دانشکده ریاضی و علوم‌کامپیوتر

بررسی مسئله زمان‌بندی کارگاه باز

با استفاده از

الگوریتم بهینه‌سازی کلونی مورچگان

رشته علوم‌کامپیوتر

سعید سبزه

استاد راهنما:

دکتر جلیل رشیدی‌نیا

شهریور ماه 1403

فهرست

مقدمه

مسئله زمان‌بندی کارگاه باز (Open-Shop Scheduling Problem یا OSSP) یکی از موضوعات شناخته‌شده و مهم در حوزه مهندسی است که به دلیل کاربردهای گسترده صنعتی، اهمیت ویژه‌ای دارد[4] .OSSP به عنوان یکی از مسائل NP کامل، چالش‌های خاص خود را دارد و نسبت به سایر مسائل زمان‌بندی مانند زمان‌بندی کارگاه‌های جاب-شاپ و فلو-شاپ، فضای حل وسیع‌تری را در بر می‌گیرد[9] .

الگوریتم‌های فراابتکاری مانند ACO به دلیل کارایی بالا در حل مسائل بهینه‌سازی ترکیبیاتی، توجه بسیاری را به خود جلب کرده‌اند [5][14]. در این پروژه، با استفاده از الگوریتم بهینه‌سازی کلونی مورچه‌ها (Ant Colony Optimization یا ACO) به بررسی و حل مسئله OSSP پرداخته‌ایم. الگوریتم ACO ، که الهام‌گرفته از رفتار اجتماعی مورچه‌ها در طبیعت است [5]، به دلیل توانایی بالا در جستجو و یافتن راه‌حل‌های بهینه برای مسائل پیچیده، به عنوان ابزار قدرتمندی برای حل مسائل زمان‌بندی در نظر گرفته می‌شود.

با توجه به پیچیدگی و دامنه وسیع فضای جستجو در مسئلهOSSP ، کاربرد الگوریتم ACO می‌تواند به بهبود کیفیت و سرعت یافتن راه‌حل‌های بهینه کمک شایانی کند. در این گزارش، به معرفی روش‌شناسی ACO و نحوه پیاده‌سازی آن برای حل مسئله OSSPخواهیم پرداخت و نتایج به‌دست‌آمده را تحلیل خواهیم کرد.

مسئله زمان‌بندی کارگاه باز

- مسئله زمان‌بندی کارگاه باز (Open Shop Scheduling Problem) یک مسئله بهینه‌سازی ترکیبیاتی است که در آن مجموعه‌ای از کارها $(J=\{J_1,J_2,...,J_n\})$ باید توسط مجموعه‌ای از ماشین‌ها $(M=\{M_1,M_2,...,M_m\})$ پردازش شوند [13]. هر کار J_i از m عملیات O_{ij} تشکیل شده است که هر کدام نیاز به پردازش روی ماشین M_j دارند[4] .

تعریف مسئله:

- **ورودی‌ها:**
  - $J=\{J_1,J_2,...,J_n\}$: مجموعه‌ای از n کار.
  - $M=\{M_1,M_2,...,M_m\}$: مجموعه‌ای از m ماشین.
  - P_{ij} : مدت زمان پردازش عملیات O_{ij} که کار J_i روی ماشین M_j باید انجام شود.
- **محدودیت‌ها:**
  - هر ماشین Mj در هر لحظه می‌تواند تنها یک عملیات را پردازش کند.
  - هر کار Ji در هر لحظه می‌تواند تنها یک عملیات را اجرا کند.
  - ترتیب اجرای عملیات Oij روی ماشین‌ها دلخواه است و نیازی به رعایت ترتیب خاصی نیست.

- **هدف:**

  - **کمینه‌سازی زمان تکمیل کل :(Makespan)** هدف اصلی در اکثر نسخه‌های این مسئله کمینه‌سازی زمان تکمیل کل است که به صورت $C_{max} = max_{i,j}\{C_{i,j}\}$ تعریف می‌شود، که C_{ij} زمان تکمیل عملیات O_{ij} است.

مسئله زمان‌بندی کارگاه باز به دلیل انعطاف‌پذیری در ترتیب عملیات‌ها، نسبت به سایر مسائل زمان‌بندی مانند کارگاه جریان (Flow Shop) یا کارگاه کار (Job Shop) پیچیدگی بیشتری دارد و از نظر محاسباتی به عنوان یک *مسئله NP-سخت* شناخته می‌شود. به همین دلیل، روش‌های حل این مسئله اغلب شامل الگوریتم‌های ابتکاری، فراابتکاری و روش‌های تقریبی است.

	Machine 0	Machine 1	Machine 2	Machine 3	Machine 4
Job 0	80	3	65	98	9
Job 1	79	69	51	51	45
Job 2	65	37	75	53	91
Job 3	39	95	58	49	76
Job 4	54	29	27	68	93

جدول ۱.

یک مثال OSSP متشکل از ۵ دستگاه و ۵ کار [13]. هر عدد نمایش دهنده مدت‌زمان لازم جهت تکمیل کار مورد نظر روی ماشین مورد نظر است.

الگوریتم بهینه‌سازی کلونی مورچه‌ها

بهینه‌سازی کلونی مورچه‌ها (Ant Colony Optimization or ACO) یک الگوریتم فراابتکاری است که بر پایه رفتار طبیعی مورچه‌ها در جستجوی کوتاه ترین مسیر بین لانه و منابع غذایی طراحی شده است [7]. این الگوریتم برای حل مسائل بهینه‌سازی ترکیبیاتی، مانند مسئله فروشنده دوره‌گرد(TSP) ، مسئله زمان‌بندی و مسیریابی، استفاده می‌شود [7][5].

