نشـــریه بین المللـــی مهنـــدسی صنـایع و مدیـــریت تولیـــد تابستان ۱۳۸۸- صفحـه ۷۵-۵۷

شمـــاره ۲، جلـــد ۲۰، تابستـــان ۱۳۸۸

نشـــریه بین المللـــی مهنـــدسی صنایـــع و مدیـــریت تولیـــد

نشریه بین المللی مهندسی صنایع و مدیریت تولید دانشگاه علم و صنعت ایران

ISSN: 2008-4870

Journal Website: http://IJIEPM.iust.ac.ir/

A New Hybrid Particle Swarm Optimization for Job Shop Scheduling Problem

Mehdi Behroozi & Kourosh Eshghi*

Mehdi Behroozi *Industrial Engineering Department, Sharif University of Technology, mehdi.behroozi@Gmail.com*
Kourosh Eshghi *Department of Industrial Engineering, Sharif University eshghi@sharif.edu*

Keywords

Job Shop Scheduling Problem,
Particle Swarm Optimization (PSO),
Factoradic,
Simulated Annealing (SA),
GRASP

ABSTRACT

The classical Job Shop Scheduling Problem (JSSP) is NP-hard problem in the strong sense. For this reason, different metaheuristic algorithms have been developed for solving the JSSP in recent years. The Particle Swarm Optimization (PSO), as a new metaheuristic algorithm, has applied to a few special classes of the problem. In this paper, a new PSO algorithm is developed for JSSP. First, a preference list of the generated solutions is prepared to obtain the feasibility. Then, a new method using factorial base numeral system, called factoradic approach, is developed to satisfy the validity of the generated solutions. This approach permits a one to one mapping of a solution in discrete space to a PSO particle position in continuous space. Since PSO is an evolutionary approach, some modifications are implemented to the algorithm. For examples, a simple greedy algorithm is developed to generate relatively good initial population or every solution obtained by PSO is also improved by a local search operator. To avoid trapping into local optima, a new velocity updating equation is considered. Furthermore, a Simulated Annealing (SA) approach is applied to the final solution obtained by PSO to improve it. Finally, the proposed hybrid algorithm is tested by some job shop benchmark problems. The results indicate the efficiency of the proposed hybrid PSO with respect to other algorithms exist in the literature for the considered problem.

بکارگیری الگوریتم ترکیبی بهینه سازی دسته ذرات برای حل مساله سنتی زمانبندی کار کارگاهی

مهدی بهروزی و کوروش عشقی

کلمات کلیدی

مساله زمانبندی کار کارگاهی،
بهینه‌سازی دسته ذرات،
فاکتورادیک،
آنیلینگ شبیه‌سازی شده،
جستجوی تصادفی حریصانه

چکیده:

مساله زمانبندی کار کارگاهی سنتی یک مساله NP-Complete از نوع قوی است و به همین دلیل در تحقیقات صورت گرفته، الگوریتم‌های فراابتکاری زیادی برای حل آن ارایه شده است، اما تنها در تعداد معدودی از آنها الگوریتم بهینه‌سازی دسته ذرات (PSO) مورد توجه قرار گرفته است که یکی از دلایل آن می‌تواند جدید بودن این روش باشد. در الگوریتم ارایه شده در این مقاله ابتدا بمنظور حفظ موجه بودن جوابها در هر تکرار الگوریتم شیوه نمایش بر مبنای فهرست اولویت برای

تاریخ وصول: ۸۷/۹/۲۲

تاریخ تصویب: ۸۸/۸/۲۶

مهدی بهروزی، فارغ التحصیل کارشناسی ارشد، دانشکده صنایع، دانشگاه صنعتی شریف، mehdi.behroozi@gmail.com

کوروش عشقی، استاد، دانشکده صنایع، دانشگاه صنعتی شریف، eshghi@sharif.edu

جوابها انتخاب شده است. همچنین برای ایجاد رابطه یک به یک بین جواب مساله که ماهیت گسسته دارد و جواب مورد پذیرش الگوریتم که اعداد پیوسته هستند و همچنین حفظ قانونی بودن جوابها در هر تکرار، یک شیوه جدید بر اساس تبدیل مبنای اعداد و با استفاده از نمایش اعداد در مبنای فاکتوریل توسعه داده شده است. با توجه به تکاملی بودن PSO، بمنظور شروع از جوابهای نسبتا خوب از یک الگوریتم ابتکاری جستجوی تصادفی حریصانه به عنوان مولد جوابهای اولیه استفاده شده است. هر جواب بدست آمده توسط PSO بوسیله یک الگوریتم جستجوی محلی بهبود داده می‌شود. برای فرار از دام بهینه‌های محلی یک رابطه جدید بهنگام سازی سرعت در الگوریتم PSO توسعه داده شده است. به همین منظور و همچنین برای بهبود نهایی جواب بدست آمده توسط PSO از یک الگوریتم آنیلینگ شبیه‌سازی شده استفاده شده است. الگوریتم بر روی تعدادی از مسایل نمونه آزمایش شده و نتایج حاصل بیانگر دقت و کارایی جوابها نسبت به سایر الگوریتم‌های موجود برای حل مساله مورد بحث است.

۱. مقدمه

مساله زمانبندی کار کارگاهی سنتی (JSSP)[1] عمومی‌ترین مدل سنتی زمانبندی است که طیف وسیعی از مسایل عملی زمانبندی را در خود جای می‌دهد. از کاربردهای مساله کار کارگاهی در صنایع و خدمات می‌توان به تولید ویفر[2] در صنایع تولید نیمه‌رساناها[3]، مراجعه بیماران به بیمارستان و رسیدگی به آنها [۱] و یا مراجعه کنندگان به یک اداره اشاره کرد. مساله JSSP را می‌توان با استفاده از شیوه نگارش سه تایی گراهام [۲] به صورت $Jm||C_{\max}$ نشان داد که تعریف آن در ادامه بطور خلاصه ارایه می‌شود.

در مساله زمانبندی کار کارگاهی m ماشین و n کار وجود دارد که در آن کارها باید بر روی ماشینها پردازش شوند. در این مدل می‌توان فرض کرد که هر کاری باید بر روی همه ماشینها پردازش شود پردازش کار j بر روی ماشین i یک فعالیت نامیده می‌شود و به صورت زوج مرتب (i, j) و یا بصورت O_{ij} نشان داده می‌شود. بین هر دو فعالیت یک کار یک رابطه پیش‌نیازی وجود دارد ولی بین هر دو فعالیت از کارهای مختلف هیچگونه رابطه پیش‌نیازی وجود ندارد.. هر یک از کارها برای پردازش بر روی ماشینها دارای یک مسیر پردازش مختص خود است که بوسیله توالی ماشینها مشخص می‌شود و در واقع روابط پیش نیازی مابین فعالیتهای آن را نشان می‌دهد. در این مقاله، برای نشان دادن مسیر پردازش هر کار ابتدا کلیه ماشینهای موجود را به صورت $M_1, M_2, ..., M_m$ نامگذاری می‌کنیم و سپس مسیر پردازش را به صورت مجموعه $JP_j=\{M_{1j}=i_1, M_{2j}=i_2, ..., M_{mj}=i_m\}$ تعریف می‌کنیم که در آن مقدار i_k در عبارت $M_{kj}=i_k$; $\forall k=1,2,...,m$ برابر با عددی است که نشان دهنده‌ی اولویت ماشین M_k (فعالیت O_{kj}) در مسیر پردازش کار j است و عددی بین ۱ تا m را اختیار می‌کند.

در واقع عبارت $M_{kj}=i_k$ نشان می‌دهد که ماشین M_k، i_k امین ماشین در مسیر پردازش کار j خواهد بود. در این مساله هر کار در هر لحظه زمانی تنها می‌تواند بر روی یک ماشین پردازش شود و هر ماشین نیز در هر لحظه زمانی حداکثر می‌تواند یک کار را پردازش کند. همه کارها در زمان صفر در دسترس هستند و هیچ یک از کارها دارای موعد تحویل نمی‌باشند. هدف مساله یافتن یک زمانبندی است که زمان پایان تمام کارها (کار آخر) را کمینه کند.

گری و همکارانش [۳] ثابت کردند که JSSP یک مساله NP-hard است. بنابراین نمی‌توان جواب بهینه آن را در ابعاد بزرگ در زمان معقولی بصورت دقیق بدست آورد. به همین دلیل الگوریتمهای ابتکاری و فرا ابتکاری زیادی برای این مسایل ارایه شده است که از آن میان می‌توان به الگوریتم ابتکاری انتقال گلوگاه (SB)[4] [۴]، الگوریتم جستجوی ممنوع (TS)[5] در ترکیب با الگوریتم انتقال گلوگاه [۵]، الگوریتم ژنتیک (GA)[6] [۶] و [۷]، الگوریتم‌های ژنتیک در ترکیب با یک الگوریتم جستجوی محلی [۸] و [۹]، الگوریتم آنیلینگ شبیه‌سازی شده (SA)[7] [۱۰، ۱۱ و ۱۲]، الگوریتم بهینه سازی اجتماع مورچگان (ACO)[8] [۱۳]، الگوریتم بهینه سازی دسته ذرات (PSO)[9] در ترکیب با الگوریتم آنیلینگ شبیه سازی شده [۱۴] و الگوریتم بهینه سازی دسته ذرات در ترکیب با الگوریتم جستجوی ممنوع [۱۵] اشاره کرد.

در ادامه مقاله در بخش ۲ پیشینه الگوریتم بهینه‌سازی دسته ذرات ارایه می‌شود. در بخش ۳ ساختار جواب و نحوه تعریف آن در الگوریتم شرح داده شده است و در بخش ۴ الگوریتمی برای تولید جوابهای اولیه توسعه داده شده است. الگوریتم پیشنهادی در بخش ۵ ارایه شده است. در بخش ۶ الگوریتم پیشنهادی بر روی تعدادی از مسایل نمونه آزمایش شده و نتایج حاصل از آن گزارش شده

[1] *Job Shop Scheduling Problem*

[2] Wafer: یک لایه سیلیکونی که بر روی آن مدارهای مجتمع جهت ایجاد یک تراشه قرار می‌گیرند.

[3] *Semiconductors fabrication facilities (fabs)*

[4] *Shifting Bottleneck Procedure (SBP)*

[5] *Tabu Search*

[6] *Genetic Algorithm*

[7] *Simulated Annealing*

[8] *Ant Colony Optimization*

[9] *Particle Swarm Optimization*

است. سرانجام نتیجه‌گیری و پیشنهادهایی برای مطالعات آتی در بخش ۷ ارایه شده است.

۲. الگوریتم بهینه‌سازی دسته ذرات

الگوریتم PSO توسط کندی و ابرهارت در طی دو مقاله [۱۶] و [۱۷] برای دسته مسایل بهینه‌سازی که ماهیت پیوسته بر جوابهای آنها حاکم است، ارایه شد. الگوریتم PSO یک الگوریتم تکاملی[1] است و با یک جمعیت اولیه از جوابهای تصادفی شروع می‌کند. هر جواب بالقوه را یک ذره می‌نامد که در یک ابرفضا با یک سرعت تصادفی که در هر مرحله به آن تخصیص داده می‌شود، در حال پرواز از موقعیتی به موقعیت دیگر است. به بهترین موقعیت هر ذره در طی مراحل قبلی با توجه به یک تابع برازندگی[2] (تابع هدف) مقداری تخصیص می‌یابد و این مقدار در متغیری به نام ***pbest*** که مختص آن ذره است، ذخیره می‌شود. گونه‌ای از PSO که دارای جهت‌گیری عمومی[3] است و در این مقاله مورد استفاده قرار می‌گیرد، بهترین مقدار مربوط به موقعیت کلیه ذرات جمعیت در طی مراحل قبلی را در متغیری به نام ***gbest*** ذخیره می‌کند. در هر مرحله با تغییر هر یک از بردارهای سرعت v_i، هر ذره i به سمت موقعیت‌های متناظر با مقادیر ***pbest*** و ***gbest*** حرکت می‌کند و اگر به موقعیت‌های بهتری نسبت به آنها رسید آنها را بهنگام می‌کند. هر یک از مؤلفه‌های بردار v_i باید در بازه $[-V_{\max}, V_{\max}]$ قرار داشته باشد که در آن $V_{\max}$ بوسیله محدوده مجاز برای حرکت هر ذره و یا بوسیله طراح الگوریتم جهت ساماندهی میزان پراکندگی و دقت جستجو تعیین می‌شود [۱۷].

