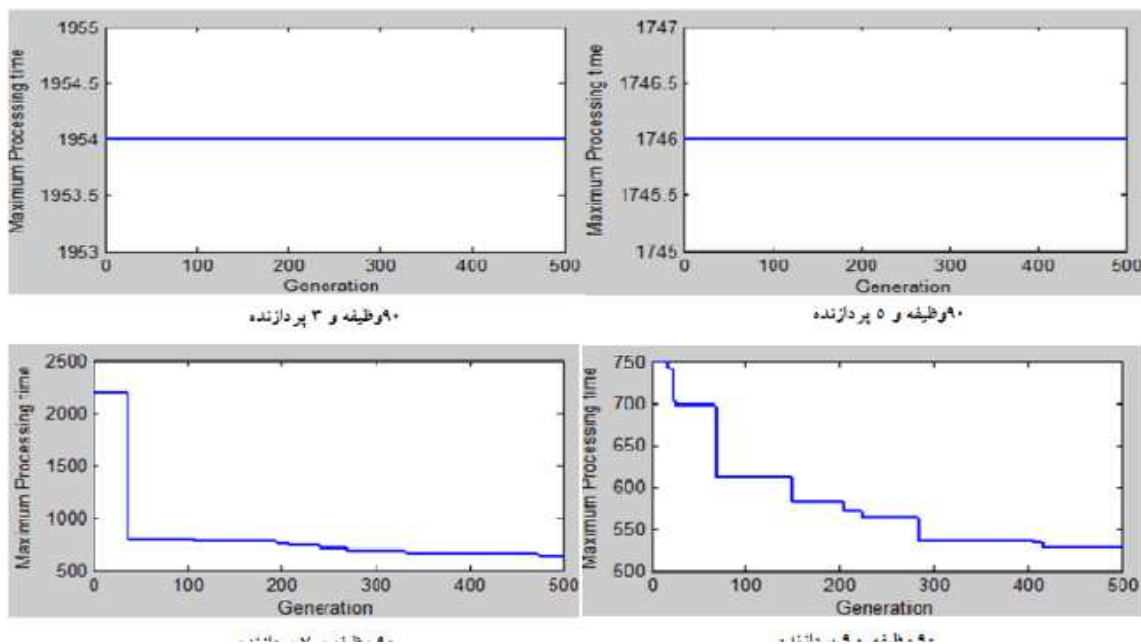
شکل ۱۳. نمودار زمان اجرای گراف ۳۰ وظیفه‌ای با تعداد پردازنده‌های مختلف با الگوریتم لیست



شکل ۱۴. نمودار زمان اجرای گراف ۵۰ وظیفه‌ای با تعداد پردازنده‌های مختلف با الگوریتم لیست



شکل ۱۵. نمودار زمان اجرای گراف ۷۰ وظیفه‌ای با تعداد پردازنده‌های مختلف با الگوریتم لیست



شکل ۱۶. نمودار زمان اجرای گراف ۹۰ وظیفه‌ای با تعداد پردازنده‌های مختلف با الگوریتم لیست

## ۹ نتیجه‌گیری پایانی

تخصیص تعدادی وظایف موازی روی سیستم‌های چندپردازنده یا چند کامپیوتری مشکل است چون نتیجه‌ی زمان‌بندی در سیستم‌های کامپیوتری بسیار اهمیت دارد، در این مقاله قصد داریم تا زمان‌نهایی اجرای وظایف



زمان بندی شده را مینیمم کنیم. نتایج این تحقیق می تواند در مساله های عمومی زیادی از زمان بندی ماشین های موازی به کار رود. ما مساله زمان بندی گراف وظایف اختیاری روی تعداد معقولی پردازنده را در نظر گرفته ایم. در بسیاری از موارد ما خواستار این مضمون هستیم که الگوریتم زمان بندی با زمان چند جمله ای (P-Complete) داشته باشیم، اما در برخی موارد راه حل بهینه وجود ندارد [۲۲].

LSA نوعی الگوریتم زمان بندی را فراهم می کند. الگوریتم ژنتیک که از روش LSA برای اولویت دهی استفاده می کند نتایج بهتری نسبت به دو روش اولویت دهی دیگر به دست آورده است. شبیه سازی این الگوریتم زمان اجرای زمان بندی را بهبود می بخشد. تحقیقات بیش تری نیاز است تا پتانسیل ها و محدودیت های رویکرد الگوریتم ژنتیک روی مجموعه های آزمونی پیچیده تر، نمایش های متفاوت و عملگرهای ژنتیکی مناسب، بیش تر مورد بررسی قرار دهد [۲].