نحوه کار ACO :

- **الهام از رفتار مورچه‌ها:**

  - در طبیعت، مورچه‌ها هنگام جستجوی غذا به طور تصادفی حرکت می‌کنند، اما پس از پیدا کردن غذا، در مسیر خود به لانه، ماده‌ای شیمیایی به نام "فرمون" ترشح می‌کنند. مورچه‌های دیگر مسیری که مقدار بیشتری فرمون روی آن وجود دارد را به احتمال بیشتری نسبت به دیگر مسیرها دنبال می‌کنند [5].
  - با گذشت زمان، مسیرهایی که کوتاه‌تر و کارآمدتر هستند، فرمون بیشتری جذب می‌کنند و سایر مسیرهای ضعیف‌تر رفته‌رفته رها می‌شوند.

- **شبیه‌سازی در الگوریتم:**
  - درACO ، تعدادی "مورچه مصنوعی" ایجاد می‌شوند که در فضای مسئله حرکت می‌کنند. هر مورچه در هر مرحله یک حرکت انجام می‌دهد و مسیری را طی می‌کند.
  - انتخاب مسیر بر اساس احتمال‌هایی انجام می‌شود که تحت تأثیر میزان فرمون روی مسیرهای مختلف و دیگر پارامترهای مسئله است.
  - پس از طی مسیر، هر مورچه فرمونی روی مسیر طی شده خود ترشح می‌کند که متناسب با کیفیت مسیر است (مثلاً کوتاهی یا هزینه کمتر).
  - با تکرار این فرآیند، مسیرهای بهینه‌تر تشخیص داده می‌شوند و مورچه‌ها به تدریج این مسیرها را ترجیح می‌دهند.
- **بروز رسانی فرمون:**
  - فرمون‌ها به تدریج تبخیر می‌شوند تا از تثبیت زودهنگام الگوریتم بر روی مسیرهای محلی نامناسب جلوگیری شود.
  - به‌روزرسانی فرمون به مرور زمان و بر اساس کیفیت مسیرها انجام می‌شود.

پیاده سازی الگوریتم بهینه سازی کلونی مورچه‌ها برای برنامه‌ریزی کارگاه باز

برای استفاده از الگوریتم بهینه سازی کلونی مورچه‌ها در مسئله زمان‌بندی کارگاه باز(Open Shop Scheduling) ، لازم است مسئله را به صورت یک گراف ریاضی مدل‌سازی کنیم[8][11] . در این مدل‌سازی، هر عملیات O_{ij} که در آن i نشان‌دهنده کار (Job)و j نشان‌دهنده ماشین (Machine) است، به عنوان یک گره v_{ij} در گراف جهت‌دار کاملاً متصل $G=(V,E)$ در نظر گرفته می‌شود [6]. در این گراف:

- $V=\{v_{ij}\}$: مجموعه گره‌ها که نشان‌دهنده عملیات‌ها هستند.
- $E=\{(v_{ij},v_{kl})\}$: مجموعه یال‌های جهت‌دار که نشان‌دهنده ترتیب انجام عملیات‌ها هستند.

هدف مسئله، یافتن یک مسیر در این گراف است که تمام محدودیت‌های زمانی و اولویت‌ها را رعایت کرده و زمان کل تکمیل تمام کارها C_{max} را کمینه کند. هر مورچه k در سیستم مورچه‌ها یک مسیر π_k را در گراف طی می‌کند که نشان‌دهنده یک ترتیب

ممکن برای انجام عملیات‌ها است. احتمال انتخاب یال (v_{ij},v_{kl}) توسط مورچه k بر اساس مقدار فرومون $\tau_{ij,kl}$ موجود روی یال و یک تابع هوریستیک $\eta_{ij,kl}$ که به صورت معکوس زمان انجام عملیات O_{kl} تعریف می‌شود[6]، به صورت زیر است:

$$p_{ij}^{k}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[\eta_{ij}\right]^{\beta}}{\sum_{l \in J_i^k} \left[\tau_{il}(t)\right]^{\alpha} \cdot \left[\eta_{il}\right]^{\beta}},$$

- τ_{ij} مقدار فرومون بین عملیات i و j است.
- $\eta_{ij,kl}=1/t_{kl}$: معکوس زمان انجام عملیات O_{kl} است که t_{kl} مدت زمان اجرای آن است.
- α و β پارامتر کنترل‌کننده تأثیر فرومون و اطلاعات اکتشافی هستند.

پس از تکمیل مسیر توسط تمام مورچه‌ها، مقدار فرومون‌ها بر روی یال‌ها به‌روزرسانی می‌شود. فرمول به‌روزرسانی فرومون‌ها به صورت زیر است [5] :

$$\tau_{ij}(t) \leftarrow (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t),$$
$$\text{where } \Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t),$$

در این فرمول:

- ρ نرخ تبخیر فرومون است.$(0\leq\rho\leq1)$
- $\Delta\tau_{ij}^{k}$ میزان فرومون اضافه‌شده توسط مورچه k بر روی مسیر i به j است و به صورت زیر تعریف می‌شود.

$$\Delta\tau_{ij,kl}^{k} = \begin{cases} \frac{Q}{C_k} & if\ ant\ k\ traversed\ the\ edge\ (v_{ij}, v_{kl}) \\ 0 & otherwise \end{cases}$$

که در آن C_k زمان کل تکمیل مسیر π_k توسط مورچه k و Q یک ثابت است.