در مدل اولیه الگوریتم در هر مرحله بردار سرعت و موقعیت هر ذره i بترتیب بوسیله روابط (۱) و (۲) بهنگام می‌شوند:

$$\vec{v}_i \leftarrow \vec{v}_i + c_1\vec{r}_1 \otimes (\vec{p}_i - \vec{x}_i) + c_2\vec{r}_2 \otimes (\vec{p}_g - \vec{x}_i) \qquad (۱)$$

$$\vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i \qquad (۲)$$

که در آن و ثوابت شتاب، $\vec{r}_1$ و $\vec{r}_2$ بردار اعداد تصادفی در بازه [0,1]، $\vec{p}_i$ بردار بهترین موقعیت بدست آمده توسط ذره i و $\vec{p}_g$ بردار بهترین موقعیت بدست آمده توسط تمام ذرات هستند. علامت $\otimes$ نیز ضرب مؤلفه به مؤلفه دو بردار را نشان می‌دهد. در توسعه‌های فراوانی که برای PSO ارایه شده است، اغلب نوآوری‌ها بصورت تغییر این دو رابطه اصلی بوده است. یکی از این توسعه‌ها اضافه کردن ضریب وزن اینرسی[4] به جمله سرعت فعلی ذره است [۱۸]. در این حالت رابطه سرعت بصورت رابطه (۳) در می‌آید:

$$\vec{v}_i \leftarrow w\vec{v}_i + c_1\vec{r}_1 \otimes (\vec{p}_i - \vec{x}_i) + c_2\vec{r}_2 \otimes (\vec{p}_g - \vec{x}_i) \qquad (۳)$$

که در آن w ضریب وزن اینرسی است که مقدار آن بصورت دلخواه تعیین می‌شود.

مقادیر $w>1$ ممکن است باعث ناپایدار شدن سیستم شود در حالی که در بین مقادیر $w<1$ مقادیر نزدیک به ۱ به منزله جستجوی بیشتر در کل فضای جستجو است در حالی که مقادیر کوچکتر به منزله جستجوی دقیق‌تر در اطراف نقاط بهینه محلی است. معمولا توصیه می‌شود که w از یک مقدار بزرگ شروع شود و بتدریج در هر مرحله کم شده تا به یک مقدار کوچک برسد. استفاده از ضریب اینرسی باعث می‌شود که نیاز به درنظر گرفتن $V_{\max}$ دلخواه برای جلوگیری از پراکندگی زیاد ذرات تا حد ممکن کم شود.

یکی دیگر از روشهایی که برای PSO توسعه داده شده است و باعث عدم نیاز به $V_{\max}$ می‌شود، در نظر گرفتن ضریب انقباض[5] است که در هر مرحله مقدار سرعت را در ضریبی کوچکتر از ۱ ضرب می‌کند تا از سرعت پراکنده شدن بیش از حد ذرات بکاهد [۱۸]. در این روش دو فرمول اصلی بصورت رابطه (۴) در می‌آید:

$$\vec{v}_i \leftarrow \chi\left(\vec{v}_i + c_1\vec{r}_1 \otimes (\vec{p}_i - \vec{x}_i) + c_2\vec{r}_2 \otimes (\vec{p}_g - \vec{x}_i)\right) \qquad (۴)$$

که در آن مقدار χ برابر با $\chi=\frac{2}{C-2+\sqrt{C^2-4C}}$, $C=c_1+c_2>4$ است.

در این تحقیق از یک رابطه جدید بهنگام سازی سرعت ذره استفاده شده است و نتایج بهتری حاصل شده است. این رابطه از ترکیب دو رابطه (۳) و (۴) بدست می‌آید و هر دو ضریب اینرسی و انقباض را برای جستجوی بهتر و دقیق‌تر فضای جواب بکار می‌گیرد. این رابطه در بخش الگوریتم پیشنهادی در رابطه (۱۰) ارایه شده است.

۳. ساختار جواب و نحوه تعریف آن در الگوریتم

۳-۱. روش نمایش جواب

در این مقاله از روش نمایش بر مبنای فهرست اولویت[6] برای نمایش جوابهای مساله استفاده شده است. در روش نمایش بر مبنای فهرست اولویت جواب بصورت یک رشته از کارها نشان داده می‌شود. یک رشته از m زیر رشته که هر یک از آنها متناظر با یک ماشین تعریف شده است، تشکیل می‌شود، به عبارت دیگر هر ماشین فهرست اولویت خاص خود را دارد.

[1] *Evolutionary Algorithm*
[2] *Fitness function*
[3] *Globally Oriented (GBEST)*
[4] *Inertia weight*
[5] *Constriction Coefficient*
[6] *Preference list based representation*

هر زیر رشته بصورت آرایه (رشته)ای n تایی از اعداد تعریف می‌شود که در آن هر عدد نشان دهنده‌ی فعالیتی از کار متناظر با خود است که باید بر روی ماشین متناظر با آن زیر رشته پردازش شود. این زیر رشته‌ها توالی پردازش فعالیتها بر روی ماشین را نشان نمی‌دهند که اگر اینگونه بود ممکن بود جوابهای تولیدی موجه نباشند (محدودیت مسیر پردازش کارها را رعایت نکنند). این زیر رشته‌ها در واقع فهرستهای اولویت هستند و اولویت پردازش یک کار بر کار بعدی خود را در صورتی که قابل انجام باشد، نشان می‌دهد [۱۹].

برای روشن شدن مطلب نمونه‌ای از نمایش یک جواب در یک مساله کار کارگاهی با ۳ ماشین و ۳ کار بر مبنای فهرست اولویت در شکل ۱ نشان داده شده است. زیر رشته اول که مربوط به ماشین M_1 است به این معنا است که در هر مرحله کار J_2 بر کار J_1 برای پردازش بر روی M_1 اولویت دارد و J_1 نیز به نوبه‌ی خود بر J_3 اولویت دارد.

اولویت داشتن کار J_2 بر کار J_1 برای پردازش بر روی M_1 بدین معنا نیست که حتما باید زودتر از آن پردازش شود. تنها در صورتی که قابلیت پردازش را داشته باشد (پیشنیازهای آن پردازش شده باشند) آنگاه زودتر انجام می‌شود در غیر اینصورت کار J_1 اگر قابل انجام باشد بر کار J_2 مقدم خواهد شد.

در روشهای حل فرا ابتکاری مسایل زمانبندی جواب قانونی[1] به جوابی گفته می‌شود که در توالی کارهای هر ماشین، هر کاری دقیقا یک بار ذکر شده باشد. با استفاده از روش نمایش بر مبنای فهرست اولویت، زمانبندی بدست آمده از هر جواب قانونی، با توجه به اینکه در هر مرحله تنها فعالیتهایی را اختصاص می‌دهیم که فعالیتهای پیش نیاز آن انجام شده است محدودیت روابط پیش‌نیازی فعالیتها (محدودیت مسیر پردازش کارها) را برآورده می‌کند و در نتیجه موجه نیز است.

۳-۲. انطباق الگوریتم PSO با جوابهای مساله

جواب مساله JSSP بصورت توالی اعداد و گسسته است در حالی که جواب مورد پذیرش الگوریتم PSO برداری از اعداد پیوسته است. در اینجا برای انطباق شیوه نمایش جواب بر مبنای فهرست اولویت با شیوه مورد پذیرش الگوریتم، یک شیوه تبدیل جدید با استفاده از تبدیل مبنای اعداد توسعه داده شده است.

با بکارگیری این روش تناظری یک به یک بین اعداد هر زیر رشته با یک عدد پیوسته مبنای ۱۰ ایجاد می‌شود و در نهایت با استفاده از اعداد تولید شده توسط تمام زیر رشته‌ها یک بردار m بعدی بصورت $X = (x_1, x_2, ..., x_m)$ بدست می‌آید که در واقع جواب مورد پذیرش PSO است و نشان دهنده‌ی موقعیت مختصاتی یک ذره از جمعیت مورد استفاده در ابرفضای m بعدی است. تبدیل مبنای مورد استفاده تبدیل مبنای فاکتوریلی[2] نام دارد و به طور غیر مستقیم عمل تبدیل یک جایگشت به یک عدد مبنای ۱۰ و بالعکس را با یک واسطه انجام می‌دهد [۲۰].

این تبدیل باعث می‌شود علاوه بر تولید جوابهای صرفا قانونی یک تناظر یک به یک نیز بین توالی‌ها (جایگشت‌ها) و موقعیت ذرات در الگوریتم PSO ایجاد شود.

۳-۲-۱. سیستم نمایش اعدادرمبنای فاکتوریلی (فاکتورادیک) و شیوه تبدیل آنها

فاکتورادیک[3] یک سیستم نمایش اعداد در یک مبنای مختلط با پایه فاکتوریل است که در آن i امین رقم با شمارش از راست در مقدار $i!$ ضرب می‌شود . حاصل جمع آنها عدد در مبنای ۱۰ (عدد دهدهی) را نتیجه می‌دهد. به عنوان مثال عدد ۳۸ در مبنای ۱۰ را می‌توان بصورت منحصر به فرد زیر در مبنای فاکتوریل نوشت:

$$38 = (1_4 2_3 1_2 0_1 0_0)_f = 1\times 4! + 2\times 3! + 1\times 2! + 0\times 1! + 0\times 0!$$

در این سیستم نمایش اعداد اولین رقم از سمت راست همواره 0 است، دومین رقم 0 یا 1 است، سومین رقم 0، 1 یا 2 است و همینطور تا n امین رقم که 0، 1، ... یا $(n\text{-}1)$ است [۲۱]. البته می‌توان از رقم اول سمت راست صرفنظر کرد چون همواره صفر است. یک عدد در مبنای ۱۰ مثل y را نیز می‌توان با استفاده از تبدیل معکوس زیر به یک عدد در مبنای فاکتوریلی بصورت $d_n d_{(n-1)} ... d_1$ تبدیل کرد.

$$y_n \leftrightarrow d_n d_{(n-1)} ... d_1 \qquad (۵)$$

$$y_n = y_{(n-1)} = y \qquad (۶)$$

$$y_j = \left\lfloor \frac{y_{j+1}}{n-j} \right\rfloor \quad , \quad j = (n-2), ..., 2, 1 \qquad (۷)$$

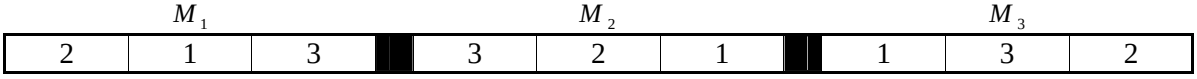
$$d_j = y_{(n-j+1)} - \left\lfloor \frac{y_{(n-j+1)}}{j} \right\rfloor \times (j) \quad , \quad j = 1, 2, ..., n \qquad (۸)$$

در مبنای فاکتوریل بزرگترین عدد n رقمی برابر با $(n-1)_{n-1}(n-2)_{n-2}...1_1 0_0 = n! - 1$ است و بطور مشابه کوچکترین عدد n رقمی در مبنای فاکتوریل برابر با $0_{n-1} 0_{n-2} ... 0_1 0_0 = 0$ است.

[1] *Legal solution*

[2] *Factorial base transformation*

[3] *Factoradic*



شکل ۱. نمونه‌ای از شیوه نمایش یک جواب برای مساله

جدول ۱. نمونه ای از تناظر بین فاکتورادیک و جایگشت با n=3

decimal	factoradic	Permutation of (0,1,2)	Permutation of (1,2,3)
0_{10}	$0_2 0_1 0_0$	0-1-2	1-2-3
1_{10}	$0_2 1_1 0_0$	0-2-1	1-3-2
2_{10}	$1_2 0_1 0_0$	1-0-2	2-1-3
3_{10}	$1_2 1_1 0_0$	1-2-0	2-3-1
4_{10}	$2_2 0_1 0_0$	2-0-1	3-1-2
5_{10}	$2_2 1_1 0_0$	2-1-0	3-2-1

۳-۲-۲. ارتباط بین اعداد مبنای فاکتوریلی و جایگشت ها

بین اعداد صحیح $0,...,n!-1$ (یا بطور معادل اعداد n رقمی فاکتورادیک) با جایگشت‌های ممکن از n عنصر (که می‌توانند اعداد از 1 تا n باشند) یک تناظر طبیعی به ترتیب الفبایی وجود دارد. نمونه‌ای از این تناظر در جدول (۱) نشان داده شده است. به منظور تبدیل نمایش فاکتورادیک به یک جایگشت از n عنصر مثلا اعداد 1, 2, 3 از الگوریتم زیر استفاده می‌کنیم [۲۰]:

گام ۱) فهرستی به نام F بصورت $F=(0,1,2)$ از ارقام ممکن در نمایش فاکتورادیک و فهرست دیگری به نام P بصورت $P=(1,2,3)$ از عناصر ممکن در جایگشت تهیه کنید.

گام ۲) k امین رقم از سمت چپ در نمایش فاکتورادیک را در نظر بگیرید و مشخص کنید که چندمین رقم از فهرست F است. فرض کنید i امین رقم باشد. i امین عنصر از فهرست P فعلی را بعنوان k امین عنصر جایگشت از سمت چپ انتخاب کرده و آن را از فهرست P حذف کنید ($k=1,...,n$).

به منظور تبدیل یک جایگشت از n عنصر به نمایش فاکتورادیک تنها گام ۲ الگوریتم فوق بصورت زیر تغییر می‌یابد [۲۰]:

گام ۲) k امین عنصر از سمت چپ در جایگشت را در نظر بگیرید و ببینید که چندمین عنصر از فهرست P فعلی است. فرض کنید i امین عنصر باشد. آن را از فهرست P حذف کرده و i امین عنصر از فهرست F را بعنوان k امین رقم نمایش فاکتورادیک از سمت چپ انتخاب کنید ($k=1,...,n$).