زمان محاسباتی و دقت عددی دو موضوع اساسی در حل مسایل می باشد. عمدتاً روش های مستقیم، زمان محاسباتی کم و دقت پایین دارند و در مقابل روش های ابتکاری زمان محاسباتی بالاتر و دقت بهتری دارند [۲۴].

## منابع

- [۱] مهرابی داودآبادی، ع.، مهرابی داودآبادی، س.، (۱۳۸۸). زمان بندی کارها روی سیستم های چند پردازنده ای با استفاده از الگوریتم های ژنتیک. اولین کنفرانس ملی نرم افزار ایران، رودهن.
- [۲] عبدیزدان، م.، (۱۳۸۶). زمان بندی کارها در سیستم چند پردازنده ای با استفاده از یک الگوریتم ژنتیک جدید اولویت بر اساس تعداد فرزندان. سیزدهمین کنفرانس ملی انجمن کامپیوتر ایران، جزیره کیش، خلیج فارس، ایران.
- [۲۰] محمدی، ل.، (۱۳۹۳). الگوریتم ژنتیک.
- [۲۱] شمس، م.، (۱۳۹۰). پیاده سازی و حل مسائل کاربردی با الگوریتم ژنتیک، آریا پروژه.
- [۲۴] نژادحسین، س.، حیدری، ع.، (۱۳۹۳). حل دسته ای از مسایل کنترل بهینه با استفاده از الگوریتم ژنتیک ترکیبی. تحقیق در عملیات در کاربردهای آن، ۱۱(۳)، ۱۳۷-۱۲۵.

- [3] Dhingra, S., Bal Gupta, S., Biswas, B., (2014). Genetic Algorithm Parameters Optimization for bi-Criteria Multiprocessor Task Scheduling Using Design of Experiments. World Academy of Science, Engineering and Technology International Journal of Computer Electrical Automation control and information, 8.
- [4] Zahorjan, J., McCann, C., (1990). Processor Scheduling in Shared Memory Multiprocessors Performance Evaluation Review, 18(1), 214-225.
- [5] Ercal, F., (1988). Heuristic approaches to task allocation for parallel computing. Doctoral Dissertation, Ohio State University.
- [6] DeJong, K. A., Spears, W. M., (1989). Using Genetic Algorithms to Solve NP-complete Problems. Proceedings of the 3rd International Conference on Genetic Algorithms, 124-132.
- [7] Holland, J. H., (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press.
- [8] Thomas, J., Naughton, A. J., (2005). Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing. Proceedings of the 19th IEEE/ACM International parallel and distributed processing symposium, Denver USA. 207-1530.
- [9] Kim, E. S., Sung, C. S., Lee, I. S., (2009). Scheduling of parallel machines to minimize total completion time subject to S-precedence constraints. The Journal of Computers and Operations Research, 36, 689-710.

- [10] Wu, A. S., Jin, Yu. H. S., Lin, K. C., Schiavone, G., (2004). An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 15(9), 824–834.
- [11] Cena, M., Crespo, M., Gallard, R., (1995). Transparent remote execution in LAHNOS by means of a neural network device. *ACM Press Operating Systems Review*, 29(1), 17-28.
- [12] Flower, J., Otto, S., Salama, M., (1987). Optimal mapping of irregular finite element domains to parallel processors, 292.
- [13] Fox, G. C., (1988). A review of automatic load balancing and decomposition method for the hipercube, In M. Shultz ed. *Numerical algorithms for modern parallel computer architectures*, Springer Verlag, 63-76.
- [14] Graham, R. L., (1972). Bounds on multiprocessing anomalies and packing algorithm. *Proceedings of the AFIPS (1972) Spring Joint Computer Conference*, 205-217.
- [15] Kidwell, M., (1993). using genetic algorithm to schedule task on bus based system, *Proceeding of the Fifth International Conference on Genetic Algorithms*, 368-374.
- [16] Fox, G. C., Kolawa A., Williams, R., (1987). The implementation of a dynamic load balancer. *2nd Conference on Hipercube multiprocessors*, 114-121.
- [17] Mansour, N., Fox, G. C., (1991). A hybrid genetic algorithm for task allocation in multicomputers. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 466-473.
- [18] Dutot, P. F., Mounie, G., Trystram, D., (2004). *Scheduling Parallel Tasks Approximation Algorithms*.
- [19] Lodi, A., Martello, S., Monaci, M., *Two dimensional packing problems. A survey European*.
- [22] ANPCYT<sup>1</sup>, (2014). A genetic approach using direct representation of solutions for the parallel task scheduling problem. *Departamento de Informática. Universidad Nacional de San Luis. San Luis, Argentina*.
- [23] Bruns, R., (1993). Direct chromosome representation and advanced genetic operators for production scheduling. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 352-359.

---

<sup>1</sup>National Agency to Promote Science and Technology