شبه کد الگوریتم **ACO** :

الگوریتم ۱: بهینه سازی کلونی مورچه ها**:**

/* مقداردهی اولیه فرمون‌ها */

برای هر زوج عملیات (**i, j**) انجام بده

$$\tau_{ij}(0)=\tau_0$$

پایان برای

/* حلقه اصلی */

برای نسل = 1 تا تعداد_نسل‌ها انجام بده:

/* تولید راه‌حل برای هر مورچه */

برای مورچه = 1 تا تعداد_مورچه‌ها انجام بده:

راه‌حل_مورچه را به‌صورت یک لیست خالی مقداردهی کن {

عملیات_فعلی = عملیات_آغازین

/* ساخت راه‌حل با انتخاب عملیات‌ها */

تا زمانی که عملیات‌های باقی‌مانده وجود دارند انجام بده:

احتمال‌ها را برای هر عملیات باقی‌مانده محاسبه کن

عملیات_بعدی را بر اساس احتمال‌های محاسبه‌شده انتخاب کن

عملیات_بعدی را به راه‌حل_مورچه اضافه کن

عملیات_فعلی را به عملیات_بعدی به‌روزرسانی کن

پایان تا

/* ارزیابی راه‌حل */

زمان_اتمام را برای راه‌حل_مورچه محاسبه کن (makespan)

زمان_اتمام را در هزینه_راه‌حل_مورچه‌ها[مورچه] ذخیره کن

پایان برای}

/* ارزیابی و به‌روزرسانی بهترین راه‌حل */

مورچه‌ای که حداقل زمان_اتمام را دارد پیدا کن

اگر راه‌حل فعلی بهتر از بهترین راه‌حل کلی است:

بهترین_راه‌حل و هزینه_بهترین_راه‌حل را به‌روزرسانی کن

پایان اگر

/* به‌روزرسانی سطح فرمون‌ها */

فرمون را بر روی همه مسیرها تبخیر کن

برای هر زوج عملیات **(i, j)** در بهترین_راه‌حل انجام بده:

فرمون τ_{ij} را به میزان یک پاداش افزایش بده

پایان برای

پایان برای

/* بهینه‌سازی راه‌حل‌ها با استفاده از جستجوی محلی */

در مسیرها جستجوی محلی انجام بده.

اگر راه‌حل فعلی بهتر از بهترین راه‌حل کلی است:

بهترین_راه‌حل و هزینه_بهترین_راه‌حل را به‌روزرسانی کن

پایان اگر

/* خروجی بهترین راه‌حل و هزینه آن */

بهترین_راه‌حل و هزینه_بهترین_راه‌حل را **برگردان**

پیاده‌سازی توابع کلیدی الگوریتم **ACO**

الگوریتم ۲: ایجاد راه‌حل(create_solution)

1. هر مورچه از یک مسیر خالی (π) شروع می‌کند.
2. در حالی که طول مسیر ($|\pi|$) کمتر از تعداد کل کارها (n) است، این مراحل را انجام بده:
   - یک عملیات (oj) که هنوز به مسیر (π) اضافه نشده را انتخاب کن.
   - عملیات (oj) را به مسیر (π) اضافه کن.
3. هزینه‌ی عملیات جدید را به هزینه کل مسیر (π) اضافه کن.
4. مسیر (π) را برگردان.

الگوریتم ۳: محاسبه‌ی زمانبندی(calculate_makespan)

i. لیست زمان‌بندی را برای هر ماشین به صورت خالی مقداردهی کن.

ii. برای هر عملیات در راه‌حل:

1. شماره کار و ماشین مربوط به عملیات فعلی را مشخص کن.
2. زمان اجرای این عملیات را از داده‌ها بگیر.
3. عملیات در حال اجرا (operation_executed) را به False مقداردهی کن.
4. در حالی که عملیات در حال اجرا نیست:

a) برای هر ماشین دیگر:

1. بررسی کن که آیا کار فعلی در حال اجرا بر روی ماشین دیگر است یا خیر.
2. اگر کار در حال اجرا بود:

1. زمان فعلی ماشین و زمان پایان کار روی ماشین دیگر را محاسبه کن.
2. اگر شرایط زمان‌بندی برای اجرای عملیات روی این ماشین مناسب بود:

1. عملیات در حال اجرا را به True تغییر بده.

b) پایان برای.

c) اگر عملیات در حال اجرا بود:

1. لیست زمان‌بندی ماشین فعلی را به‌روز کن و عملیات را به پایان برسان.

d) در غیر این صورت:

1. ماشین را برای اجرای عملیات منتظر بگذار.

5. پایان در حالی که

iii. بیشترین طول لیست‌های زمان‌بندی را به عنوان makespanبرگردان.

iv. پایان الگوریتم.

الگوریتم ۴: انتخاب عملیات بعدی(select_next_operation)

تابعی برای انتخاب عملیات بعدی که مورچه باید به آن مراجعه کند.

ورودی‌ها:

- current_operation_index : شاخص عملیات فعلی.
- remaining_operations : لیست عملیات‌هایی که هنوز بازدید نشده‌اند.

i. احتمالات انتخاب عملیات را با استفاده از مکانیزم مبتنی بر احتمال محاسبه کن.
ii. یک عملیات از میان عملیات‌های باقی‌مانده با توجه به احتمال‌های محاسبه‌شده انتخاب کن.
iii. عملیات انتخاب‌شده را بازگردان.
iv. پایان الگوریتم.