به عنوان مثال تبدیل $3_{10}=(1_2 1_1 0_0)=2-3-1$ را در نظر بگیرید که در شکل ۲ نشان داده شده است.

لازم به ذکر است که پیچیدگی زمانی این تبدیل جدید در هر دو جهت، چند جمله‌ای ($O(n)$) بوده و کارایی بالایی دارد. به منظور کسب اطلاعات بیشتر در زمینه سیستم نمایش فاکتورادیک، مبنای فاکتوریلی اعداد و ارتباط آن با جایگشت‌ها می‌توانید به مراجع [۲۰، ۲۱ و ۲۲] مراجعه کنید.

Factoradic:	1	1	0
F=	(0,1,2)	(0,1,2)	(0,1,2)
P=	(1,2,3)	(1,3)	(1)
Permutation:	2	3	1
P=	(1,2,3)	(1,3)	(1)
F=	(0,1,2)	(0,1,2)	(0,1,2)
Factoradic:	1	1	0

شکل ۲. نمونه ای از یک تبدیل مبنای فاکتوریلی

۴. تولید جوابهای (جمعیت) اولیه

از آنجا که الگوریتم PSO یک الگوریتم تکاملی است کیفیت جوابهای اولیه بر کیفیت جواب نهایی آن اثرگذار است.

بنابراین، اگر به جای جوابهای تصادفی از یک دسته جوابهای با کیفیت نسبتا خوب به عنوان جمعیت اولیه استفاده شود، در سرعت الگوریتم و کیفیت جواب نهایی آن بهبود حاصل خواهد شد. به منظور بدست آوردن چنین جوابهایی باید از یک الگوریتم سازنده جواب[1] استفاده کرد.

[1] *Constructive Algorithm*

الگوریتم جستجوی تصادفی حریصانه [1](GRASP) یکی از مهمترین و متداولترین الگوریتمهای سازنده جواب است و می‌تواند الگوریتم GRASP می‌تواند به عنوان سازنده جوابهای موجه، جوابهای اولیه الگوریتم PSO را فراهم کند. الگوریتم GRASP یک الگوریتم فرا ابتکاری است و در تقسیم‌بندی کلی جزو دسته الگوریتمهایی به شمار می‌آید که از رهیافت آزمندی[2] برای حل مساله استفاده می‌کنند با این تفاوت که به جای آنکه تنها یک جواب به عنوان بهترین جواب ارایه دهد، با استفاده از یک ساختار تصادفی تعداد زیادی جواب در تکرارهای مختلف تولید می‌کند و از بین آنها بهترین جواب را انتخاب می‌کند.

۴-۱. الگوریتم GRASP برای مساله کار کارگاهی مورد بررسی

در این تحقیق الگوریتم GRASP با توجه به خصوصیات ذکر شده برای آن به عنوان الگوریتم سازنده جمعیت اولیه PSO در نظر گرفته شده است.

برای این منظور یک الگوریتم GRASP جدید توسعه داده شده و از یک تکرار این الگوریتم که در آن به اندازه جمعیت مورد نیاز جواب تولید می‌شود، برای بدست آوردن جمعیت اولیه PSO استفاده شد. الگوریتم GRASP برای مساله مورد بررسی را می‌توان در گامهای ذیل پیاده کرد:

گام ۱: با استفاده از مجموعه‌های $JP_j=\{M_{1j}=i_1, M_{2j}=i_2, ..., M_{nj}=i_m\}$ که مسیر پردازش کار j را نشان می‌دهند و جزو داده‌های مساله هستند، ماتریسی به نام $JobPath$ که سطرهای آن نشان دهنده‌ی کارها و ستونهای آن نشان دهنده‌ی ماشینها است تشکیل دهید. بعبارت دیگر هر مجموعه JP را در یک سطر این ماتریس بگذارید.

گام ۲: کارها را بر اساس قانون ERD[3] مرتب کنید و به هر کاری بر این اساس یک رتبه از ۱ تا n بدهید و این رتبه را با ERD_j مشخص کنید (در صورت تساوی رتبه برابر بگیرند). سپس مجموع زمانهای آماده‌سازی و پردازش هر کار بر روی هر ماشین را بدست آورید و کارها را بر اساس قانون SSPT[4] بر روی ماشینها مرتب کنید و رتبه هر کار بر روی هر ماشین را با $SSPT_{ij}$ نشان دهید (در صورت تساوی رتبه برابر بگیرند). پس از آن مقادیر $LSPT_{ij}$ (برای در نظر گرفتن قانون LSPT[5]) را بصورت $LSPT_{ij}=SSPT_{i(n-j+1)}$ به ازای تمام i و j هایی که $SSPT_{ij}$ برای آنها تعریف شده است، بدست آورید.

گام ۳: مقدار f_{ij} را به ازای $i=1,...,m$, $j=1,...,n$ به صورت زیر بدست آورید (هرچه مقدار f_{ij} کمتر باشد اولویت کار j برای پردازش بر روی ماشین i بیشتر است) :

- اگر $JobPath_{ji}=0$ است مقدار f_{ij} را برابر یک مقدار بسیار بزرگ قرار دهید.
- اگر $JobPath_{ji}\neq 0$ است مقدار f_{ij} را با استفاده از رابطه (۹) محاسبه کنید که در آن پارامترهای w_1 ، w_2 ، w_3 و w_4 مؤلفه‌های وزنی هستند و برای تنظیم اهمیت نسبی چهار عامل استفاده شده در رابطه فوق بکار می‌روند.

$$f(O_{ij})=w_1M_{ij}+w_2SSPT_{ij}+w_3LSPT_{ij}+w_4ERD_j \quad (۹)$$

گام ۴: کارهای ۱ تا n را در فهرستی به نام $A(i)$ بریزید و آن را به ترتیب صعودی مقادیر f_{ij} به ازای تمام کارهای j موجود در آن سطر مرتب کنید. سپس کارهای متناظر با γ درصد اول این مقادیر را در فهرست جدیدی به نام $B(i)$ به عنوان مناسب‌ترین کارهای نامزد برای تخصیص بر روی ماشین i ذخیره کنید. سپس قرار دهید $s:=1$ و $t:=1$.

گام ۵: مقادیر فهرست‌های $A(i)$ و $B(i)$ را عینا در دو فهرست جدید به نامهای $A'(i)$ و $B'(i)$ بریزید. حال قرار دهید $i:=1$ و $p:=1$.

گام ۶: فعالیت p ام از ماشین i را برای افزودن به جواب جزیی S_s، در صورتی که $B'(i)$ هنوز خالی نشده است به صورت تصادفی از فهرست $B'(i)$ انتخاب کنید، در غیر اینصورت این فعالیت را به ترتیب کمترین مقادیر f از فهرست $A'(i)-B'(i)$ انتخاب کنید. فعالیت (کار) مورد نظر را از فهرست $B'(i)$ $\left(A'(i)-B'(i)\right)$ حذف کنید. حال شرایط ذیل را بررسی کنید:

- اگر $i=m$ و $p<n$ بود قرار دهید؛ $i:=1$ و $p:=p+1$ و گام ۶ را تکرار کنید.
- در صورتی که $i<m$ و $p\leq n$ است قرار دهید؛ $i:=i+1$ و گام ۶ را تکرار کنید.
- اگر $i=m$ و $p=n$ بود به گام بعد بروید.

گام ۷: جواب بدست آمده را به عنوان S_s ذخیره نمایید و آن را به یک زمانبندی فعال[6] تبدیل کرده و مقدار تابع هدف آن $C_{\max}(S_s)$ را با مقدار تابع هدف بهترین جواب بدست آمده تا این مرحله $C_{\max}(S^*)$، مقایسه کنید. در صورتی که $C_{\max}(S_s)<C_{\max}(S^*)$ است آن را جایگزین S^* فعلی

1 *Greedy Randomized Adaptive Search Procedure (GRASP)*
2 *Greedy Approach*
3 *Earliest Release Date first*
4 *Shortest Setup and Processing Time first*
5 *Longest Setup and Processing Time first*
6 *Active Schedule*

کنید. سپس در صورتی که $s < N_s$ و $t \le nIter$ باشد با قرار دادن $s := s+1$ به گام ۵ برگردید. که در آن N_s تعداد جوابهای مورد نیاز در هر تکرار است و $nIter$ نیز تعداد تکرارهای مورد نیاز برای اجرای الگوریتم است. در غیر اینصورت اگر $s = N_s$ و $t < nIter$ باشد با قرار دادن $s := 1$ و $t := t+1$ به گام ۵ برگردید و اگر $s = N_s$ و $t = nIter$ بود متوقف شوید.

۴-۲. چرا این الگوریتم جوابهای قانونی و موجه تولید می کند؟

هر جواب الگوریتم GRASP ارایه شده بصورت فهرستی از کارها برای تک تک ماشینها بدست می‌آید. در گام ۵ با اختصاص هر فعالیت (کار) به یک ماشین آن فعالیت (کار) از فهرست $B'(i)$ یا فهرست $A'(i) - B'(i)$ حذف می‌شود. در نتیجه هیچ گاه در یک رشته مربوط به یک ماشین شماره هیچ کاری تکرار نمی‌شود. بنابراین جوابهای بدست آمده همواره قانونی خواهند بود. از طرف دیگر با توجه به اینکه هر یک از این فهرستها بصورت فهرست اولویت کارها بر روی ماشین مربوطه تفسیر می‌شوند جوابهای بدست آمده همواره موجه نیز هستند.

۵. الگوریتم پیشنهادی برای مساله

به دلیل توانایی‌های PSO و نتایج مطلوبی که از خود نشان داده است، در این مقاله از یک الگوریتم PSO برای حل این مساله استفاده شده است. جمعیت اولیه الگوریتم PSO ارایه شده توسط الگوریتم GRASP که در بخش ۴ ارایه شد، تولید می‌شود. به منظور جستجوی بهتر در اطراف هر جواب یافته شده توسط PSO از یک الگوریتم جستجوی محلی (LS)[1] استفاده شده است. برای فرار از دام بهینه‌های محلی از یک رابطه جدید بهنگام سازی سرعت در PSO استفاده شده است. همچنین برای این منظور و نیز برای بهبود نهایی بهترین جواب بدست آمده از الگوریتم GRASP-PSO/LS از یک الگوریتم SA استفاده شده است. قبل از ارایه الگوریتم PSO ترکیبی لازم است توضیحاتی راجع به ساختار همسایگی در این دو الگوریتم داده شود. ساختار همسایگی در این دو الگوریتم بر مبنای یک خصوصیت مهم گراف انفصالی طراحی شده است که در ادامه مفهوم گراف انفصالی به همراه این خصوصیت توضیح داده می‌شود.

۵-۱. گراف انفصالی و ساختار همسایگی

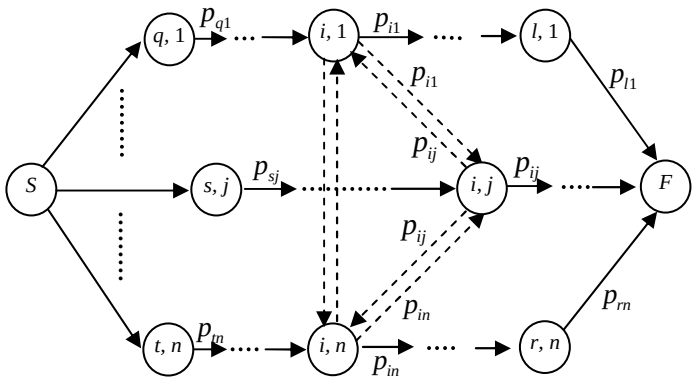
یکی از شیوه‌های متداول نمایش مساله JSSP استفاده از گراف انفصالی[2] است. ایگون بالاش [۲۳] برای نخستین بار مفهوم گراف انفصالی را که ابتدا توسط روی و ساسمان [۲۴] معرفی شده بود، برای نمایش مساله JSSP بکار برد و از آن در یک الگوریتم بهینه‌سازی بر مبنای شمارش ضمنی[3] برای حل مساله $Jm \,||\, C_{\max}$ استفاده کرد. نمونه‌ای از گراف انفصالی سنتی در شکل ۳ نشان داده شده است که در آن فرض شده است که ۳ کار 1، j و n به ماشین یکسان i نیاز دارند. در گراف انفصالی دو نوع گره و دو نوع کمان وجود دارد. دو نوع گره شامل دو گره ابتدایی و انتهایی و سایر گره‌های میانی است.

گره ابتدایی (S) برای شروع کارها و گره انتهایی (F) به منزله پایان کارها است. گره‌های میانی به طور کلی به صورت نشان داده می‌شوند که نشان دهنده فعالیتی از کار j است که بر روی ماشین i انجام می‌شود. سایر گره‌های نشان داده شده که به ماشین i نیاز ندارند نشان دهنده‌ی فعالیت اول یا فعالیت آخر کار متناظر خود هستند. گره‌های متناظر با سایر فعالیتها برای پرهیز از شلوغ شدن شکل نشان داده نشده‌اند.