الگوریتم ۵: محاسبه احتمال‌های انتخاب عملیات(_calculate_operation_probabilities)

i. تابعی برای محاسبه احتمال‌های انتخاب عملیات‌های باقی‌مانده.

ii. ورودی‌ها:

- current_operation_index : شاخص عملیات فعلی.
- remaining_operations : لیست عملیات‌هایی که هنوز بازدید نشده‌اند.

iii. تعداد عملیات‌های باقی‌مانده را محاسبه کن.

iv. برای هر عملیات در remaining_operations :

a) مقدار τ (مقدار فرمون) را از ماتریس فرمون بگیر.

b) مقدار η (بر اساس زمان اجرای عملیات) را محاسبه کن. اگر شاخص عملیات فعلی برابر با تعداد کل عملیات باشد، η را ۱ قرار بده، در غیر این صورت مقدار η معکوس زمان اجرای عملیات روی ماشین مربوطه قرار بده.

c) مقدار احتمال انتخاب عملیات را با استفاده از τ و η محاسبه کن.

$$p_{ij}^{k}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[\eta_{ij}\right]^{\beta}}{\sum_{l \in J_i^k} \left[\tau_{il}(t)\right]^{\alpha} \cdot \left[\eta_{il}\right]^{\beta}},$$

v. احتمال‌های انتخاب عملیات را نرمال‌سازی کن تا مجموع آن‌ها برابر با ۱ شود.

vi. احتمال‌های نرمال‌شده را بازگردان.

الگوریتم ۶: پاداش‌دهی (rewards(s, k))

برای تمام جفت‌های عملیات o_i و o_j که در مسیر s ظاهر می‌شوند :

مقدار فرومون τ_{ij} را به اندازه K/C_s افزایش بده.

پایان برای.

الگوریتم ۷: به‌روزرسانی فرمون (update_pheromone)

تابعی برای به‌روزرسانی ماتریس فرمون بر اساس بهترین راه‌حل‌های یافت‌شده.

ورودی‌ها: هیچ ورودی‌ای لازم نیست.

عملیات:

$$\tau_{ij}(t) \leftarrow (1-\rho)\cdot\tau_{ij}(t) + \Delta\tau_{ij}(t)\,,$$
$$\text{where } \Delta\tau_{ij}(t) = \textstyle\sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t),$$

i. تبخیر فرمون:

مقدار فرمون در ماتریس فرمون را بر اساس **ρ** کاهش بده.

ii. پاداش دادن به بهترین راه‌حل کنونی:

با استفاده از تابع reward مقدار فرمون را با توجه به self.best_solution و self.current_best_solution_reward افزایش بده.

iii. پاداش دادن به بهترین راه‌حل کلی:

با استفاده از تابع reward مقدار فرمون را با توجه به self.ant_solutions[self.ibest] و self.overall_best_solution_reward افزایش بده.

پارامترهای کلیدی

برای بهینه‌سازی عملکرد الگوریتمACO ، تنظیم صحیح پارامترها ضروری است. در ادامه، توضیحات درباره پارامترهای اصلی که تأثیر قابل توجهی بر عملکرد الگوریتم دارند، ارائه می‌شود:

- **Alpha (α)**:

این پارامتر میزان اهمیت فرومون (τ) را در انتخاب مسیر توسط مورچه‌ها تعیین می‌کند. هرچه مقدار آلفا بیشتر باشد، تأثیر فرومون در تصمیم‌گیری مورچه‌ها بیشتر خواهد بود.

- **Beta (β)**:

این پارامتر میزان تأثیر تابع اکتشافی (θ) را در انتخاب مسیر مشخص می‌کند. مقدار بالاتر بتا نشان‌دهنده تأکید بیشتر بر اطلاعات اکتشافی در تعیین مسیر بهینه است.

انتخاب معمول این است که این دو پارامتر برابر و مقداری معادل 1 به آنها اختصاص داده شود. این تنظیم به مورچه‌ها اجازه می‌دهد تا به‌طور متوازن تحت تأثیر هر دو عامل، یعنی فرومون و اکتشاف، قرار گیرند. این روش به طور عمومی باعث می‌شود که مورچه‌ها تصمیماتی متعادل بگیرند و از وابستگی بیش از حد به یک عامل خاص جلوگیری کنند [7] .

- **Current_Best_Solution_Reward** :

این پارامتر نشان‌دهنده میزان پاداشی است که به بهترین راه‌حل یافت شده در تکرار جاری (Iteration_Best) تعلق می‌گیرد. پاداش‌دهی به این راه‌حل‌ها به معنی ترویج کاوش (***Exploration***) و شناسایی راه‌حل‌های جدید در فضای جستجو است.

- **Overall_Best_Solution_Reward :**

این پارامتر میزان پاداشی که به بهترین راه‌حل کلی تا کنون (Best_So_Far) داده می‌شود را تعیین می‌کند. پاداش‌دهی به این راه‌حل به معنی تقویت همگرایی (***Exploitation***) و استفاده از راه‌حل‌های موفق قبلی است.

مقادیر معمول برای این پارامترها، مانند 2/3 برای Reward ـSolution_Best_Overallو 1/3 برای Current_Best_Solution_Reward ، به الگوریتم کمک می‌کند تا تعادلی میان کاوش فضای جستجو و استفاده از اطلاعات به‌دست‌آمده برقرار کند. این تنظیمات به‌طور کلی باعث می‌شود که الگوریتم به‌طور مؤثری کاوش کند و در عین حال از تجربیات قبلی بهره ببرد.

- **Rho (ρ)**:

پارامتر **ρ** تعیین‌کننده سرعت بخار شدن فرومون‌ها در مسیرهای قبلاً طی‌شده است. این پارامتر تأثیر زیادی بر سرعت همگرایی الگوریتم دارد. اگر مقدار **ρ** بسیار کوچک باشد، فرومون‌ها به‌طور کندی بخار می‌شوند که ممکن است باعث شود الگوریتم به خوبی همگرا نشود، به ویژه اگر تعداد نسل‌ها کافی نباشد. برعکس، مقدار **ρ** بسیار بزرگ موجب می‌شود که فرومون‌ها به سرعت بخار شوند، که ممکن است باعث همگرایی سریع به راه‌حل‌های محلی و از دست دادن راه‌حل‌های بالقوه بهتری شود. انتخاب مقدار مناسب برای **ρ** نیازمند تعادل است تا الگوریتم به‌طور مؤثر همگرا شود بدون اینکه راه‌حل‌های خوب را نادیده بگیرد[2].