کمانهای موجود در گراف نیز عبارت‌اند از کمانهای ربطی (ارتباطی)[4] که بصورت خطوط پر (⟶) نشان داده می‌شوند و کمانهای فصلی (انفصالی)[5] که بصورت خط چین (⇢) نشان داده می‌شوند. کمانهای ربطی بین فعالیتهای مختلف یک کار رسم می‌شوند و مسیر تکنولوژیکی پردازش کار را نشان می‌دهند. کمانهای فصلی در دو جهت بین فعالیتهای کارهای مختلف رسم می‌شوند و در واقع فعالیتهایی از این کارها را که به یک ماشین یکسان برای پردازش نیاز دارند، به صورت یک زیر گراف کامل (خوشه[6]) به هم مرتبط می‌کنند. ایجاد هر جواب موجه در مساله به منزله انتخاب یک جهت در هر جفت کمان انفصالی موجود مابین هر دو فعالیتی که به یک ماشین برای پردازش نیاز دارند، است، بگونه‌ای که دوری نیز در زیر گراف مذکور وجود نداشته باشد. بر روی هر کمان ارتباطی یا انفصالی زمان پردازش فعالیتی که این کمان از گره مربوط به آن خارج می‌شود، نوشته می‌شود. زمان پایان تمام کارها ($C_{\max}$) از محاسبه طولانی‌ترین مسیر (مسیر بحرانی) در گراف حاصل از انتخاب یک جهت برای هر کمان انفصالی (جواب موجه) بدست می‌آید.

بالاش [۲۳] قضیه‌ای را به اثبات رسانده بود که اگر جهت یک کمان انفصالی واقع بر مسیر بحرانی یک جواب موجه را برعکس کنیم، جواب حاصل همچنان موجه خواهد ماند. این قضیه اساس تعریف همسایگی در بسیاری از الگوریتمهای جستجوی محلی است که از گراف انفصالی برای پیاده‌سازی الگوریتم کمک می‌گیرند.

[1] *Local Search*
[2] *Disjunctive Graph*
[3] *Implicit Enumeration*
[4] *Conjunctive arcs*
[5] *Disjunctive arcs*
[6] *Clique*



شکل ۳. یک گراف انفصالی در حالت کلی

۵-۲. الگوریتم بهینه‌سازی دسته ذرات ترکیبی (HPSO) برای مساله JSSP

الگوریتم بهینه سازی دسته ذرات ترکیبی که آن را به اختصار HPSO می‌نامیم را می‌توان در گامهای ذیل خلاصه کرد:

گام ۱: مقادیر داده‌ها و پارامترها را بگیرید. فهرستی از اعداد $0,1,2,...,(n-1)$ تهیه کنید و آن را فهرست F بنامید. دو فهرست دیگر از اعداد $1,2,...,n$ برای هر ماشین i تهیه کنید که در آن هر عدد نشان دهنده‌ی یک کار است و این فهرستها را P_i و P_i' بنامید.

گام ۲: با استفاده از الگوریتم GRASP ارایه شده یک مجموعه جواب اولیه موجه با اندازه $popsize$ تولید کنید.

گام ۳: توالی کارها بر روی ماشین i را به عنوان یک جایگشت فرض کرده و با استفاده از تبدیل جایگشت‌ها به مبنای فاکتوریل (f) و سپس تبدیل از مبنای f به مبنای ۱۰ و همچنین با استفاده از فهرست‌های F و P_i'، توالی کارها بر روی هر ماشین در هر جواب از مجموعه جواب اولیه را به یک عدد در مبنای ۱۰ تبدیل کنید و سپس با استفاده از آنها یک بردار m بعدی شامل اعداد صحیح نامنفی ایجاد کنید. این بردار نشان دهنده‌ی موقعیت ذره مورد استفاده در الگوریتم است. پس از آنکه بردارهای m بعدی برای هر جواب r تشکیل شد آن را برابر اولین بردار موقعیت ذره r و بصورت $X_r^1 = \left(x_{r1}^1,...,x_{rm}^1\right),\ \forall r = 1,...,popsize$ قرار دهید.

گام ۴: یک مجموعه بردار m بعدی به عنوان اولین بردار سرعت هر ذره r با اندازه $popsize$ و بصورت $V_r^1 = \left(v_{r1}^1,...,v_{rm}^1\right),\ \forall r = 1,...,popsize$ از اعداد صحیح موجود در بازه $\left[0-x_{ri}^1,(n!-1)-x_{ri}^1\right]$ بصورت تصادفی تولید کنید. سپس به گام ۶ بروید.

گام ۵: i امین بعد از بردار سرعت حرکت هر ذره r را در k امین حرکت آن (k امین تکرار الگوریتم) با استفاده از فرمول ذیل تغییر دهید:

$$v_{ri}^k = \chi\left(wv_{ri}^{k-1} + c_1r_1(p_{ri}^k - x_{ri}^k) + c_2r_2(p_{gi}^k - x_{ri}^k)\right), \quad \forall i = 1,2,...,m\ ,\forall r = 1,...,popsize \tag{۱۰}$$

که در آن k شماره تکرار فعلی الگوریتم است و w ضریب وزن اینرسی بوده و بصورت پویا مقادیر نزولی به خود می‌گیرد که در هر مرحله بصورت زیر مشخص می‌شود:

$$w = w_{\max} - \frac{w_{\max} - w_{\min}}{N_{pso}} \times k \tag{۱۱}$$

$w_{\max}$ و $w_{\min}$ ثابت بوده و توسط کاربر تعیین می‌شوند، N_{pso} تعداد کل تکرارها است و χ ضریب محدود سازی سرعت است که به همان صورت رابطه ۴ تعریف می‌شود. در صورتی که v_{ri}^k در بازه $[0-x_{ri}^k,(n!-1)-x_{ri}^k]$ بود به گام ۶ بروید و در غیر اینصورت اگر مقدار v_{ri}^k کوچکتر از حد پایین بازه بود قرار دهید:

$$v_{ri}^k = rand \times (0 - x_{ri}^k) \tag{۱۲}$$

و اگر بزرگتر از حد بالای بازه بود قرار دهید:

$$v_{ri}^k = rand \times \left((n!-1) - x_{ri}^k\right) \tag{۱۳}$$

که در آن $rand$ یک عدد تصادفی در بازه $[0,1]$ است. سپس به گام ۶ بروید.

گام ۶: با تغییر i امین بعد از بردار موقعیت هر ذره r را با استفاده از فرمول زیر، آن ذره را به موقعیت جدید خود انتقال دهید:

$$x_{ri}^{k+1} = \left\lfloor x_{ri}^k + v_{ri}^k \right\rfloor, \quad \forall i = 1,2,...,m\ ,\forall r = 1,...,popsize \tag{۱۴}$$

که در آن k شمارنده تکرار است. با موقعیت جدید X_r^{k+1} برای هر ذره r به گام بعد بروید.

گام ۷: ابتدا کلیه مقادیر فهرست P_i را عینا در فهرست P_i' بریزید. موقعیت فعلی هر یک از ذرات جمعیت که بصورت $X_r^{k+1}=\left(x_{r1}^{k+1},...,x_{rm}^{k+1}\right),\ \forall r=1,...,popsize$ نشان داده می‌شود را در نظر بگیرید. i امین بعد آن که یک عدد در مبنای ۱۰ است را به یک عدد در مبنای فاکتوریلی تبدیل کنید ($i=1,2,...,m$). حال عدد بدست آمده در مبنای فاکتوریلی را با توجه به فهرست‌های F و P_i' به یک جایگشت تبدیل کنید (که نشان دهنده‌ی توالی کارها بر روی ماشین i نیز است). سپس با این جایگشت‌ها برای هر ماشین i جواب متناظر با هر ذره r را تشکیل دهید. این جواب که بصورت فهرست اولویت کارها (جایگشت) بدست آمده است را به یک زمانبندی فعال تبدیل کنید و مقدار تابع هدف آن ($C_{\max}$) را محاسبه کنید. سپس هر زمانبندی فعال بدست آمده را در دو حافظه S_r و $S_{r_{temp}}$ ذخیره کنید و به گام ۸ بروید.

گام ۸ (گام جستجوی محلی؛ پرقدرت سازی و متنوع سازی): در صورتی که $C_{\max}(S_{r_{temp}})\le C_{\max}(S_r)$ باشد (مساوی به منظور متنوع سازی) قرار دهید $S_r \leftarrow S_{r_{temp}}$. در صورتی که N_{ls} تکرار از اجرای گام ۸ گذشته است، با جواب S_r و مقدار تابع هدف $C_{\max}(S_r)$ برای آن از گام ۸ خارج شوید. در غیر اینصورت مسیر بحرانی زمانبندی فعال $S_{r_{temp}}$ را بیابید. کمانهای انفصالی موجود بر مسیر بحرانی را مشخص کنید و از بین آنها یکی را به تصادف انتخاب کنید و با استفاده از تعویض زوجی دو کار همسایه در توالی کارهای زیر رشته (ماشین) مربوطه، جهت آن کمان انفصالی را معکوس کنید. در صورتی که هیچ کمان انفصالی‌ای بر روی مسیر بحرانی قرار نداشت، یک کمان انفصالی را به تصادف انتخاب کنید و جهت آن را معکوس کنید. درصورتی که کمانهای انفصالی بر روی مسیر بحرانی وجود دارند اما تعداد آنها کمتر از ۳ است یک کمان واقع بر مسیر بحرانی و یک کمان غیر واقع بر مسیر بحرانی را به تصادف انتخاب کرده و جهت آنها را بر عکس کنید. مقدار تابع هدف جواب جدید بدست آمده را محاسبه کرده و آن را به زمانبندی فعال جدید $S_{r_{temp}}$ تبدیل کنید و گام ۸ را تکرار نمایید.

گام ۹: موقعیت هر ذره را با توجه به زمانبندی فعال جدید بدست آمده و با استفاده از تبدیل مبنای فاکتوریلی بهنگام کنید.

گام ۱۰: اگر مقدار فعلی تابع هدف متناظر با موقعیت فعلی هر ذره r کمتر یا مساوی بهترین مقدار قبلی آن ذره ($pbest[r]$) بود، مقدار فعلی تابع هدف را جایگزین $pbest[r]$ نموده و موقعیت فعلی ذره را جایگزین بهترین موقعیت قبلی آن یعنی $P_r=\left(p_{r1},...,p_{rm}\right)$ کنید. همچنین اگر مقدار فعلی تابع هدف متناظر با موقعیت فعلی هر ذره r کمتر یا مساوی بهترین مقدار قبلی بدست آمده توسط کلیه ذرات ($pbest[gbest]$) بود، قرار دهید: $gbest=r$ و موقعیت فعلی ذره را جایگزین بهترین موقعیت قبلی بدست آمده توسط کلیه ذرات یعنی $P_g=\left(p_{g1},...,p_{gm}\right)$ کنید.

گام ۱۱: شروط توقف زیر را بررسی کنید.

- **شرط توقف ۱:** تعداد تکرارهای الگوریتم به اندازه از پیش تعیین شده‌ی N_{pso} برسد ($k>N_{pso}$ شود).
- **شرط توقف ۲:** میزان بهبود تابع هدف در N'_{pso} تکرار متوالی از مقدار از پیش تعیین شده‌ی ε کمتر باشد.
- **شرط توقف ۳**: از زمان شروع اجرای الگوریتم مدت زمان τ گذشته باشد.

در صورتی که هیچ یک از شروط توقف زیر برقرار نشده است، با قرار دادن $k:=k+1$ به گام ۵ برگردید. در غیر اینصورت زمانبندی فعال متناظر با $P_g=\left(p_{g1},...,p_{gm}\right)$ که پیشتر بدست آمده است را به عنوان بهترین جواب الگوریتم GRASP-PSO/LS و با نام S' شناخته ، $C_{\max}(S')$ را محاسبه و آن را به گام ۱۲ منتقل کنید.

گام ۱۲ (گام پرقدرت سازی): با استفاده از الگوریتم جستجوی محلی ذکر شده در گام ۸ سعی کنید جواب S' را به جواب بهتری تبدیل کنید. این کار را به تعداد N_{pow} تکرار کنید. سپس قرار دهید $S''\leftarrow S'$ و $S^*\leftarrow S'$ و $T\leftarrow T_0$ و به گام ۱۳ بروید.