- **number_of_ants (تعداد مورچه‌ها)**:

تعداد مورچه‌ها در هر نسل، میزان کاوش در هر تکرار را تحت تأثیر قرار می‌دهد. افزایش تعداد مورچه‌ها به معنای تلاش‌های بیشتر در هر نسل و در نتیجه احتمال یافتن راه‌حل‌های بهتر است. معمولاً تعداد مورچه‌ها بین 20 تا 50 تنظیم می‌شود.

- **number_of_generations (تعداد نسل‌ها)**:

این پارامتر تعیین می‌کند که الگوریتم تا چند نسل ادامه یابد. افزایش تعداد نسل‌ها به الگوریتم فرصت بیشتری برای بهبود راه‌حل‌ها می‌دهد، اما زمان محاسباتی بیشتری نیز نیاز دارد.

برای مسائل بزرگ‌تر، تعداد بیشتر مورچه‌ها و نسل‌ها معمولاً به بهبود کیفیت راه‌حل‌ها کمک می‌کند، اما این امر به افزایش زمان محاسباتی نیز منجر می‌شود. در مقابل، برای مسائل کوچک‌تر، تعداد کمتری از مورچه‌ها و نسل‌ها کافی است و افزایش بیش از حد این پارامترها ممکن است به بهبود کیفیت راه‌حل‌ها کمک نکند.[1]

بهینه‌سازی راه‌حل‌ها با استفاده از جستجوی محلی تکرارشونده (ILS)

برای بهبود نتایج برنامه‌ریزی به‌دست‌آمده از الگوریتم بهینه‌سازی کلونی مورچه‌ها (ACO)، از روش جستجوی محلی تکرارشونده (Iterated Local Search) استفاده شده است. این روش به کمک الگوریتم جستجوی محلی، راه‌حل‌های بهتری را برای مسائل بهینه‌سازی تولید می‌کند[12] .

الگوریتم جستجوی محلی

جستجوی محلی یک تکنیک هوریستیکی است که برای حل مسائل بهینه‌سازی گسسته به کار می‌رود[10]. این روش با شروع از یک راه‌حل اولیه و کاوش در همسایگی آن، به دنبال بهبود راه‌حل می‌گردد. روند کار به شرح زیر است:

i. **راه‌اندازی اولیه:** الگوریتم با یک راه‌حل اولیه شروع می‌کند و زمان تکمیل (Makespan) آن را ارزیابی می‌کند.

ii. **کاوش همسایگی:** برای هر جفت عملیات، موقعیت‌های آنها جابه‌جا می‌شود تا راه‌حل‌های جدیدی تولید شود.

iii. **ارزیابی و بهبود:** زمان تکمیل راه‌حل جدید ارزیابی می‌شود. اگر زمان تکمیل جدید بهتر باشد، راه‌حل به‌روزرسانی شده و جستجو از نو آغاز می‌شود.

iv. **پایان:** این روند ادامه می‌یابد تا هیچ بهبودی بیشتر یافت نشود.

الگوریتم جستجوی محلی تکرارشونده

جستجوی محلی تکرارشونده بر پایه جستجوی محلی طراحی شده و با استفاده از چندین تکرار و اختلالات، به بهبود بیشتر نتایج کمک می‌کند [10] :

i. **مقدمه‌چینی:** الگوریتم با یک راه‌حل اولیه و ارزیابی زمان تکمیل آن شروع می‌شود.

ii. **اختلال:** یک تابع اختلال (perturbations)، راه‌حل جدیدی با انجام چندین جابجایی تصادفی از عملیات ها تولید می‌کند.

iii. **اعمال جستجوی محلی:** راه‌حل اختلال‌یافته تحت الگوریتم جستجوی محلی قرار می‌گیرد تا بیشتر بهبود یابد.

iv. **مقایسه و به‌روزرسانی:** اگر راه‌حل بهبود یافته زمان تکمیل بهتری داشته باشد، به‌عنوان بهترین راه‌حل جدید پذیرفته می‌شود. در غیر این صورت، شمارش تلاش‌های ناموفق افزایش می‌یابد و الگوریتم با اختلالات جدید ادامه می‌یابد.

v. **پایان:** پس از تعداد معینی از تلاش های ناموفق بدون بهبود، به پایان می رسد.