گام ۱۳ (الگوریتم آنیلینگ شبیه سازی شده): با استفاده از الگوریتم جستجوی محلی که در گام ۸ توضیح داده شد جواب S'' را به جواب جدید S''_{new} تبدیل کنید و مقدار $\Delta=C_{\max}(S''_{new})-C_{\max}(S'')$ را محاسبه کنید. قرار دهید $S''\leftarrow S''_{new}$. اگر $\Delta\le 0$ بود قرار دهید $S'\leftarrow S''$ و $C_{\max}(S')<C_{\max}(S^*)$ $C_{\max}(S')\leftarrow C_{\max}(S'')$. در این حالت اگر بود قرار دهید $C_{\max}(S^*)\leftarrow C_{\max}(S')$ و $S^*\leftarrow S'$. اگر $\Delta>0$ بود، یک عدد تصادفی در بازه [0,1] تولید کنید و آن را r بنامید، آنگاه اگر $r\le e^{\left(-\frac{\Delta}{T}\right)}$ بود، قرار دهید $S'\leftarrow S''$ و $C_{\max}(S')\leftarrow C_{\max}(S'')$ و در غیر اینصورت قرار دهید

$S'' \leftarrow S'$ و $C_{\max}(S'') \leftarrow C_{\max}(S')$. در صورتی که L تکرار از اجرای این گام گذشته است متوقف شوید و با قرار دادن $T \leftarrow \alpha T$ به گام ۱۴ بروید که در آن $0 < \alpha < 1$ و ضریب خنک سازی است. در غیر اینصورت گام ۱۳ را تکرار کنید.

گام ۱۴: مادامی که دمای فعلی T بزرگتر از یا مساوی با دمای پایانی T_f است به گام ۱۳ برگردید، در غیر اینصورت متوقف شوید و S^* را با مقدار تابع هدف $C_{\max}(S^*)$ به عنوان بهترین جواب الگوریتم HPSO بگیرید.

در این الگوریتم از یک رابطه جدید بهنگام سازی سرعت و یک شیوه جدید برای تبدیل جوابهای JSSP به جوابهای مورد پذیرش PSO استفاده شده است. همچنین یک الگوریتم LS برای جستجوی بهتر در اطراف جوابهای بدست آمده توسط PSO در داخل آن تعبیه شده است. یک الگوریتم SA نیز برای بهبود جواب نهایی بدست آمده توسط GRASP-PSO/LS بکار گرفته شد.

یک شبه کد برای الگوریتم در شکل ۴ ارایه شده است. همچنین فلوچارت مراحل اجرای الگوریتم با تکیه بر جزئیات الگوریتم PSO در شکل ۵ ارایه شده است.

HPSO Pseudo Code

Begin
STEP 1: Get all data and parameter values.
STEP 2: Generate initial solutions with GRASP algorithm.
STEP 3: For each generated solution convert job order on each machine i via factorial base transformation into a number in base of 10 and create first position vector ($X_r^1 = (x_{r1}^1, ..., x_{rm}^1)$) for each particle r.
STEP 4: Generate first velocity vectors. For particle r select elements of vector $V_r^1 = (v_{r1}^1, ..., v_{rm}^1)$ randomly from interval $[0 - x_{ri}^1, (n!-1) - x_{ri}^1]$. Then go to STEP 6.
STEP 5: Update V_r for particle r in iteration k with $v_{ri}^k := \chi(wv_{ri}^{k-1} + c_1 r_1(p_{ri}^k - x_{ri}^k) + c_2 r_2(p_{gi}^k - x_{ri}^k))$, $\forall i$ such that v_{ri} be in the interval $[0 - x_{ri}^k, (n!-1) - x_{ri}^k]$.
If $v_{ri}^k < (0 - x_{ri}^k)$
$v_{ri}^k := rand \times (0 - x_{ri}^k)$;
If $v_{ri}^k < ((n!-1) - x_{ri}^k)$
$v_{ri}^k := rand \times ((n!-1) - x_{ri}^k)$;
Then go to STEP 6.
STEP 6: Update X_r for particle r in iteration k with $x_{ri}^{k+1} := \lfloor x_{ri}^k + v_{ri}^k \rfloor, \forall i$.
STEP 7: Convert X_r^{k+1} to a solution with preference list-based representation via factorial base transformation and then transform it an active schedule and Calculate $C_{\max}$ of it. Then Save r^{th} active schedule in S_r and $S_{r_{temp}}$.
STEP 8 (Local Search; Intensification and Diversification):
For N_{ls} iteration of this step do:
Reverse a randomly selected disjunctive arc that located on the critical path of the disjunctive graph corresponded to $S_{r_{temp}}$;
Transform obtained $S_{r_{temp}}$ to an active schedule and calculate $C_{\max}(S_{r_{temp}})$;
If $C_{\max}(S_{r_{temp}}) \leq C_{\max}(S_r)$
$S_r \leftarrow S_{r_{temp}}$;
End if
End for
STEP 9: Update $X_r^{k+1}, \forall r$ according to S_r via factorial base transformation.
STEP 10: Update P_r for each particle r. Also update P_g for all particles.
STEP 11: Check the termination conditions.
If conditions is satisfied
Get the active schedule corresponded to P_g, call it S' and calculate $C_{\max}(S')$. Then return S' and $C_{\max}(S')$;
Else go back to STEP 5
STEP 12: Intensify S' with procedure of STEP 8 and repeat it N_{pow} times. Then do $S'' \leftarrow S'$, $S^* \leftarrow S'$, $T \leftarrow T_0$.
STEP 13 (SA procedure):
While (repetition number of this procedure is not greater than L)
Transform S' to S''_{new} with local search of STEP 8
$\Delta := C_{\max}(S''_{new}) - C_{\max}(S')$
$S' \leftarrow S''_{new}$
IF $\Delta \leq 0$
$S' \leftarrow S'$
IF $C_{\max}(S') < C_{\max}(S^*)$
$S^* \leftarrow S'$
End if
Else generate r a random number in $[0,1]$
If $r \leq e^{\left(-\frac{\Delta}{T}\right)}$
$S' \leftarrow S''$;
Else $S'' \leftarrow S'$;
End else
End while
$T \leftarrow \alpha T, 0 < \alpha < 1$;
STEP 14: While ($T \geq T_f$)
Go back to STEP 13;
End while
Return S^* as the best obtained solution by HPSO
End

شکل۴. شبه کد برای الگوریتم HPSO

۶. نتایج محاسباتی

به منظور آزمایش عملکرد الگوریتم ارایه شده آن را بر روی مسایل نمونه مشهور موجود در ادبیات مساله JSSP اجرا کرده و نتایج را با سایر الگوریتـمها مقایسه نموده‌ایم. مسایل نمونه اســتفاده شده عبـارت‌اند از: FT06، FT10، FT20 [۲۵] که گاهـی با عنوان MT نیز اسـتفاده می‌شوند، ORB01 تا ORB10 [۲۶]، LA01 تا LA40 [۲۷]، ABZ5 تا ABZ9 [۴] و YN1 تا YN4 [۲۸]. مجموعا ۶۲ مساله انتخاب شده است که تمامی آنها در تارنمای کتابخانه الکترونیکی تحقیق در عملیات ارایه شده توسط بیزلی [۲۹] به آدرس اینترنتی URL:http://people.brunel.ac.uk/~mastjjb/jeb/info.html موجود است. الگوریتم پیشنهادی (HPSO) در اینجا مورد آزمایش قرار گرفته و عملکرد آن در مقایسه با بهترین جوابهای شناخته شده موجود در ادبیات (BKS)[1] و همچنین برخی از بهترین الگوریتمهای ارایه شده برای مساله کار کارگاهی سنجیده شده است. این الگوریتم با استفاده از زبان C کد شده و در محیط Visual C++ و برای حل مسایل نمونه بر روی یک دستگاه رایانه با 512 MB RAM و 2.4 GHz Intel Celeron CPU اجرا شده است.

۶-۱. تنظیم پارامترها

به منظور تنظیم پارامترهای الگوریتم، تعدادی از مسایل از هر یک از این ۵ دسته مساله به عنوان نمونه انتخاب شد و با استفاده از تجزیه تحلیل حساسیت پارامترها بگونه‌ای تنظیم شدند که بهترین جواب را بدست دهند. پارامترهایی که برای این مسایل بهترین جواب را ارایه می‌دادند بر روی دسته مسـایل نظیر آنها به کار گرفـته شد. این مســایل عبـارت‌اند از: FT10، ABZ 7&9، LA 02&15&21&29&39، ORB 2&4 و YN 2&4 . بهترین نتایج بدست آمده برای تنظیم پارامترها در جدول ۲ نشان داده شده است. از آنجا که در الگوریتم HPSO از فرایند SA به منظور جستجوی دقیقتر در پیرامون بهترین جواب یافته شده توسط فرایند PSO استفاده می‌شود دمای اولیه T_0 نسبتا کوچک انتخاب شده است.

۶-۲. نتایج بدست آمده

همانطور که پیشتر گفته شد، در الگوریتم HPSO از یک تکرار یک الگوریتم GRASP که بر اساس یک ساختار آزمندانه تصادفی عمل می‌کند، استفاده شده است. استفاده از این الگوریتم باعث می‌شودکه علاوه بر اینکه جوابهای اولیه الگوریتم HPSO کیفیت خوبی داشته باشند، از پراکندگی کافی که لازمه جستجوی سریعتر و گسترده‌تر فضای جواب است نیز برخوردار باشند. از طرف دیگر این الگوریتم بسیار سریع است و تفاوت زمانی چندانی با تولید تصادفی جوابها ندارد. در مورد سرعت الگوریتم GRASP ارایه شده می‌توان گفت که بطور متوسط تقریبا ۱۰۰۰ جواب را به همراه تبدیل آنها به زمانبندی‌های فعال و محاسبه تابع هدف آنها می‌توان توسط آن و بر روی رایانه مورد استفاده در این تحقیق در زمانی نزدیک به یک ثانیه تولید کرد. نتایج حاصل از مقایسه کیفیت جوابهای تولیدی از الگوریتم GRASP با کیفیت جوابهای تولیدی بصورت تصادفی برای تعدادی از مسایل نمونه ساده و سخت، در جدول ۳ نشان داده شده است. هر کدام از مسایل ۵ بار حل شده است و فاصله بین میانگین آنها با BKS بر حسب درصد محاسبه شده است.

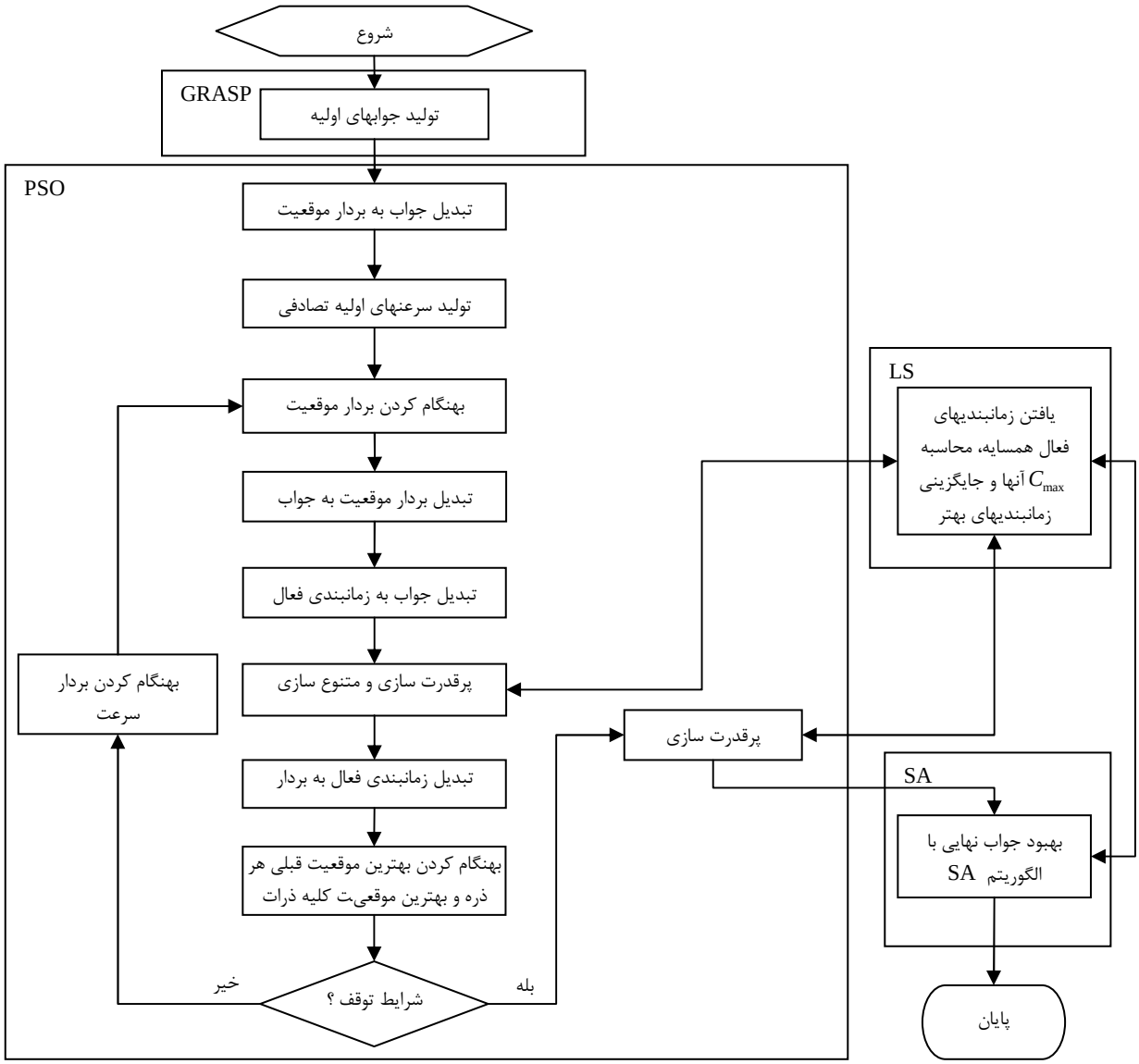
همانطور که مشاهده می‌کنید کیفیت جواب‌های تولیدی الگوریتم GRASP تقریبا دو برابر بهتر از جوابهای تصادفی است. مقایسه نتایج حاصل از GRASP و جوابهای تصادفی در شکل ۴ نشان داده شده است.

معیار مقایسه اصلی در این مساله BKS است. به علت قدمت مساله و حجم عظیم مطالعات صورت گرفته بر روی آن ، جواب بهینه بسیاری از این مسایل علیرغم پیچیدگی بالای آنها پیدا شده است که این موارد بصورت BKS (Opt.) در جداول ارایه شده است. در مواردی که جواب بهینه پیدا نشده است بهترین حد، پایین شناخته شده نیز بصورت BKLB[2] در جداول ارایه شده است. الگوریتم‌های انتخاب شده برای مقایسه بگونه‌ای انتخاب شده‌اند که از سالهای مختلف باشند تا روند بهبود مطالعات صورت گرفته بر این مساله را نشان دهد.

این الگوریتم‌ها هر یک در زمان خود جزو بهترین الگوریتمها بوده‌اند. این الگوریتمها عبارت‌اند از : الگوریتم انتقال گلوگاه [۴]، الگوریتم ترکیب شده ژنتیک با الگوریتم گیفلر و تامپسون [۲۸]، الگوریتم ترکیب شده جستجوی ممنوع با انتقال گلوگاه [۵]، الگوریتم ژنتیک ترکیب شده با جستجوی محلی با استفاده از فضای جوابهای فعال پارامتری [۸]، الگوریتم ترکیب شده بهینه سازی دسته ذرات با الگوریتم گیفلر و تامپسون [۱۵]، الگوریتم ترکیب شده بهینه سازی دسته ذرات با الگوریتم آنیلینگ شبیه سازی شده [۱۴] و الگوریتم بهینه سازی اجتماع مورچگان [۱۳]. از آنجا که هر یک از این الگوریتمها برای حل همه ۶۲ مساله مورد بررسی استفاده نشده‌اند نتایج مربوط به دسته مسایل مختلف در جداول مختلف ارایه شده است. برای هر الگوریتم مقدار تابع هدف گزارش شده و فاصله آن با بهترین جواب در دست بر حسب درصد ارایه شده است. در پایین همه جداول نیز برای هر الگوریتم میانگین این اختلاف‌ها به همراه تعداد مسایل نمونه حل شده و تعداد مسایل نمونه‌ای که مقدار BKS برای آنها توسط آن الگوریتم بدست آمده است، ارایه شده است.

[1] *Best Known Solution*

[2] *Best Known Lower Bound*



شکل۵. فلوچارت الگوریتم HPSO

جدول ۲. بهترین مقادیر بدست آمده برای پارامترهای الگوریتم HPSO

Parameters	FT 6 & 10 & 20	ABZ 6-9	ORB 1-10	YN 1-4	LA 1-10	LA 10-20	LA 21-30 & 36-40	LA 31-35
popsize	20	30	35	40	20	25	35	25
γ	65%	65%	65%	65%	65%	65%	65%	65%
w_1	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3
w_2	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
w_3	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
w_4	0	0	0	0	0	0	0	0
w_{max}	1	1	1	1	1	1	1	1
w_{min}	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
c_1	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1
c_2	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1
N_{ls}	50	50	50	150	50	50	100	50

ادامه جدول۲. بهترین مقادیر بدست آمده برای پارامترهای الگوریتم HPSO

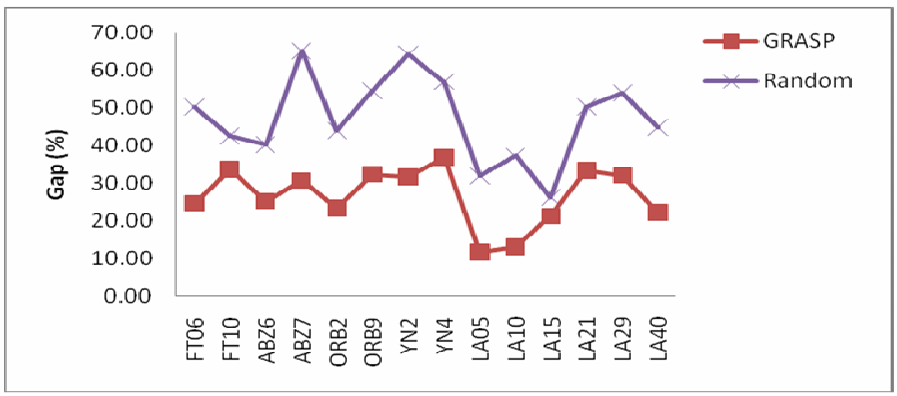
Parameters	FT 6 & 10 & 20	ABZ 6-9	ORB 1-10	YN 1-4	LA 1-10	LA 10-20	LA 21-30 & 36-40	LA 31-35
N_{pso}	220	185	212	246	173	205	254	215
N'_{pso}	75	75	75	100	75	75	100	75
τ (sec.)	10000	10000	10000	10000	10000	10000	10000	10000
N_{pow}	100	100	100	200	100	100	150	100
T_0	5	7	6	15	5	5	10	8
T_f	0.1	0.1	0.05	0.05	0.5	0.2	0.05	0.1
α	0.983	0.985	0.985	0.99	0.983	0.983	0.983	0.983
L	100	120	140	200	100	100	160	110

از آنجا که الگوریتم ترکیب شده بهینه سازی دسته ذرات با الگوریتم آنیلینگ شبیه سازی شده [۱۴] از نظر ساختاری نزدیکترین الگوریتم به HPSO است، برای این الگوریتم علاوه بر مقدار تابع هدف زمان حل گزارش شده نیز در جداول آورده شده است. این الگوریتم بر روی یک رایانه با مشخصات Intel Celeron 300 و 128 MB RAM اجرا شده بود.

با استفاده از الگوریتم HPSO هر مساله ۱۰ بار حل شده است و بهترین مقدار بدست آمده از تابع هدف در جداول مربوطه نوشته شده است. علاوه بر آن زمان حل کل الگوریتم برای رسیدن به بهترین جواب ارایه شده در هر مساله بر حسب ثانیه و همچنین زمانی که الگوریتم صرف فرایند آنیلینگ شبیه‌سازی شده کرده است نیز در جداول ذکر شده‌اند.

جدول۳. مقایسه نتایج حاصل از الگوریتم GRASP و جوابهای تصادفی

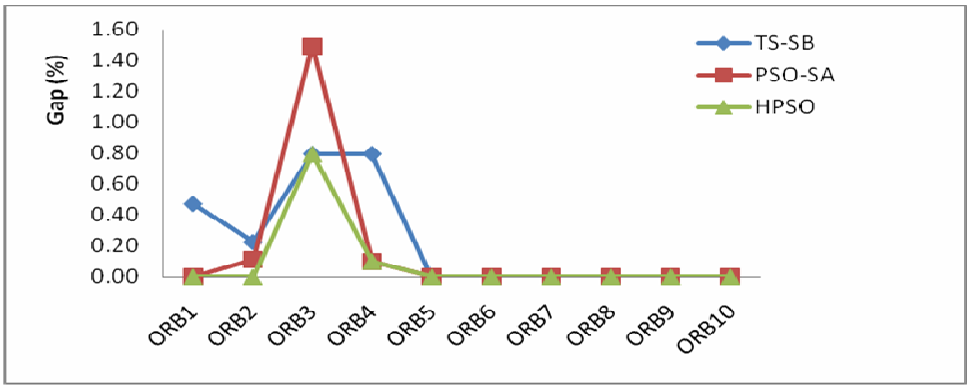
Problem	*n*	*m*	BKS	GRASP		Random	
				Average	Gap(%)	Average	Gap(%)
FT06	6	6	55	68.6	24.73	82.6	50.18
FT10	10	10	930	1244	33.76	1326.2	42.60
ABZ6	10	10	943	1181.2	25.26	1322.4	40.23
ABZ7	20	15	656	857.8	30.76	1083.4	65.15
ORB2	10	10	888	1096.6	23.49	1278.8	44.01
ORB9	10	10	934	1236.6	32.40	1443.4	54.54
YN2	20	20	909	1198	31.79	1493.6	64.31
YN4	20	20	970	1328.2	36.93	1522.6	56.97
LA05	10	5	593	662.2	11.67	782.8	32.01
LA10	15	5	958	1083.8	13.13	1315.2	37.29
LA15	20	5	1207	1464.6	21.34	1523.6	26.23
LA21	15	10	1046	1395	33.37	1571.6	50.25
LA29	20	10	1157	1529.4	32.19	1781.4	53.97
LA40	15	15	1222	1495.4	22.37	1769	44.76



شکل۶. مقایسه نتایج حاصل از الگوریتم GRASP و جوابهای تصادفی

جدول۴. نتایج بدست آمده برای مسایل ORB

Prob.	Size	BKS (Opt.)	TS-SB		Particle Swarm Optimization								
					PSO-SA			HPSO					
			Pezzella & Merelli (2000)		Xia&Wu (2006)			Cmax	Time	Time of SA	PI (%)	Best Gap (%)	Average Gap (%)
	$n \times m$		Cmax	Gap (%)	Cmax	Time	Gap (%)						
ORB1	10 × 10	1059	1064	0.47	1059	334	0.00	1059	43	11	23	0.00	1.28
ORB2	10 × 10	888	890	0.23	889	182	0.11	888	24	2	43	0.00	2.30
ORB3	10 × 10	1005	1013	0.80	1020	297	1.49	1013	49	16	64	0.80	3.20
ORB4	10 × 10	1005	1013	0.80	1006	141	0.10	1006	42	5	31	0.10	1.42
ORB5	10 × 10	887	887	0.00	887	404	0.00	887	37	8	52	0.00	0.45
ORB6	10 × 10	1010	-		1010	337	0.00	1010	51	3	15	0.00	0.31
ORB7	10 × 10	397	-		397	336	0.00	397	73	18	73	0.00	0.22
ORB8	10 × 10	899	-		899	340	0.00	899	56	9	12	0.00	0.56
ORB9	10 × 10	934	-		934	365	0.00	934	68	16	37	0.00	0.37
ORB10	10 × 10	944	-		944	351	0.00	944	79	13	26	0.00	0.42
Average Gap (%)				0.4579			0.1705					0.0896	1.0530
No. of Instance				5			10				10		
No. of BKS Obtained				1			7				8		



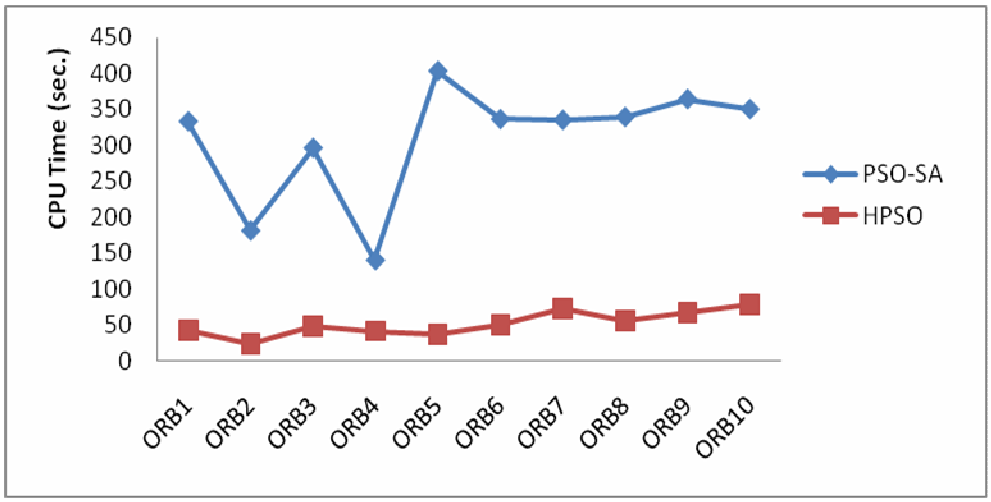
شکل۷. مقایسه اختلاف مقادیر بدست آمده سه الگوریتم TS-SB، PSO-SA و HPSO با مقدار بهینه برای مسایل ORB