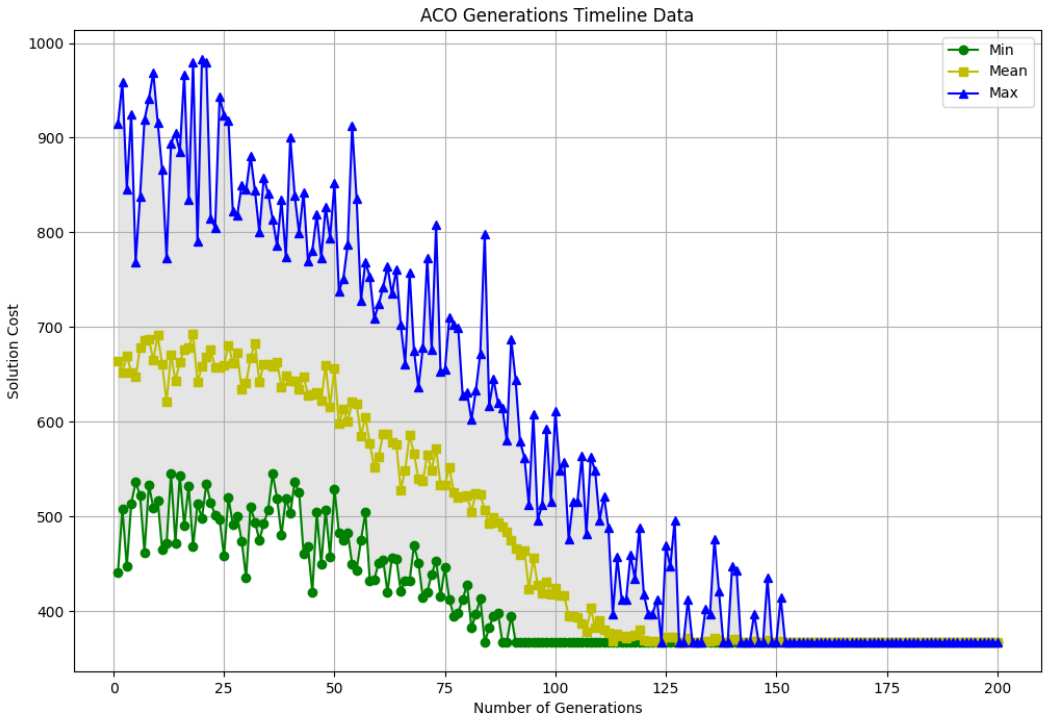
بررسی پیشرفت الگوریتم **ACO** در حل دو نمونه مسئله**OSSP** :

مقادیر پارامترهای به‌کاررفته در اجرای این الگوریتم به صورت زیر تنظیم شده‌اند:

- **Alpha (α)**: 1
- **Beta (β)**: 1
- **Rho (ρ)**: 0.1
- **Initial Pheromone (τ)**: 1
- **Overall Best Solution Reward**: 2 / 3
- **Current Best Solution Reward**: 1 / 3
- **Number of Ants**: 25
- **Number of Generations**: 200

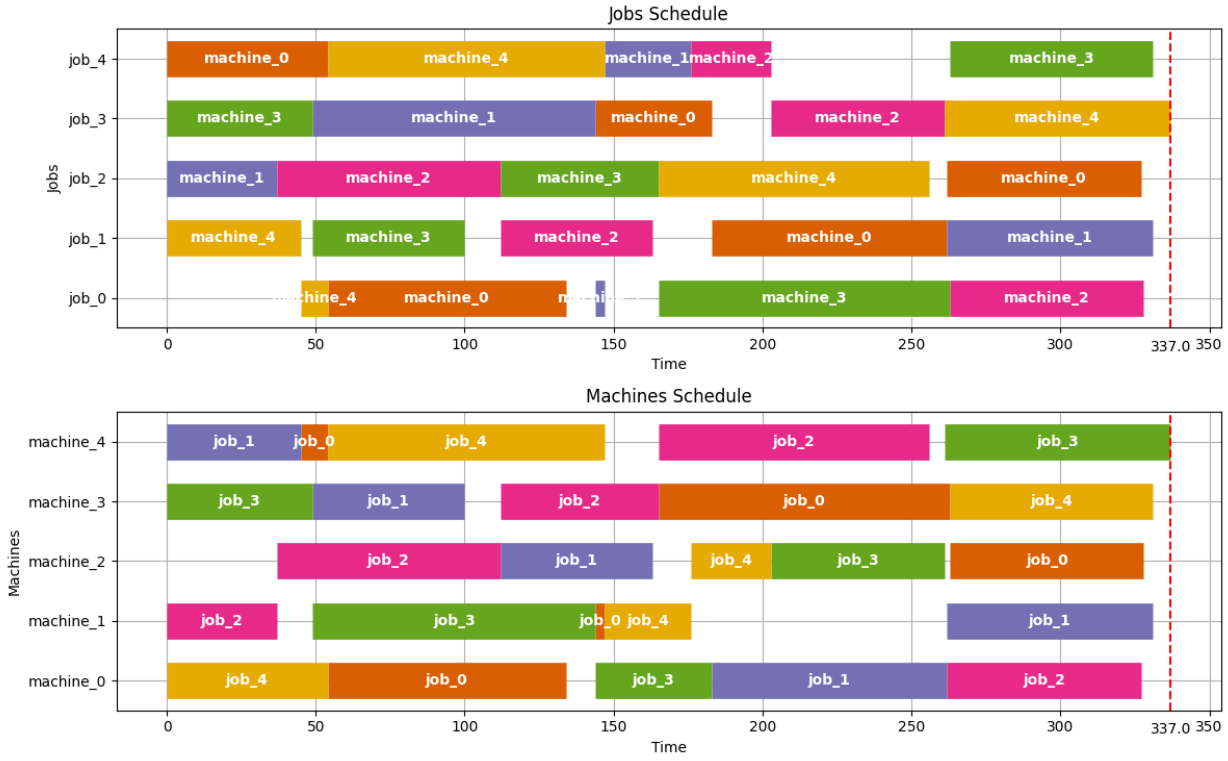
مثال ۱: حل مسئله OSSP جدول۱

نمودار زیر پیشرفت الگوریتم ACO در مسئله OSSP که در جدول ۱ ارائه شده و در جدول ۳ تحت عنوان "5x5 -3" آمده است، را نمایش می‌دهد:



نمودار 1

نمودار زیر، حل به‌دست‌آمده و ترتیب اجرای عملیات‌های کارها بر روی ماشین‌ها را نشان می‌دهد .