درصد تکرارهای لازم برای فرایند PSO برای اینکه به بهترین جواب خود برسد در ستون PI[1] آورده شده است. فاصله بین بهترین مقدار بدست آمده توسط HPSO و مقدار BKS و همچنین فاصله بین میانگین مقادیر بدست آمده توسط HPSO در ۱۰ بار اجرا و مقدار BKS در دو ستون انتهایی جداول آمده است. نتایج بدست آمده برای دسته مسایل ORB در جدول ۴ نشان داده شده است. این نتایج در شکل ۵ مقایسه شده‌اند. مقایسه بین زمان حل مسایل ORB با دو الگوریتم PSO-SA و HPSO نیز در شکل ۶ آمده است. نتایج بدست آمده برای مسایل YN در جدول ۵ آورده شده است و مقایسه اختلاف مقادیر حاصل از دو الگوریتم GA-G&T و HPSO با مقادیر BKS برای این مسایل در شکل ۷ ارایه شده است. نتایج مربوط به دسته مسایل FT و ABZ در جدول۶ نشان داده است.مقایسه اختلاف مقادیر حاصل از چهار الگوریتم TS-SB، PSO-SA، SB و HPSO با مقادیر BKS برای مسایل FT و ABZ نیز در شکل ۸ نشان داده شده است. عملکرد الگوریتم HPSO و سایر الگوریتم‌ها برای دسته مسایل LA در جدول ۷ آورده شده است. نتایج بدست آمده نشان می‌دهند که الگوریتم HPSO قادر است که مقدار BKS را برای درصد زیادی از مسایل نمونه بیابد و برای سایر مسایل نیز از درصد اختلاف ناچیزی برخوردار است. این درصد اختلاف در مقایسه با سایر الگوریتم‌ها بسیار مناسب به نظر می‌رسد. درصد اختلاف میانگین جوابهای بدست آمده در ۱۰ بار اجرای الگوریتم نیز بسیار کم است و نشان می‌دهد که کیفیت جوابهای بدست آمده در مجموع نیز عالی است. زمان حل مسایل توسط الگوریتم HPSO نسبت به الگوریتم PSO-SA [۱۴] در

[1] *Percent of needed Iterations in PSO procedure to reach to the best solution of PSO*

بیشتر مسایل با اختلاف فاحشی کمتر است در حالی که کیفیت جوابهای بدست آمده نسبت به آن بهتر بوده است.

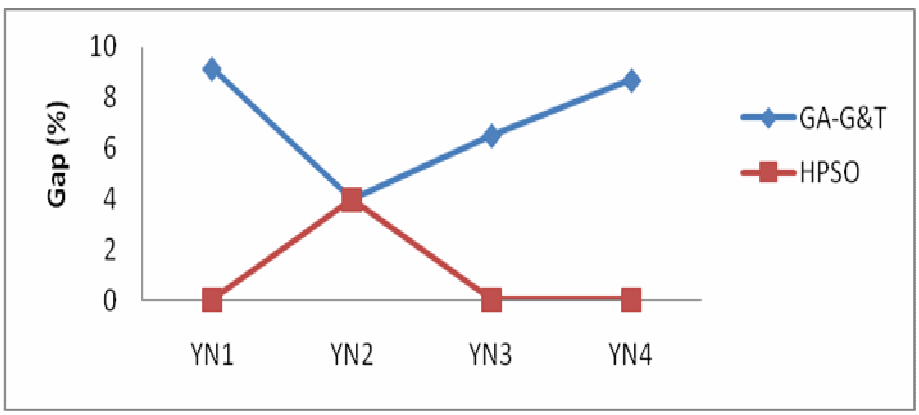
زمانی که الگوریتم HPSO صرف فرایند SA می‌کند بسیار کمتر از زمانی است که صرف فرایند PSO می‌کند و این نشان می‌دهد که فرایند PSO قادر است به خوبی فضای جوابها را مورد جستجو قرار دهد. بهترین جوابی که فرایند PSO برای هر مساله بدست آورده است در اغلب موارد در تعداد تکراری کمتر از تکرار توقف آن بدست آمده است و این نشان می‌دهد که با تنظیم دقیق پارامترها برای هر مساله بجای یک دسته مساله می‌توان زمان حل را از این مقدار نیز کمتر کرد.



شکل۸. مقایسه زمان حل مسایل ORB بوسیله دو الگوریتم PSO-SA و HPS

جدول ۵. نتایج بدست آمده برای مسایل نمونه YN

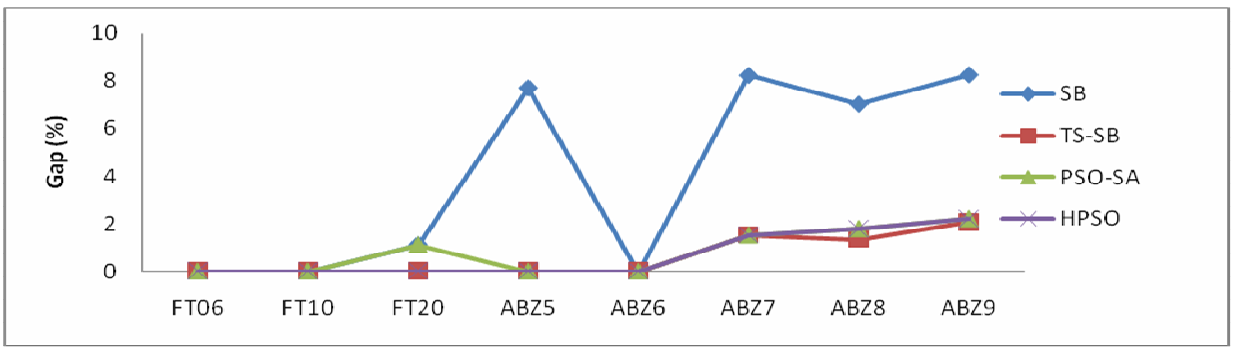
Prob.	Size	BKLB	BKS	GA-G&T		HPSO					
				Yamada & Nakano (1992)		Cmax	Time	Time of SA	PI (%)	Best GAP (%)	Average GAP (%)
	n × *m*			Cmax	GAP (%)						
YN1	20 × 20	826	886	967	9.14	886	123	45	65	0.00	2.15
YN2	20 × 20	861	909	945	3.96	945	117	51	58	3.96	4.36
YN3	20 × 20	827	893	951	6.49	893	151	57	72	0.00	1.25
YN4	20 × 20	918	968	1052	8.68	968	148	43	48	0.00	4.72
Average GAP (%)					7.0688					0.9901	3.1200
No. of Instance					4					4	
No. of BKS Obtained					0					3	



شکل۹. مقایسه اختلاف مقادیر حاصل از دو الگوریتم GA-G&T و HPSO با مقادیر BKS برای مسایل YN

جدول ۶. نتایج بدست آمده برای مسایل نمونه FT و ABZ

Prob.	Size			BKLB	BKS	Shifting bottleneck (SB)			TS-SB		HGA-Param		Particle Swarm Optimization									
						Adams et al. (1988)			Pezzella & Merelli (2000)		Gonçalves et al. (2005)		PSO-G&T Sha&Hsu (2006)		PSO-SA Xia&Wu (2006)			HPSO					
	n	×	m			Cmax SBI	Cmax SBII	Gap(%) SBII	Cmax	Gap (%)	Cmax	Gap (%)	Cmax	Gap (%)	Cmax	Time	Gap (%)	Cmax	Time	Time of SA	PI (%)	Best Gap (%)	Average Gap (%)
FT06	6	×	6		55	55	55	0.00	55	0.00	55	0.00	55	0.00	55	1	0.00	55	1	0	5	0.00	0.00
FT10	10	×	10		930	1015	930	0.00	930	0.00	930	0.00	930	0.00	930	142	0.00	930	28	4	57	0.00	2.46
FT20	20	×	5		1165	1290	1178	1.12	1165	0.00	1165	0.00	1165	0.00	1178	21	1.12	1165	25	3	43	0.00	2.61
ABZ5	10	×	10		1234	1306	1329	7.70	1234	0.00	-		-		1234	330	0.00	1234	41	8	14	0.00	0.74
ABZ6	10	×	10		943	962	943	0.00	943	0.00	-		-		943	158	0.00	943	31	5	23	0.00	0.53
ABZ7	20	×	15		656	730	710	8.23	666	1.52	-		-		666	4332	1.52	666	121	35	46	1.52	4.23
ABZ8	20	×	15	645	669	774	716	7.03	678	1.35	-		-		681	4356	1.79	681	105	24	74	1.79	3.64
ABZ9	20	×	15	656	679	751	735	8.25	693	2.06	-		-		694	4374	2.21	694	117	18	63	2.21	3.57
Average GAP (%)								4.0399		0.6164		0.0000		0.0000			0.8304					0.6909	2.2225
No. of Instance								8		8		3		3			8					8	
No. of BKS Obtained								3		5		3		3			4					5	



شکل ۱۰. مقایسه اختلاف مقادیر حاصل از چهار الگوریتم SB، TS-SB، PSO-SA و HPSO با مقادیر BKS برای مسایل FT و ABZ

جدول ۷. نتایج بدست آمده برای مسایل نمونه LA

Prob.	Size	BKS (Opt.)	Shifting bottleneck (SB)			TS-SB		HGA-Param		ACO Best		Particle Swarm Optimization										
			Adams et al. (1988)			Pezzella & Merelli (2000)		Gonçalves et al. (2005)		Rossi & Dini (2007)		PSO-G&T Sha & Hsu (2006)		PSO-SA Xia&Wu (2006)			HPSO					
	n × m		Cmax SBI	Cmax SBII	Gap(%) SBII	Cmax	Gap (%)	Cmax	Gap (%)	Cmax	Gap (%)	Cmax	Gap (%)	Cmax	Time	Gap (%)	Cmax	Time	Time of SA	PI (%)	Best GAP (%)	Average GAP (%)
LA01	10 × 5	666	666	666	0	666	0	666	0	666	0	666	0	666	2	0	666	3	0	5	0	0
LA02	10 × 5	655	720	669	2.14	655	0	655	0	665	1.53	655	0	655	3	0	655	3	0	47	0	0.45
LA03	10 × 5	597	623	605	1.34	597	0	597	0	597	0	597	0	597	5	0	597	4	0	13	0	0.25
LA04	10 × 5	590	597	593	0.51	590	0	590	0	598	1.36	590	0	590	3	0	590	5	0	15	0	0.1
LA05	10 × 5	593	593	593	0	593	0	593	0	593	0	593	0	593	2	0	593	1	0	11	0	0
LA06	15 × 5	926	926	926	0	926	0	926	0	926	0	926	0	926	5	0	926	2	0	19	0	0.24
LA07	15 × 5	890	890	890	0	890	0	890	0	890	0	890	0	890	5	0	890	5	0	12	0	0
LA08	15 × 5	863	868	863	0	863	0	863	0	863	0	863	0	863	5	0	863	9	0	15	0	0
LA09	15 × 5	951	951	951	0	951	0	951	0	951	0	951	0	951	5	0	951	2	0	23	0	0
LA10	15 × 5	958	959	959	0.1	958	0	958	0	958	0	958	0	958	1	0	958	1	0	25	0	0
LA11	20 × 5	1222	1222	1222	0	1222	0	1222	0	1222	0	1222	0	1222	4	0	1222	6	0	21	0	0.32
LA12	20 × 5	1039	1039	1039	0	1039	0	1039	0	1039	0	1039	0	1039	12	0	1039	7	0	32	0	0