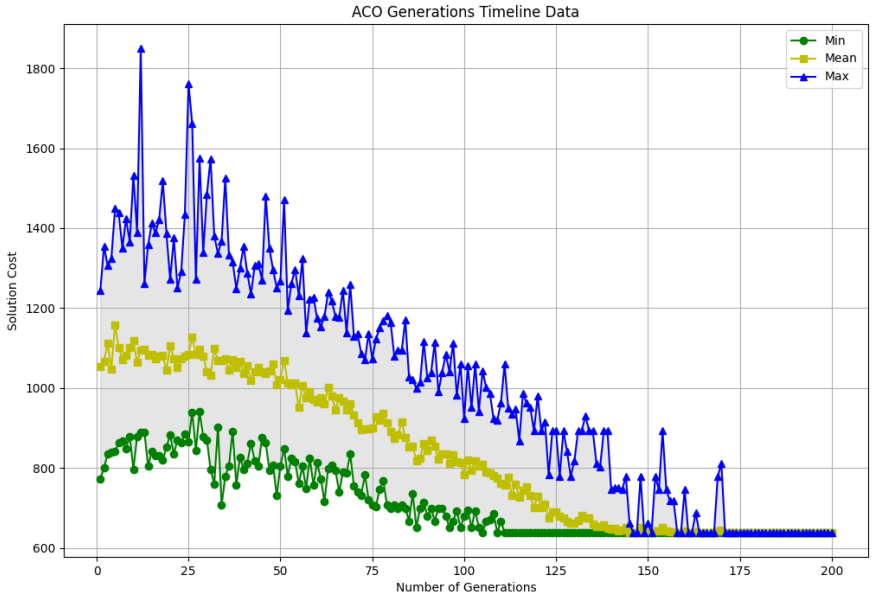


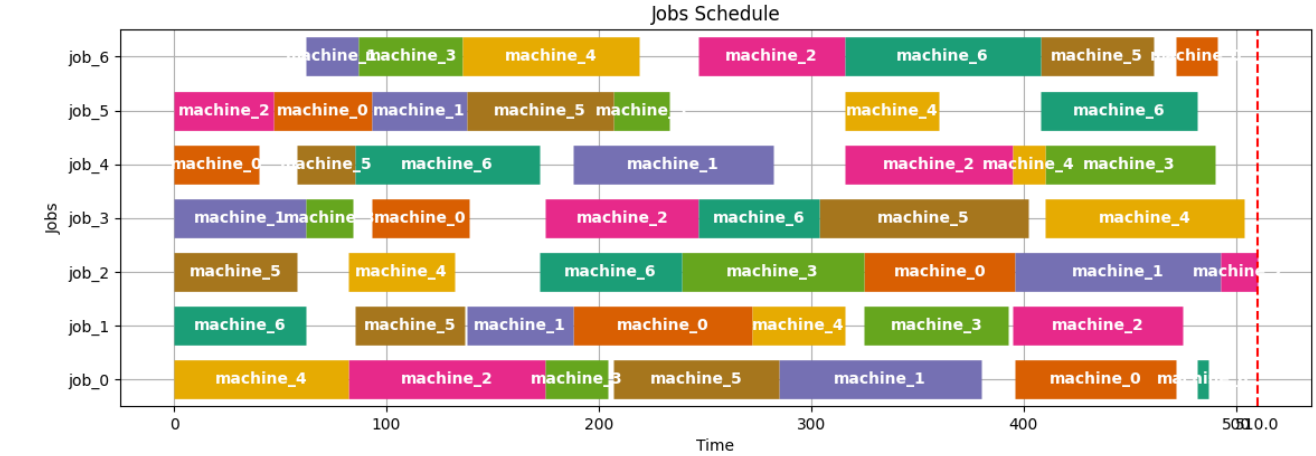
نمودار 2

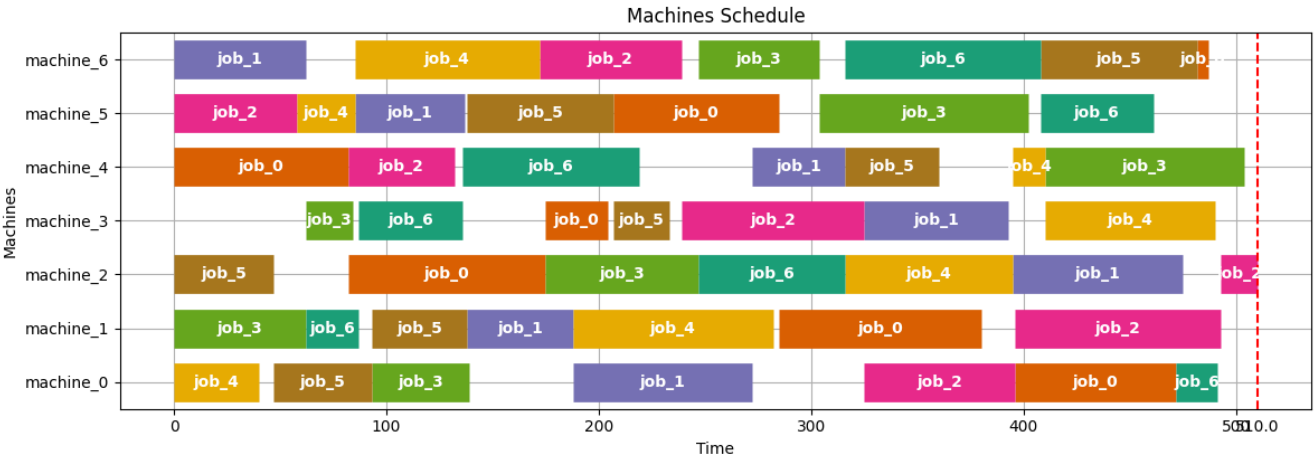
زمان مورد نیاز برای حل مسئله

- زمان مورد نیاز برای یافتن راه‌حل با استفاده ازACO : 25.66 ثانیه
- زمان مورد نیاز برای بهبود راه‌حل با استفاده از جستجوی محلی: 26.79 ثانیه

مثال۲: یک مسئله OSSP با ۷ ماشین و ۷ کار







نتایج به‌دست‌آمده از آزمون‌های آزمایشی

در این بخش، نتایج حاصل از آزمون‌های مختلف برنامه‌ریزی کارگاه باز (Open Shop Scheduling) با ابعاد متفاوت که در پژوهش E. Taillard تحت عنوان "Benchmarks for Basic Scheduling Problems" منتشر شده است[14] ، ارائه می‌شود. اطلاعات مربوط به این آزمون‌ها از طریق لینک در دسترس است.