ادامه جدول ۷. نتایج بدست آمده برای مسایل نمونه LA

Prob.	Size	BKS (Opt.)	Shifting bottleneck (SB)			TS-SB		HGA-Param		ACO Best		Particle Swarm Optimization									
												PSO-G&T		PSO-SA			HPSO				
			Adams et al. (1988)			Pezzella & Merelli (2000)		Gonçalves et al. (2005)		Rossi & Dini (2007)		Sha & Hsu (2006)		Xia&Wu (2006)			Cmax	Time	Time of SA	PI (%)	Best GAP (%)	Average GAP (%)
	$n \times m$		Cmax SBI	Cmax SBII	Gap(%) SBII	Cmax	Gap (%)	Cmax	Gap (%)	Cmax	Gap (%)	Cmax	Gap (%)	Cmax	Time	Gap (%)						
LA13	20 × 5	1150	1150	1150	0	1150	0	1150	0	1150	0	1150	0	1150	4	0	1150	5	0	22	0	0.73
LA14	20 × 5	1292	1292	1292	0	1292	0	1292	0	1292	0	1292	0	1292	2	0	1292	4	0	41	0	0.38
LA15	20 × 5	1207	1207	1207	0	1207	0	1207	0	1212	0.41	1207	0	1207	11	0	1207	10	0	35	0	0.32
LA16	10 × 10	945	1021	978	3.49	945	0	945	0	961	1.69	945	0	945	127	0	945	17	1	33	0	0.24
LA17	10 × 10	784	796	787	0.38	784	0	784	0	787	0.38	784	0	784	127	0	784	23	2	29	0	0
LA18	10 × 10	848	891	859	1.3	848	0	848	0	848	0	848	0	848	127	0	848	22	1	34	0	0.17
LA19	10 × 10	842	875	860	2.14	842	0	842	0	850	0.95	842	0	842	127	0	842	21	4	43	0	0
LA20	10 × 10	902	924	914	1.33	902	0	907	0.55	907	0.55	902	0	907	127	0.55	907	19	2	36	0.55	1.24
LA21	15 × 10	1046	1172	1084	3.63	1046	0	1046	0	1088	4.02	1046	0	1047	387	0.1	1046	31	4	69	0	0.56
LA22	15 × 10	927	1040	944	1.83	927	0	935	0.86	986	6.36	927	0	927	863	0	927	45	6	54	0	0.35
LA23	15 × 10	1032	1061	1032	0	1032	0	1032	0	1032	0	1032	0	1032	92	0	1032	18	1	71	0	0.43
LA24	15 × 10	935	1000	976	4.39	938	0.32	953	1.93	970	3.74	935	0	938	766	0.32	935	57	8	37	0	0.89
LA25	15 × 10	977	1048	1017	4.09	979	0.2	986	0.92	1003	2.66	977	0	977	95	0	977	12	0	28	0	0.51
LA26	20 × 10	1218	1304	1224	0.49	1218	0	1218	0	1252	2.79	1218	0	1218	89	0	1218	15	0	74	0	0.62
LA27	20 × 10	1235	1325	1291	4.53	1235	0	1256	1.7	1299	5.18	1235	0	1236	1415	0.08	1236	87	14	65	0.08	2.1
LA28	20 × 10	1216	1256	1250	2.8	1216	0	1232	1.32	1289	6	1216	0	1216	476	0	1216	61	17	49	0	1.3
LA29	20 × 10	1157	1294	1239	7.09	1168	0.95	1196	3.37	1241	7.26	1163	0.52	1164	1442	0.61	1168	98	34	77	0.95	2.46
LA30	20 × 10	1355	1403	1355	0	1355	0	1355	0	1358	0.22	1355	0	1355	94	0	1355	16	1	82	0	0.26
LA31	30 × 10	1784	1784	1784	0	1784	0	1784	0	1784	0	1784	0	1784	56	0	1784	112	73	37	0	0.16
LA32	30 × 10	1850	1850	1850	0	1850	0	1850	0	1850	0	1850	0	1850	56	0	1850	96	61	51	0	0.54
LA33	30 × 10	1719	1719	1719	0	1719	0	1719	0	1719	0	1719	0	1719	62	0	1719	123	85	48	0	0.22
LA34	30 × 10	1721	1721	1721	0	1721	0	1721	0	1721	0	1721	0	1721	73	0	1721	164	99	39	0	0.67
LA35	30 × 10	1888	1888	1888	0	1888	0	1888	0	1888	0	1888	0	1888	100	0	1888	181	113	40	0	0.76
LA36	15 × 15	1268	1351	1305	2.92	1268	0	1279	0.87	1314	3.63	1268	0	1269	2473	0.08	1268	91	21	43	0	1.2
LA37	15 × 15	1397	1485	1423	1.86	1411	1	1408	0.79	1466	4.94	1397	0	1401	2512	0.29	1397	102	32	35	0	0.45
LA38	15 × 15	1196	1280	1255	4.93	1201	0.42	1219	1.92	1284	7.36	1201	0.42	1208	2586	1	1201	85	14	28	0.42	0.95
LA39	15 × 15	1233	1321	1273	3.24	1240	0.57	1246	1.05	1291	4.7	1233	0	1240	2492	0.57	1233	83	19	57	0	1.14
LA40	15 × 15	1222	1326	1269	3.85	1233	0.9	1241	1.55	1273	4.17	1224	0.16	1226	2534	0.33	1224	114	43	52	0.16	2.36
Average GAP (%)					1.4597		0.1091		0.4209		1.7481		0.0275			0.0980					0.0542	0.5593
No. of Instance					40		40		40		40		40			40					40	
No. of BKS Obtained					13		33		30		19		37			30					35	

۷. نتیجه‌گیری و پیشنهادهایی برای مطالعات آتی

در این مقاله مساله زمانبندی کار کارگاهی سنتی که بصورت $Jm||C_{\max}$ قابل نمایش است، مد نظر قرار گرفت. ابتدا مروری بر ادبیات مساله مورد بررسی صورت گرفت. سپس یک الگوریتم ترکیبی PSO برای حل مساله مورد بررسی توسعه داده شد.از آنجا که الگوریتم PSO برای مسایل با ماهیت پیوسته طراحی شده است، یک روش جدید نیز برای تبدیل جوابهای مساله زمانبندی که ماهیت گسسته دارند به جوابهای مورد پذیرش الگوریتم PSO با ماهیت پیوسته و بالعکس، با استفاده از تبدیل مبنای اعداد و بکارگیری مبنای فاکتوریلی اعداد یا همان شیوه نمایش اعداد در مبنای فاکتوریل (سیستم فاکتورادیک) توسعه داده شد که می‌تواند در سایر مسایل زمانبندی نیز به راحتی استفاده شود. پیچیدگی زمانی این تبدیل جدید چند جمله‌ای ($O(n)$) بوده و کارایی بالایی دارد. این روش به خوبی بر مساله مورد بررسی منطبق شده است. همچنین از آنجا که کیفیت جوابهای اولیه در همه الگوریتمهای تکاملی از جمله PSO از اهمیت بالایی برخوردار است، یک الگوریتم GRASP نیز توسعه داده شد و از یک تکرار آن برای تولید جوابهای (جمعیت) اولیه PSO استفاده شد. به منظور غلبه بر گرایش الگوریتم تکاملی PSO برای به دام افتادن در جواب‌های بهینه محلی نیز تغییراتی در رابطه بهنگام سازی سرعت ذرات داده شد و از ضرایب اینرسی (W) و انقباض (χ) همزمان استفاده شد و نتایج بهتری بدست آمد. همچنین برای جستجوی دقیق‌تر در اطراف نقاط بهینه محلی از یک الگوریتم جستجوی محلی بر اساس تعویض جهت کمانهای انفصالی موجود بر مسیر بحرانی در هر تکرار الگوریتم و یک الگوریتم آنیلینگ شبیه سازی شده در انتهای کار

الگوریتم PSO استفاده شد و در واقع الگوریتم مورد نظر به صورت الگوریتم ترکیبی (HPSO) مورد بررسی قرار گرفت.

نتایج محاسباتی کارایی الگوریتم GRASP ارایه شده را نسبت به جوابهای تصادفی نشان داد. همچنین کارایی الگوریتم پیشنهادی (HPSO) نسبت به سایر الگوریتمهای ارایه شده برای JSSP نیز با توجه به نتایج محاسباتی بر روی مسایل استاندارد حاصل شد. با توجه به زیاد بودن تعداد پارامترها در الگوریتم ترکیبی ارایه شده، انتظار می‌رود که با تنظیم دقیقتر پارامترها جوابهای بهتری حاصل شود که این مطلب می‌تواند موضوع تحقیقات آتی برای نویسندگان و یا مطالعه کنندگان این مقاله باشد. موضوع مهم دیگر بکارگیری الگوریتم ارایه شده در حل یک مساله واقعی است که از اهمیت بالایی برخوردار است و جا دارد که مطالعه‌ای در این زمینه صورت گیرد. همچنین ترکیب الگوریتم PSO ارایه شده با سایر روشهای فرا ابتکاری از جمله الگوریتم جستجوی ممنوع نیز می‌تواند مفید باشد.

مراجع

[1] Pinedo, M.L., *Planning and Scheduling in Manufacturing and Services.* Springer Series in Operations Research, Springer, New York, NY, USA, 2005.

[2] Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., *Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey.* Annals of Discrete Mathematics 5, 1979, pp. 287–326.

[3] Garey, M.R., Johnson, D.S., Sethi, R., *The Complexity of Flowshop and Jobshop Scheduling.* Mathematics of Operations Research 1, 1976, pp. 117-129.

[4] Adams, J., Balas, E., Zawack, D., *The Shifting Bottleneck Procedure for Job shop Scheduling*. Management Science 34, 1988, pp. 391–401.

[5] Pezzella, F., Merelli, E., *A Tabu Search Method Guided by Shifting Bottleneck for the Job Shop Scheduling Problem*. European Journal of Operational Research, Vol., 120, No. 2, 2000, pp. 297–310.

[6] Dominic, P.D.D., Kaliyamoorthy, S., Murugan, R., *A Conflict-Based Priority Dispatching Rule and Operation-Based Approaches to Job Shops.* International Journal of Advanced Manufacturing Technology 24, 2004, pp. 76–80.

[7] Park, B.J., Choi, H.R., Kim, H.S., *A Hybrid Genetic Algorithm for the Job Shop Scheduling Problems*. Computers & Industrial Engineering 45, 2003, pp. 597–613.

[8] Gonçalves, J.F., Mendes, J.J.M., Resende, M.G.C., *A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem.* European Journal of Operational Research, Vol. 167, No. 1, 2005, pp. 77–95.

[9] Ombuki, B., Ventresca, M., *Local Search Genetic Algorithms for the Job Shop Scheduling Problem*. Applied Intelligence 21, 2004, pp. 99–109.

[10] Steinhöfel, K., Albrecht, A., Wong, C.K., *Two Simulated Annealing-Based Heuristics for the Job Shop Scheduling Problem.* European Journal of Operational Research 118, 1999, pp. 524-548.

[11] Steinhöfel, K., Albrecht, A., Wong, C.K., *Fast parallel Heuristics for the Job Shop Scheduling Problem.* Computers & Operations Research 29, 2002, pp. 151-169.

[12] Steinhöfel, K., Albrecht, A., Wong, C.K., *An Experimental Analysis of Local Minima to Improve Neighbourhood Search*. Computers & Operations Research 30, 2003, pp. 2157-2173.

[13] Rossi, A., Dini, G., *Flexible Job-Shop Scheduling with Routing Flexibility and Separable Setup Times Using ant Colony Optimisation* method Robotics and Computer-Integrated Manufacturing 23, 2007, pp. 503–516.

[14] Xia, W.J., Wu, Z.M., *A Hybrid Particle Swarm Optimization Approach for the Job-Shop Scheduling Problem.* International Journal of Advanced Manufacturing Technology 29, 2006, pp. 360–366.

[15] Sha, D.Y., Hsu, C.-Y., *A Hybrid Particle Swarm Optimization for Job Shop Scheduling Problem Computers & Industrial Engineering* 51, 2006,pp. 791–808.

[16] Kennedy J., Eberhart R., *Particle Swarm Optimization*. In: Proceedings of the 1995 IEEE international conference on neural networks. New Jersey: IEEE Press; 1995, pp. 1942–1948.

[17] Eberhart R., Kennedy J., *A New Optimizer Using Particle Swarm Theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. Nagoya, Japan, 1995, pp. 39–43.

[18] Poli, R., Kennedy, J., Blackwell, T., *Particle Swarm Optimization: An Overview*. Swarm Intelligence, 1, 2007, pp. 33–57.

[19] Cheng, R., Gen, M., Tsujimura, Y., *A Tutorial Survey of Job-Shop Scheduling Problems Using Genetic Algorithms*, part I: representation. International Journal of Computers and Industrial Engineering, 30, 1996, pp. 983-997.

[20] McCaffrey, J., *Using Permutations in .NET for Improved Systems Security. Volt Information Sciences*, Inc. (It is online accessible at URL: http://msdn.microsoft.com/en-s/library/aa302371.aspx). 2003.

[21] Knuth, D., *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Third Edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997, pp. 65–66.

[22] Doliskani, J.N., Malekian, E., Zakerolhosseini, A., *A Cryptosystem Based on the Symmetric Group S_n*. International Journal of Computer Science and Network Security, Vol.8, No. 2, 2008, pp. 226-234.

[23] Balas, E., *Machine Sequencing Via Disjunctive Graphs: An Implicit Enumeration Approach*. Operations Research 17, 2008, pp. 941–957.

[24] Roy, S., Sussmann, B., *Les Problèmes D'ordonnancement Avec Constraintes Disjonctives*, SEMA, Note D.S., No. 9, Paris, 1964.

[25] Fisher, H., Thompson, G.L., *Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules*. Muth, J.F., Thompson, G.L., (eds.), Industrial Scheduling, Prentice Hall, Englewood Cliffs, New Jersey, 1963. pp. 225-251.

[26] Applegate, D., Cook, W., *A Computational Study of the Job-Shop Scheduling Problem*. ORSA Journal on Computing 3, 1991, pp. 149–156.

[27] Lawrance, S., *Resource Constrained Project Scheduling: an Experimental Investigation of Heuristic Scheduling Techniques. Dissertation*, Graduate School of Industrial Administration, Carnegie Mellon University,Pittsburgh, Pensylvania, 1984.

[28] Yamada, T., Nakano, R., *A Generic Algorithm Applicable to Large-Scale Job Shop Problems*. In: Manner R, Manderick B, editors. Parallel problem solving from nature II. Amsterdam: North-Holland, 281–90, 1992.

[29] Beasley, J.E., *OR-Library: Distributing Test Problems by Electronic Mail*. Journal of the Operational Research Society, Vol. 41, No. 11, 1990, pp. 1069–1072.