در جداول زیر، نتایج به‌دست‌آمده از اجرای الگوریتم بهینه‌سازی مبتنی بر کلونی مورچه‌ها (ACO) که پیش‌تر توصیف شد، آورده شده است. این جداول شامل بهترین راه‌حل‌های ارائه‌شده در کار Taillard و نیز بهترین نتایج حاصل از ترکیب ACO و جستجوی محلی (Local Search) به همراه زمان مورد نیاز برای اجرای کامل الگوریتم است.

مقادیر پارامترهای به‌کاررفته در آزمایش به‌صورت زیر است:

- **Alpha (α)**: 1
- **Beta (β)**: 1
- **Rho (ρ)**: 0.1
- **Initial Pheromone (τ)**: 1
- **Overall Best Solution Reward**: 2 / 3
- **Current Best Solution Reward**: 1 / 3
- **Number of Ants**: 25
- **Number of Generations**: 200

جدول نتایج آزمایش‌های ۴ کار و ۴ ماشین

جدول زیر نتایج حاصل از آزمایش‌های انجام‌شده برای مسئله ۴کار و ۴ ماشین را نمایش می‌دهد:

example	Upper Bound	*ACO*	Final (*+ Local Search*)
4 x 4 – 1	*193*	*193*	*193*
4 x 4 – 2	*236*	*252*	*241*
4 x 4 – 3	*271*	*278*	*271*
4 x 4 – 4	*250*	*263*	*253*
4 x 4 – 5	*295*	*305*	*295*
4 x 4 – 6	*189*	*194*	*193*
4 x 4 – 7	*201*	*209*	*203*
4 x 4 – 8	*217*	*227*	*220*
4 x 4 – 9	*261*	*268*	*267*
4 x 4 – 10	*217*	*221*	*221*

جدول ۲

جدول نتایج آزمایش‌های ۵ کار و ۵ ماشین

جدول زیر نتایج حاصل از آزمایش‌های انجام‌شده برای مسئله ۵ کار و ۵ ماشین را نمایش می‌دهد:

example	Upper Bound	ACO	Final (+ Local Search)
5 x 5 – 1	*300*	*322*	*300*
5 x 5 – 2	*262*	*283*	*266*
5 x 5 – 3	*328*	*367*	*337*
5 x 5 – 4	*310*	*374*	*332*
5 x 5 – 5	*329*	*400*	*348*

جدول۳

جدول نتایج آزمایش‌های ۷ کار و ۷ ماشین

جدول زیر نتایج حاصل از آزمایش‌های انجام‌شده برای مسئله ۷ کار و ۷ ماشین را نمایش می‌دهد:

example	Upper Bound	ACO	Final (+ Local Search)
7 x 7 - 1	*438*	*567*	*477*
7 x 7 – 2	*449*	*523*	*505*
7 x 7 – 3	*479*	*638*	*510*
7 x 7 – 4	*467*	*565*	*489*

جدول ۴

منابع

[1] Arnaout, J. P., Musa, R., & Rabadi, G. (2010). "Ant colony optimization algorithm to parallel machine scheduling problem with setups." *International Journal of Production Economics*, 123(2), 175-182.

[2] Blum, C. (2005). "Ant colony optimization: Introduction and recent trends." *Physics of Life Reviews*, 2(4), 353-373.

[3] Blum, C., & Li, X. (2008). "Swarm intelligence in optimization." *Natural Computing*, 7(3), 375-376.

[4] Blum, C., & Roli, A. (2003). "Metaheuristics in combinatorial optimization: Overview and conceptual comparison." *ACM Computing Surveys (CSUR)*, 35(3), 268-308.

[5] Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press.

[6] Dorigo, M., & Gambardella, L. M. (1997). "Ant colonies for the traveling salesman problem." *Biosystems*, 43(2), 73-81.

[7] Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization*. MIT Press.

[8] Dorigo, M., Birattari, M., & Stützle, T. (2006). "Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique." *IEEE Computational Intelligence Magazine*, 1(4), 28-39.

[9] Ghosh, I. K., Tiwari, M. K., & Smith, J. R. F. (2012). "A review of ant colony optimization metaheuristic and its applications in industrial engineering." *Advances in Engineering Software*, 43(5), 555-569.

[10] Hoos, H. H., & Stützle, T. (2004). *Stochastic Local Search: Foundations & Applications*. Elsevier.

[11] Lin, Y.-F., Cheng, W.-C., & Lin, J.-C. (2011). "Ant Colony Optimization for Shop Scheduling Problems." *Journal of Intelligent Manufacturing*, 22(6), 823-837.

[12] Schaller, J. E. (2004). "Ant Colony Optimization in Discrete Event Simulation Modeling." *Simulation Modelling Practice and Theory*, 12(3-4), 156-172.

[13] Taillard, É. D. (1993). "Benchmarks for basic scheduling problems." *European Journal of Operational Research*, 64(2), 278-285.

[14] Talbi, E. G. (2009). *Metaheuristics: From Design to Implementation*. John Wiley & Sons.